# IEC 61158-4-2

Edition 3.0 2014-08

# INTERNATIONAL STANDARD

# NORME INTERNATIONALE

## Industrial communication networks – Fieldbus specifications – Part 4-2: Data-link layer protocol specification – Type 2 elements

## Réseaux de communication industriels – Spécifications des bus de terrain – Partie 4-2: Spécification du protocole de la couche liaison de données – Eléments de type 2

IEC 61158-4-2:2014-08(en-fr)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch


**About the IEC**
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

**About IEC publications**
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

**IEC Catalogue - webstore.iec.ch/catalogue**
The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

**IEC publications search - www.iec.ch/searchpub**
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, replaced and withdrawn publications.

**IEC Just Published - webstore.iec.ch/justpublished**
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

**Electropedia - www.electropedia.org**
The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 14 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

**IEC Glossary - std.iec.ch/glossary**
More than 55 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

**IEC Customer Service Centre - webstore.iec.ch/csc**
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

---

**A propos de l'IEC**
La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

**A propos des publications IEC**
Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

**Catalogue IEC - webstore.iec.ch/catalogue**
Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

**Recherche de publications IEC - www.iec.ch/searchpub**
La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,…). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

**IEC Just Published - webstore.iec.ch/justpublished**
Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

**Electropedia - www.electropedia.org**
Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 14 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

**Glossaire IEC - std.iec.ch/glossary**
Plus de 55 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

**Service Clients - webstore.iec.ch/csc**
Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

# IEC 61158-4-2

Edition 3.0 2014-08

# INTERNATIONAL STANDARD

# NORME INTERNATIONALE

## Industrial communication networks – Fieldbus specifications – Part 4-2: Data-link layer protocol specification – Type 2 elements

## Réseaux de communication industriels – Spécifications des bus de terrain – Partie 4-2: Spécification du protocole de la couche liaison de données – Eléments de type 2

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE XH
CODE PRIX

ICS 25.040.40; 35.100.20; 35.110

ISBN 978-2-8322-1720-7

# CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

# INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

## Part 4-2: Data-link layer protocol specification – Type 2 elements

## FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

Attention is drawn to the fact that the use of the associated protocol type is restricted by its intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a layer protocol type to be used with other layer protocols of the same type, or in other type combinations explicitly authorized by its intellectual-property-right holders.

NOTE Combinations of protocol types are specified in IEC 61784-1 and IEC 61784-2.

International Standard IEC 61158-4-2 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial process measurement, control and automation.

This third edition cancels and replaces the second edition published in 2010. This edition constitutes a technical revision.

The main changes with respect to the previous edition are listed below.

- Addition of conventions in 3.6
- Updates of ControlNet object in 7.2
- Addition of missing V/NV attribute characteristic in 7.5, 7.6, 7.7
- Extensions and clarifications of TCP/IP interface object in 7.5
- Extensions and clarifications of Ethernet Link object in 7.6
- Extensions and clarifications of CCO object in 7.8
- Extensions and updates of DLR object in 7.9
- Updates of QoS object in 7.10
- Addition of Port object in 7.11
- Updates to DL state machines in 8.1 and 9.2
- Extensions and updates of DLR protocol in Clause 10
- Update of indicator behaviour in A.2.2 and A.2.3
- Miscellaneous editorial corrections

The text of this standard is based on the following documents:

| FDIS | Report on voting |
|---|---|
| 65C/762/FDIS | 65C/772/RVD |

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with ISO/IEC Directives, Part 2.

A list of all parts of the IEC 61158 series, published under the general title *Industrial communication networks – Fieldbus specifications*, can be found on the IEC web site.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

**IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.**

# INTRODUCTION

This part of IEC 61158 is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the "three-layer" fieldbus reference model described in IEC 61158-1.

The data-link protocol provides the data-link service by making use of the services available from the physical layer. The primary aim of this standard is to provide a set of rules for communication expressed in terms of the procedures to be carried out by peer data-link entities (DLEs) at the time of communication. These rules for communication are intended to provide a sound basis for development in order to serve a variety of purposes:

a) as a guide for implementers and designers;

b) for use in the testing and procurement of equipment;

c) as part of an agreement for the admittance of systems into the open systems environment;

d) as a refinement to the understanding of time-critical communications within OSI.

This standard is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this standard together with other standards positioned within the OSI or fieldbus reference models, otherwise incompatible systems may work together in any combination.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of patents given in several subclauses as indicated in the table below. These patents are held by their respective inventors under license to ODVA, Inc:

| | | | |
|---|---|---|---|
| US 5,400,331 | [ODVA] | Communication network interface with screeners for incoming messages | Subclause 3.4, Clauses 4 to 9 |
| US 5,471,461 | [ODVA] | Digital communication network with a moderator station election process | |
| US 5,491,531 | [ODVA] | Media access controller with a shared class message delivery capability | |
| US 5,493,571 | [ODVA] | Apparatus and method for digital communications with improved delimiter detection | |
| US 5,537,549 | [ODVA] | Communication network with time coordinated station activity by time slot and periodic interval number | |
| US 5,553,095 | [ODVA] | Method and apparatus for exchanging different classes of data during different time intervals | |
| US 8,244,838 | [ODVA] | Industrial controller employing the network ring topology | Clause 10 |

IEC takes no position concerning the evidence, validity and scope of these patent rights.

ODVA and the holders of these patent rights have assured the IEC that ODVA is willing to negotiate licences either free of charge or under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of ODVA and the holders of these patent rights is registered with IEC. Information may be obtained from:

[ODVA] ODVA, Inc.
2370 East Stadium Boulevard #1000
Ann Arbor, Michigan 48104
USA
Attention: Office of the Executive Director
e-mail: odva@odva.org

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. IEC shall not be held responsible for identifying any or all such patent rights.

ISO (www.iso.org/patents) and IEC (http://patents.iec.ch) maintain on-line databases of patents relevant to their standards. Users are encouraged to consult the databases for the most up to date information concerning patents.

# INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

# Part 4-2: Data-link layer protocol specification – Type 2 elements

## 1 Scope

### 1.1 General

The data-link layer provides basic time-critical messaging communications between devices in an automation environment.

This protocol provides communication opportunities to all participating data-link entities, sequentially and in a cyclic synchronous manner. Foreground scheduled access is available for time-critical activities together with background unscheduled access for less critical activities.

Deterministic and synchronized transfers can be provided at cyclic intervals up to 1 ms and device separations of 25 km. This performance is adjustable dynamically and on-line by re-configuring the parameters of the local link whilst normal operation continues. By similar means, DL connections and new devices may be added or removed during normal operation.

This protocol provides means to maintain clock synchronization across an extended link with a precision better than 10 μs.

This protocol optimizes each access opportunity by concatenating multiple DLSDUs and associated DLPCI into a single DLPDU, thereby improving data transfer efficiency for data-link entities that actively source multiple streams of data.

The maximum system size is an unlimited number of links of 99 nodes, each with 255 DLSAP-addresses. Each link has a maximum of $2^{24}$ related peer and publisher DLCEPs.

### 1.2 Specifications

This standard specifies

a) procedures for the timely transfer of data and control information from one data-link user entity to a peer user entity, and among the data-link entities forming the distributed data-link service provider;

b) the structure of the fieldbus DLPDUs used for the transfer of data and control information by the protocol of this standard, and their representation as physical interface data units.

### 1.3 Procedures

The procedures are defined in terms of

a) the interactions between peer DL-entities (DLEs) through the exchange of fieldbus DLPDUs;

b) the interactions between a DL-service (DLS) provider and a DLS-user in the same system through the exchange of DLS primitives;

c) the interactions between a DLS-provider and a Ph-service provider in the same system through the exchange of Ph-service primitives.

### 1.4 Applicability

These procedures are applicable to instances of communication between systems which support time-critical communications services within the data-link layer of the OSI or fieldbus reference models, and which require the ability to interconnect in an open systems interconnection environment.

Profiles provide a simple multi-attribute means of summarizing an implementation's capabilities, and thus its applicability to various time-critical communications needs.

### 1.5 Conformance

This standard also specifies conformance requirements for systems implementing these procedures. This standard does not contain tests to demonstrate compliance with such requirements.

## 2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE All parts of the IEC 61158 series, as well as IEC 61784-1 and IEC 61784-2 are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this list of normative references.

IEC 61131-3, *Programmable controllers – Part 3: Programming languages*

IEC 61158-3-2:2014, *Industrial communication networks – Fieldbus specifications – Part 3-2: Data-link layer service definition – Type 2 elements*

IEC 61158-5-2:2014, *Industrial communication networks – Fieldbus specifications – Part 5-2: Application layer service definition – Type 2 elements*

IEC 61158-6-2:2014, *Industrial communication networks – Fieldbus specifications – Part 6-2: Application layer protocol specification – Type 2 elements*

IEC 61588:2009, *Precision clock synchronization protocol for networked measurement and control systems*

IEC 61784-3-2, *Industrial communication networks – Profiles – Part 3-2: Functional safety fieldbuses – Additional specifications for CPF 2*

IEC 62026-3:2008, *Low-voltage switchgear and controlgear – Controller-device interfaces (CDIs) – Part 3: DeviceNet*

ISO/IEC 3309[1], *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures – Frame structure*

ISO/IEC 7498-1, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing*

_____

1 This standard has been withdrawn.

ISO/IEC 8802-3, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*

ISO 11898:1993[2], *Road vehicles – Interchange of digital information – Controller area network (CAN) for high-speed communication*

IEEE 802.1D-2004, *IEEE standard for local and metropolitan area networks – Media Access Control (MAC) bridges,* available at <http://www.ieee.org>

IEEE 802.1Q-2005[2], *IEEE standard for local and metropolitan area networks – Virtual bridged local area networks,* available at <http://www.ieee.org>

IEEE 802.3-2008, *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications,* available at <http://www.ieee.org>

IETF RFC 951, *Bootstrap Protocol (BOOTP)*, available at <http://www.ietf.org>

IETF RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, available at <http://www.ietf.org>

IETF RFC 1542, *Clarifications and Extensions for the Bootstrap Protocol*, available at <http://www.ietf.org>

IETF RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types*, available at <http://www.ietf.org>

IETF RFC 2131, *Dynamic Host Configuration Protocol*, available at <http://www.ietf.org>

IETF RFC 2132, *DHCP Options and BOOTP Vendor Extensions*, available at <http://www.ietf.org>

IETF RFC 4541, *Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches*, available at <http://www.ietf.org>

IETF RFC 5227:2008, *IPv4 Address Conflict Detection*, available at <http://www.ietf.org>

## 3 Terms, definitions, symbols, abbreviations and conventions

For the purposes of this document, the following terms, definitions, symbols and abbreviations apply.

### 3.1 Reference model terms and definitions

This standard is based in part on the concepts developed in ISO/IEC 7498-1 and ISO/IEC 7498-3, and makes use of the following terms defined therein.

**3.1.1 called-DL-address** [ISO/IEC 7498-3]

**3.1.2 calling-DL-address** [ISO/IEC 7498-3]

---

2 A newer edition of this standard has been published, but only the cited edition applies.

| | | |
|---|---|---|
| **3.1.3** | **centralized multi-end-point-connection** | [ISO/IEC 7498-1] |
| **3.1.4** | **correspondent (N)-entities**<br>**correspondent DL-entities (N=2)**<br>**correspondent Ph-entities (N=1)** | [ISO/IEC 7498-1] |
| **3.1.5** | **demultiplexing** | [ISO/IEC 7498-1] |
| **3.1.6** | **DL-address** | [ISO/IEC 7498-3] |
| **3.1.7** | **DL-address-mapping** | [ISO/IEC 7498-1] |
| **3.1.8** | **DL-connection** | [ISO/IEC 7498-1] |
| **3.1.9** | **DL-connection-end-point** | [ISO/IEC 7498-1] |
| **3.1.10** | **DL-connection-end-point-identifier** | [ISO/IEC 7498-1] |
| **3.1.11** | **DL-connection-mode transmission** | [ISO/IEC 7498-1] |
| **3.1.12** | **DL-connectionless-mode transmission** | [ISO/IEC 7498-1] |
| **3.1.13** | **DL-data-sink** | [ISO/IEC 7498-1] |
| **3.1.14** | **DL-data-source** | [ISO/IEC 7498-1] |
| **3.1.15** | **DL-duplex-transmission** | [ISO/IEC 7498-1] |
| **3.1.16** | **DL-facility** | [ISO/IEC 7498-1] |
| **3.1.17** | **DL-local-view** | [ISO/IEC 7498-3] |
| **3.1.18** | **DL-name** | [ISO/IEC 7498-3] |
| **3.1.19** | **DL-protocol** | [ISO/IEC 7498-1] |
| **3.1.20** | **DL-protocol-connection-identifier** | [ISO/IEC 7498-1] |
| **3.1.21** | **DL-protocol-control-information** | [ISO/IEC 7498-1] |
| **3.1.22** | **DL-protocol-data-unit** | [ISO/IEC 7498-1] |
| **3.1.23** | **DL-protocol-version-identifier** | [ISO/IEC 7498-1] |
| **3.1.24** | **DL-relay** | [ISO/IEC 7498-1] |
| **3.1.25** | **DL-service-connection-identifier** | [ISO/IEC 7498-1] |
| **3.1.26** | **DL-service-data-unit** | [ISO/IEC 7498-1] |
| **3.1.27** | **DL-simplex-transmission** | [ISO/IEC 7498-1] |
| **3.1.28** | **DL-subsystem** | [ISO/IEC 7498-1] |
| **3.1.29** | **DL-user-data** | [ISO/IEC 7498-1] |
| **3.1.30** | **flow control** | [ISO/IEC 7498-1] |
| **3.1.31** | **layer-management** | [ISO/IEC 7498-1] |
| **3.1.32** | **multiplexing** | [ISO/IEC 7498-3] |
| **3.1.33** | **naming-(addressing)-authority** | [ISO/IEC 7498-3] |
| **3.1.34** | **naming-(addressing)-domain** | [ISO/IEC 7498-3] |

**3.1.35 naming-(addressing)-subdomain** [ISO/IEC 7498-3]

**3.1.36 (N)-entity** [ISO/IEC 7498-1]
**DL-entity**
**Ph-entity**

**3.1.37 (N)-interface-data-unit** [ISO/IEC 7498-1]
**DL-service-data-unit (N=2)**
**Ph-interface-data-unit (N=1)**

**3.1.38 (N)-layer** [ISO/IEC 7498-1]
**DL-layer (N=2)**
**Ph-layer (N=1)**

**3.1.39 (N)-service** [ISO/IEC 7498-1]
**DL-service (N=2)**
**Ph-service (N=1)**

**3.1.40 (N)-service-access-point** [ISO/IEC 7498-1]
**DL-service-access-point (N=2)**
**Ph-service-access-point (N=1)**

**3.1.41 (N)-service-access-point-address** [ISO/IEC 7498-1]
**DL-service-access-point-address (N=2)**
**Ph-service-access-point-address (N=1)**

**3.1.42 peer-entities** [ISO/IEC 7498-1]

**3.1.43 Ph-interface-control-information** [ISO/IEC 7498-1]

**3.1.44 Ph-interface-data** [ISO/IEC 7498-1]

**3.1.45 primitive name** [ISO/IEC 7498-3]

**3.1.46 reassembling** [ISO/IEC 7498-1]

**3.1.47 recombining** [ISO/IEC 7498-1]

**3.1.48 reset** [ISO/IEC 7498-1]

**3.1.49 responding-DL-address** [ISO/IEC 7498-3]

**3.1.50 routing** [ISO/IEC 7498-1]

**3.1.51 segmenting** [ISO/IEC 7498-1]

**3.1.52 sequencing** [ISO/IEC 7498-1]

**3.1.53 splitting** [ISO/IEC 7498-1]

**3.1.54 synonymous name** [ISO/IEC 7498-3]

**3.1.55 systems-management** [ISO/IEC 7498-1]

## 3.2 Service convention terms and definitions

This standard also makes use of the following terms defined in ISO/IEC 10731 as they apply to the data-link layer:

**3.2.1 acceptor**

**3.2.2 asymmetrical service**

**3.2.3 confirm (primitive);**

**requestor.deliver (primitive)**

**3.2.4 deliver (primitive)**

**3.2.5 DL-confirmed-facility**

**3.2.6 DL-facility**

**3.2.7 DL-local-view**

**3.2.8 DL-mandatory-facility**

**3.2.9 DL-non-confirmed-facility**

**3.2.10 DL-provider-initiated-facility**

**3.2.11 DL-provider-optional-facility**

**3.2.12 DL-service-primitive;**
**primitive**

**3.2.13 DL-service-provider**

**3.2.14 DL-service-user**

**3.2.15 DL-user-optional-facility**

**3.2.16 indication (primitive)**
**acceptor.deliver (primitive)**

**3.2.17 multi-peer**

**3.2.18 request (primitive);**
**requestor.submit (primitive)**

**3.2.19 requestor**

**3.2.20 response (primitive);**
**acceptor.submit (primitive)**

**3.2.21 submit (primitive)**

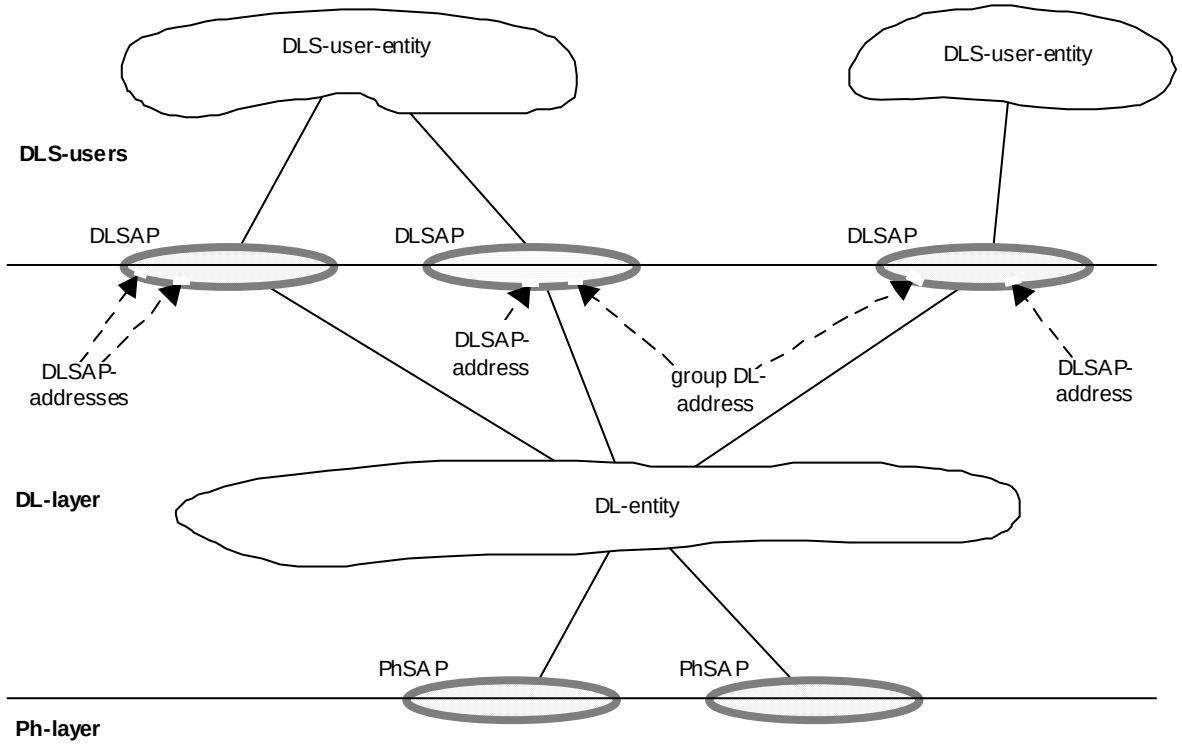
**3.2.22 symmetrical service**

## 3.3 Common terms and definitions

For the purposes of this document, the following terms and definitions apply.

NOTE Many definitions are common to more than one protocol Type; they are not necessarily used by all protocol Types.

**3.3.1**
**DL-segment**
**link**
**local link**
single DL-subnetwork in which any of the connected DLEs may communicate directly, without any intervening DL-relaying, whenever all of those DLEs that are participating in an instance of communication are simultaneously attentive to the DL-subnetwork during the period(s) of attempted communication




NOTE 1 DLSAPs and PhSAPs are depicted as ovals spanning the boundary between two adjacent layers.

NOTE 2 DL-addresses are depicted as designating small gaps (points of access) in the DLL portion of a DLSAP.

NOTE 3 A single DL-entity can have multiple DLSAP-addresses and group DL-addresses associated with a single DLSAP.

**Figure 1 – Relationships of DLSAPs, DLSAP-addresses and group DL-addresses**

**3.3.2**
**DLSAP**
distinctive point at which DL-services are provided by a single DL-entity to a single higher-layer entity

Note 1 to entry: This definition, derived from ISO/IEC 7498-1, is repeated here to facilitate understanding of the critical distinction between DLSAPs and their DL-addresses. (See Figure 1.).

**3.3.3**
**DL(SAP)-address**
either an individual DLSAP-address, designating a single DLSAP of a single DLS-user, or a group DL-address potentially designating multiple DLSAPs, each of a single DLS-user

Note 1 to entry: This terminology is chosen because ISO/IEC 7498-3 does not permit the use of the term DLSAP-address to designate more than a single DLSAP at a single DLS-user.

**3.3.4**
**(individual) DLSAP-address**
DL-address that designates only one DLSAP within the extended link

Note 1 to entry: A single DL-entity may have multiple DLSAP-addresses associated with a single DLSAP.

**3.3.5**
**extended link**
DL-subnetwork, consisting of the maximal set of links interconnected by DL-relays, sharing a single DL-name (DL-address) space, in which any of the connected DL-entities may communicate, one with another, either directly or with the assistance of one or more of those intervening DL-relay entities

Note 1 to entry: An extended link may be composed of just a single link.

**3.3.6**
**frame**
denigrated synonym for DLPDU

**3.3.7**
**group DL-address**
DL-address that potentially designates more than one DLSAP within the extended link. A single DL-entity may have multiple group DL-addresses associated with a single DLSAP. A single DL-entity also may have a single group DL-address associated with more than one DLSAP

**3.3.8**
**node**
single DL-entity as it appears on one local link

**3.3.9**
**receiving DLS-user**
DL-service user that acts as a recipient of DL-user-data

Note 1 to entry: A DL-service user can be concurrently both a sending and receiving DLS-user.

**3.3.10**
**sending DLS-user**
DL-service user that acts as a source of DL-user-data

### 3.4 Additional Type 2 definitions

**3.4.1**
**actual packet interval**
**API**
measure of how frequently a specific connection produces its Lpacket data

**3.4.2**
**allocate**
take a resource from a common area and assign that resource for the exclusive use of a specific entity

**3.4.3**
**application**
function or data structure for which data is consumed or produced

**3.4.4**
**application objects**
multiple object classes that manage and provide the run time exchange of messages across the network and within the device

**3.4.5**
**attribute**
description of an externally visible characteristic or feature of an object

Note 1 to entry: The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behavior of an object. Attributes are divided into class attributes and instance attributes.

**3.4.6**
**behavior**
indication of how the object responds to particular events

Note 1 to entry: Its description includes the relationship between attribute values and services.

**3.4.7**
**bit**
unit of information consisting of a 1 or a 0

Note 1 to entry: This is the smallest data unit that can be transmitted.

**3.4.8**
**blanking or blanking time**
length of time required after transmitting before the node is allowed to receive

**3.4.9**
**client**
1) An object which uses the services of another (server) object to perform a task
2) An initiator of a message to which a server reacts

**3.4.10**
**communication objects**
components that manage and provide run time exchange of messages across the network such as the ControlNet object, the Unconnected Message Manager (UCMM)

**3.4.11**
**connection**
logical binding between two application objects within the same or different devices

**3.4.12**
**connection ID**
**CID**
identifier assigned to a transmission, associated with a particular connection between producers and consumers, that identifies a specific piece of application information

Note 1 to entry: Within the data link this is a generic tag.

**3.4.13**
**cyclic redundancy check**
**CRC**
residual value computed from an array of data and used as a representative signature for the array

**3.4.14**
**cyclic**
term used to describe events which repeat in a regular and repetitive manner

**3.4.15**
**deafness**
situation where the node can not hear the moderator DLPDU but can hear other link traffic

**3.4.16**
**device**
physical hardware connection to the link

Note 1 to entry: A device may contain more than one node.

**3.4.17**
**DLPDU**
data-link protocol data unit

Note 1 to entry: A DLPDU consists of a source MAC ID, zero or more Lpackets and an FCS as transmitted or received by an associated PhE.

**3.4.18**
**end delimiter**
unique set of M_symbols that identifies the end of a DLPDU

**3.4.19**
**error**
discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

**3.4.20**
**extended link**
links connected by DL-routers (sometimes called a local area network)

**3.4.21**
**FCS error**
error that occurs when the computed frame check sequence value after reception of all the octets in a DLPDU does not match the expected residual

**3.4.22**
**fixed tag**
two octet identifier (tag) which identifies a specific service to be performed by either

a) that receiving node on the local link which has a specified MAC ID, or

b) all receiving nodes on the local link

Note 1 to entry: Identification of the target node(s) is included in the two octet tag.

**3.4.23**
**frame**
denigrated synonym for DLPDU

**3.4.24**
**Frame Check Sequence**
**FCS**
redundant data derived from a block of data within a DLPDU (frame), using a hash function, and stored or transmitted together with the block of data, in order to detect data corruption

**3.4.25**
**generic tag**
three octet identifier (tag) which identifies a specific piece of application information

**3.4.26**
**guardband**
time slot allocated for the transmission of the moderator DLPDU

**3.4.27**
**implicit token**
mechanism that governs the right to transmit

Note 1 to entry: No actual token message is transmitted on the medium. Each node keeps track of the MAC ID of the node that it believes currently holds the right to transmit. The right to transmit is passed from node to node by keeping a record of the node that last transmitted. A slot time is used to allow a missing node to be skipped in the rotation

**3.4.28**
**implicit token register**
register that contains the MAC ID of the node that holds the right to transmit

**3.4.29**
**instance**
actual physical occurrence of an object within a class, identifying one of many objects within the same object class

EXAMPLE California is an instance of the object class state.

Note 1 to entry: The terms object, instance, and object instance are used to refer to a specific instance.

**3.4.30**
**instance attribute**
attribute that is unique to an object instance and not shared by the object class

**3.4.31**
**instantiated**
object that has been created in a device

**3.4.32**
**Keeper**
object responsible for distributing link configuration data to all nodes on the link

**3.4.33**
**link**
collection of nodes with unique MAC IDs, attached to Ph-segments connected by Ph-repeaters

**3.4.34**
**little endian**
model of memory organization which stores the least significant octet at the lowest address, and of transmission organization which transfers the least significant octet first on the medium

**3.4.35**
**Lpacket**
well-defined sub-portion of a DLPDU consisting of

a) size and control information

b) a fixed tag or a generic tag, and

c) DLS-user data or, when the tag has DL-significance, DL-data

**3.4.36**
**M_symbol(s)**
symbols that represent the data bits and related information encoded and transmitted by the PhL

**3.4.37**
**maximum scheduled node**
node with highest MAC ID that can use scheduled time on a link

**3.4.38**
**maximum unscheduled node**
node with highest MAC ID that can use unscheduled time on a link

**3.4.39**
**Message Router**
object within a node that distributes messaging requests to the appropriate application objects

**3.4.40**
**moderator**
node with the lowest MAC ID that is responsible for transmitting the moderator DLPDU

**3.4.41**
**moderator DLPDU**
DLPDU transmitted by the node with the lowest MAC ID for the purpose of synchronizing the nodes and distributing the link configuration parameters

**3.4.42**
**multipoint connection**
connection from one node to many

Note 1 to entry: Multipoint connections allow messages from a single producer (publisher) to be received by many consumer (subscriber) nodes.

**3.4.43**
**network**
series of nodes connected by some type of communication medium

Note 1 to entry: The connection paths between any pair of nodes can include repeaters, routers and gateways.

**3.4.44**
**network access monitor**
**NAM**
component within a node that manages communication with a temporary node connected to the nodes local NAP

**3.4.45**
**network access port**
**NAP**
PhL variant that allows a temporary node to be connected to the link by connection to the NAP of a permanent node

**3.4.46**
**network address or node address**
node's address on the link

Note 1 to entry: This is also called MAC ID.

**3.4.47**
**network status indicators**
indicators on a node indicating the status of the Physical and Data Link Layers

**3.4.48**
**network update time**
**NUT**
repetitive time interval in which data can be sent on the link

**3.4.49**
**node**
logical connection to a local link, requiring a single MAC ID

Note 1 to entry: A single physical device may appear as many nodes on the same local link. For the purposes of this protocol, each node is considered to be a separate DLE.

**3.4.50**
**non-concurrence**
situation where a transmission is received from an unexpected MAC ID, which appears to violate the time based access protocol

Note 1 to entry: This may occur when a connection is made between two working links that are not synchronized with each other but who have the same configuration information.

**3.4.51**
**non-data symbol**
PhL symbol which violates the requirements of Manchester biphase L encoding

**3.4.52**
**object**
abstract representation of a particular component within a device

Note 1 to entry: An object can be

1) an abstract representation of a computer's capabilities. Objects can be composed of any or all of the following components:
   a) data (information which changes with time);
   b) configuration (parameters for behavior);
   c) methods (things that can be done using data and configuration);
2) a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behavior.

**3.4.53**
**object specific service**
service defined by a particular object class to perform a required function which is not performed by a common service

Note 1 to entry: An object specific service is unique to the object class which defines it.

**3.4.54**
**originator**
client responsible for establishing a connection path to the target

**3.4.55**
**permanent node**
node whose connection to the network does not utilize the network access port (NAP) PhL variant

Note 1 to entry: This node may optionally support a NAP PhL variant to allow temporary nodes to connect to the network.

**3.4.56**
**produce**
act of sending data to be received by a consumer

**3.4.57**
**producer**
node that is responsible for sending data

**3.4.58**
**redundant media**
system using more than one medium to help prevent communication failures

**3.4.59**
**requested packet interval**
**RPI**
measure of how frequently the originating application requires the transmission of Lpackets or data packets from the target application

**3.4.60**
**rogue**
node that has received a moderator DLPDU that disagrees with the link configuration currently used by this node

**3.4.61**
**scheduled**
data transfers that occur in a deterministic and repeatable manner on predefined NUTs

**3.4.62**
**server**
object which provides services to another (client) object

**3.4.63**
**service**
operation or function than an object and/or object class performs upon request from another object and/or object class

Note 1 to entry: A set of common services is defined and provisions for the definition of object-specific services are provided. Object-specific services are those which are defined by a particular object class to perform a required function which is not performed by a common service.

**3.4.64**
**slot time**
maximum time required for detecting an expected transmission

Note 1 to entry: Each node waits a slot time for each missing node during the implied token pass.

**3.4.65**
**start delimiter**
unique set of M_symbols that identifies the beginning of a DLPDU

**3.4.66**
**supernode**
DLE with a MAC ID of zero

Note 1 to entry: This DLE DL-address is reserved for special DLL functions

**3.4.67**
**table unique identifier**
**TUI**
unique reference code used by ControlNet and Keeper objects to reference a consistent set of link configuration attribute

**3.4.68**
**tag**
shorthand name for a specific piece of application information, two or three octets in length

**3.4.69**
**target**
end-node to which a connection is established

**3.4.70**
**temporary node**
same as transient node

**3.4.71**
**tMinus**
number of NUTs before a new set of link configuration parameters are to be used

**3.4.72**
**tone**
instant of time which marks the boundary between two NUTs

**3.4.73**
**tool**
executable software program which interacts with the user to perform some function

EXAMPLE Link scheduling software (LSS).

**3.4.74**
**transaction id**
field within a UCMM header that matches a response with the associated request, which a server echoes in the response message

**3.4.75**
**transient node**
node that is only intended to be connected to the network on a temporary basis using the NAP PhL medium connected to the NAP of a permanent node

**3.4.76**
**Unconnected Message Manager**
**UCMM**
component within a node that transmits and receives unconnected explicit messages and sends them directly to the Message Router object

**3.4.77**
**unconnected service**
messaging service which does not rely on the set up of a connection between devices before allowing information exchanges

**3.4.78**
**unscheduled**
data transfers that use the remaining allocated time in the NUT after the scheduled transfers have been completed

**3.4.79**
**vendor ID**
identification of each product manufacturer/vendor by a unique number

Note 1 to entry: Vendor IDs are assigned by ODVA, Inc.

## 3.5 Type 2 symbols and abbreviations

| | | |
|---|---|---|
| **ACD** | Address Conflict Detection | |
| **ACM** | Access control machine | |
| **BOOTP** | Bootstrap Protocol | [IETF RFC 951] |
| **CA** | Clock accuracy | |
| **CAN** | Controller area network | [ISO 11898:1993] |
| **COP** | Connection originator password | |
| **DHCP** | Dynamic Host Configuration Protocol | [IETF RFC 2131] |
| **DLR** | Device level ring | |
| **DSCP** | DiffServ code point | |
| **FCS** | Frame Check Sequence | |
| **IP** | Internet Protocol | [IETF RFC 791] |
| **LED** | Light emitting diode. | |
| **LSS** | Link scheduling software | |
| **MAC ID** | MAC address of a node | |
| **MSTP** | Multiple spanning tree protocol | [IEEE 802.1Q] |
| **NAM** | Network attachment monitor | |
| **NAP** | Network access port | |
| **ND** | Non-data symbol | |
| **NUT** | Network update time | |
| **PT** | Programming terminal (a temporary network connection) | |
| **QoS** | Quality of service | |
| **Rcv** | Receive | |
| **RPI** | Requested Lpacket interval | |
| **RSTP** | Rapid spanning tree protocol | [IEEE 802.1D] |
| **Rx** | Receive | |
| **RxLLC** | Receive logical link control | |
| **RxM** | Receive machine | |
| **SEM** | State event matrix | |
| **SMAX** | MAC ID of the maximum scheduled node | |
| **STD** | State transition diagram, used to describe object behavior | |
| **STP** | Spanning tree protocol | [IEEE 802.1D] |
| **TCP** | Terminal Control Protocol | [IETF RFC 793] |
| **TFTP** | Trivial File Transfer Protocol | [IETF RFC 1350] |
| **TUI** | Table unique identifier | |
| **Tx** | Transmit | |
| **TxLLC** | Transmit logical link control | |
| **TxM** | Transmit machine | |
| **UCCM** | Unconnected Message Manager | |
| **UMAX** | MAC ID of maximum unscheduled node | |
| **USR** | Unscheduled start register | |

### 3.6 Conventions for station management objects

Objects for station management are specified in Clause 7, using attributes, services and behaviour.

Object attributes are specified in tables formatted according to Table 1.

**Table 1 – Format of attribute tables**

| Attri-bute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| See a) | See b) | See c) | See d) | See e) | See 6.1.3. | See f) | See g) |

a) “Attribute ID” is the identification value assigned to an attribute.

b) “Need in implementation” specifies whether or not the attribute is necessary in the implementation. An attribute may be optional, required or conditional.

A conditional attribute is required if certain object behaviours and/or attributes are implemented as defined by the class.

If an Attribute is optional, then a default value shall be defined in the semantics. An optional attribute's default value shall provide the same behaviour as if the attribute was not implemented. If the default value of an optional attribute does not match the behaviour of the object when the object does not implement the attribute, the object definition shall declare the behaviour.

In all cases, the term “default” indicates a “factory default” as shipped from the vendor.

c) “Access rule” specifies how a requestor can access an attribute. The definitions for access rules are:

- Settable (Set) – The attribute shall be accessed by at least one of the set services. If the behaviour of the device does not require a set service, then a device implementation of that object is not required to implement the attribute as settable.

  Settable attributes, unless otherwise specified by the object definition, shall be also accessed by get services.

- Gettable (Get) – The attribute shall be accessed by at least one of the get services.

d) “NV” indicates whether an attribute value is maintained through power cycles. This column is used in object definitions where non-volatile storage of attribute values is required. An entry of “NV” indicates value shall be saved, ‘”V” means not saved.

e) Name refers to the attribute.

f) Description of Attribute provides general information about the attribute.

g) Semantics of Values specifies the meaning of the value of the attribute.

## 4 Overview of the data-link protocol

### 4.1 General

#### 4.1.1 DLL architecture

The Type 2 DLL is modeled as an integrated Access Control Machine (ACM), with scheduling support functions, designed for reliable and efficient support of higher-level connection-mode and connectionless data transfer services. Interoperating with these higher level functions are DLL management functions.

PhL framing and delimiters are managed by DLL functions for serializing and deserializing M_symbol requests and indications.

Within the Data Link Layer, the Access Control Machine (ACM) has the primary responsibility for detecting and recovering from network disruptions. The major objectives of the ACM are

– to make sure that the local node detects and fully utilizes its assigned access slots in the schedule;

– to make sure that the local node does not interfere with the transmissions of other nodes, especially the moderator node;

– to start the next NUT (see 4.1.2) on schedule, whether the moderator DLPDU is heard or not;

– if the local node is the moderator, to transmit each moderator DLPDU strictly on schedule.

The Data Link Layer is comprised of the components listed in Table 2:

**Table 2 – Data-link layer components**

| Components | Description |
|---|---|
| **Access Control Machine (ACM)** | receives and transmits control DLPDUs and header information, and determines the timing and duration of transmissions. |
| **Transmit LLC (TxLLC)** | buffers DLSDUs received from Station Management and the DLS-user, and determines which should be transmitted next. |
| **Receive LLC (RxLLC)** | performs the task of quarantining received link units until they are validated by a good FCS. |
| **Transmit Machine (TxM)** | receives requests to send DLPDU headers and trailers, and Lpackets from the ACM, and breaks them down into octet symbol requests that are sent to the Serializer. |
| **Receive Machine (RxM)** | assembles received Lpackets from octet symbols received from the Deserializer, and submits them to the RxLLC. |
| **Serializer** | receives octet symbols, encodes and serializes them, and sends them as M_symbols to the PhL. It is also responsible for generating the FCS. |
| **Deserializer** | receives M_symbols from the PhL, converts M_symbols into octets and sends them to the receive machine. It is also responsible for checking the FCS. |
| **DLL Management Interface** | holds the Station Management variables that belong to the DLL, and helps manage synchronized changes of the link parameters. |

The internal arrangement of these components, and their interfaces, are shown in Figure 2. The arrowheads illustrate the primary direction of flow of data and control.

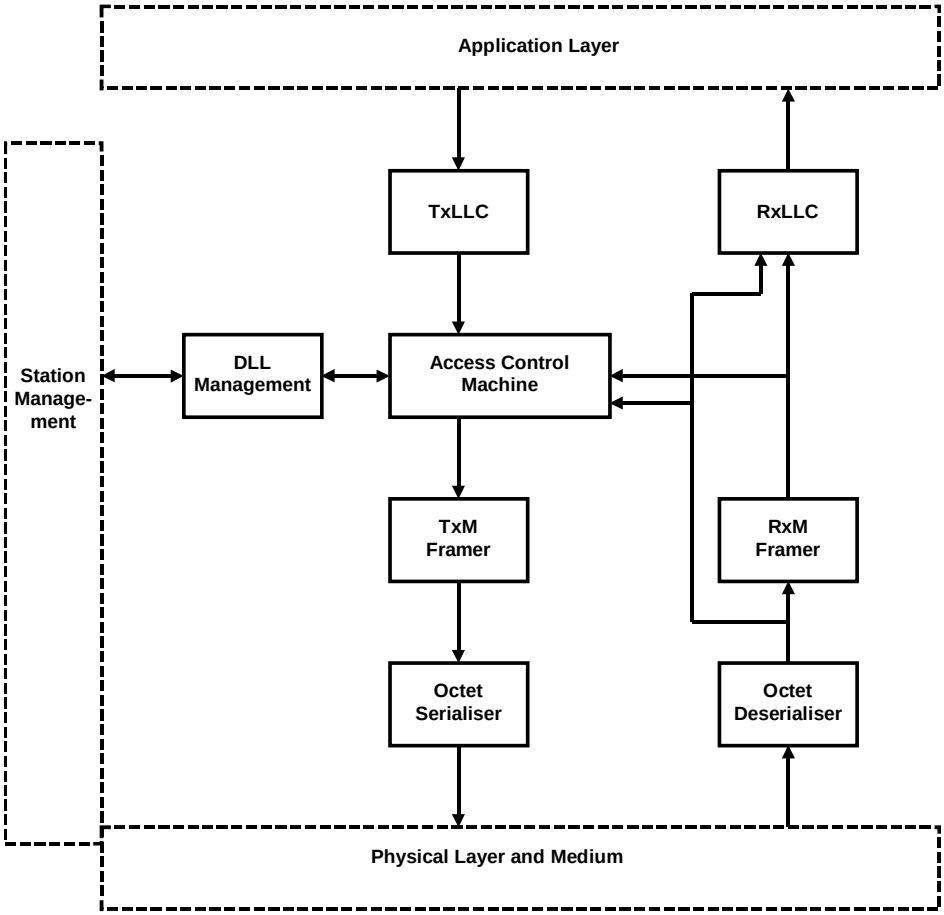



**Figure 2 – Data-link layer internal architecture**

### 4.1.2 Access control machine (ACM) and scheduling support functions

The ACM functions schedule all communication from one DLE to another. The timing of this communication are regulated at two levels,

1) to provide an accurate time based distribution of scheduled opportunities for designated communications in accord with, and with a pre-established schedule for, the local link and any extended network of links,
2) to provide a democratic distribution of unscheduled opportunities for arbitrary communications, generally in a cyclic but asynchronous manner.

Accurate schedule timing as provided by 1) is very important to support many control and data collection tasks in the applications domain of this protocol. The schedule is based on a fixed, repetitive time cycle called network update time (NUT). The NUT is maintained in close synchronism among all nodes participating in the DLL and used to provide two types of access opportunity;

1) One opportunity for each active station to transfer one scheduled DLPDU containing multiple scheduled DLSDUs.

   Only DLSDUs with scheduled as their QoS parameter may be included in a scheduled DLPDU.

   NOTE DLSDUs with scheduled as their QoS parameter are usually connection mode transfers with generic as their tag parameter.

2) One opportunity for at least one active station to send one background (unscheduled) DLPDU containing multiple DLSDUs of any QoS type. If time is available within a NUT, one background MAC opportunity is offered to each other active station in order of their MAC ID address. The station MAC ID for the first background opportunity in each NUT, is

incremented ( +1 modulo the set of background addresses) for each successive NUT to democratically share the available time among active stations.

Support procedures are included for automatic transfer to backup scheduling services and to manage the use of redundant Ph media.

#### 4.1.3 Connection-mode, connectionless-mode data transfer and DL service

The connection mode, connectionless mode and DL services provide the necessary functionality for

– managing DL provider interactions with the DL user by converting request and response primitives into necessary DLE operations and generating indications and confirm primitives where appropriate,

– determination of the DL address and control information to be added to each DLSDU,

– local confirmation of status for each outgoing DLSDU,

– DLPDU formation by concatenation of multiple DL user DLSDUs for efficient transfer as single PhPDUs, subject to DLPDU size limitations, QoS parameters and the type of access opportunity.

### 4.2 Services provided by the DL

#### 4.2.1 Overview

The DLL provides connectionless data transfer services and connection-mode data transfer services for limited-size DLSDUs, an internally synchronized time service, scheduling services to control the time allocation of the underlying shared PhL service, a DL(SAP)-address and queue management service, and a packing/unpacking service that combines multiple DLSDUs into a single data transfer DLPDU for efficient use of each access opportunity.

#### 4.2.2 QoS

##### 4.2.2.1 Definition

The QoS parameter specifies priority options for DLSDUs and access opportunities. The available values are as follows:

##### 4.2.2.2 DLL scheduled priority

The scheduled priority QoS provides accurate time based cyclic and acyclic sending of DLSDUs. The execution timing for this fieldbus scheduled service can be accurate and repeatable to better than 1 ms. Scheduled priority is normally used with connected mode services.

NOTE The master Keeper is responsible for regular publication of a unique table identifier (TUI) which ensures all nodes have a reference for the current set of link operating parameters. This TUI publication uses a fixed tag and is sent with QoS priority set to scheduled.

##### 4.2.2.3 DLL high priority (unscheduled)

The high priority QoS provides acyclic sending of DLSDUs with a bounded upper time for the sending delay. Data on this priority is sent only when all scheduled data has been sent and a non-scheduled sending opportunity is available.

##### 4.2.2.4 DLL low priority (unscheduled)

The low priority QoS provides for sending of DLSDUs only on an as-available basis. Data on this priority is sent only when all other data priorities have been sent and a non-scheduled sending opportunity is available.

NOTE High and Low priority are used only in a local sense to set the order of servicing among locally submitted, unscheduled DLS-user-data.

## 4.3 Structure and definition of DL-addresses

### 4.3.1 General

DL-addresses are used as MAC ID addresses (Link node designators), Fixed tag addresses (DLSAP-addresses and group DL-addresses), and Generic tag addresses (DLCEP-addresses).

Subclause 4.3 defines the form and structure of DL-addresses and includes specific definitions for some standard DL-addresses.

All addresses are formed of octets. Each octet shall be transferred to the Ph layer low-bit first and multiple octets shall be transferred low-order octet first (little endian format).

Three types of DL addresses are used.

### 4.3.2 MAC ID address

MAC ID addresses identify physical nodes on the local link as shown in Figure 3.

Permitted values are within the range 0 to 99, values from 100 to 254 are reserved.

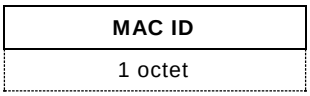



**Figure 3 – Basic structure of a MAC ID address**

Table 3 specifies the use of MAC ID addresses.

**Table 3 – MAC ID addresses allocation**

| MAC ID | Meaning and purpose |
|---|---|
| 0x0 | temporary address used for maintenance |
| SMAX | current maximum address value for Scheduled access opportunities. SMAX =< UMAX |
| UMAX | current maximum address value for Unscheduled access opportunities SMAX. =< UMAX =< 0x63 |
| 0x64 to 0xFE | reserved |
| 0xFF | broadcast address to all MAC ID nodes on this link |

### 4.3.3 Generic tag address

Generic tag addresses types identify all data transfers for connected mode services. The available capacity in the DLL is 3 octets, see Figure 4.

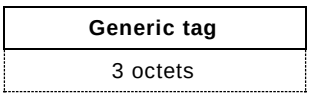



**Figure 4 – Basic structure of a generic tag address**

Generic tags are used to send connected DLSDUs and may be associated with any of the available QoS priorities (Scheduled, High, Low).

NOTE Negotiation and management of generic tags and connected mode services are the responsibility of the DL user.

### 4.3.4 Fixed tag address

Fixed tag addresses types identify DLSAPs within a designated station MAC ID. They are used with all data transfers for connectionless mode services.

NOTE Fixed tags are used to send connectionless DLSDUs and are usually associated with unscheduled QoS priorities (High, Low). Typical uses of fixed tags are for station management and to establish connections.

Each fixed tag address shall consist of two octets as shown in Figure 5.

| Fixed tag service | Destination MAC ID |
|---|---|
| 1 octet | 1 octet |

**Figure 5 – Basic structure of a fixed tag address**

The first octet shall specify the service access point in the destination node as specified in Table 4. The second octet shall be the address of the destination node. The destination address shall be either a MAC ID or 0xFF, which indicates the broadcast address to all MAC IDs on the local link.

**Table 4 – Fixed tag service definitions**

| Fixed tag service | Meaning |
|---|---|
| 0x00 | moderator |
| 0x01 - 0x08 | vendor specific |
| 0x09 | ping request |
| 0x0A – 0x14 | vendor specific |
| 0x15 | tMinus |
| 0x16 - 0x28 | vendor specific |
| 0x29 | ping reply |
| 0x2A - 0x3F | vendor specific |
| 0x40 - 0x6F | reserved |
| 0x70 – 0x7F | vendor-specific |
| 0x80 | I'm alive |
| 0x81 | link parameters |
| 0x82 | reserved |
| 0x83 | UCMM |
| 0x84 | TUI |
| 0x85 | IP (reserved for Internet Protocol) |
| 0x86 | WAMI |
| 0x87 | reserved |
| 0x88 | Keeper UCMM |
| 0x89 | Ethernet (used for address resolution protocol) |
| 0x8A - 0x8B | reserved |
| 0x8C | Time distribution |
| 0x8D – 0x8F | reserved for time distribution |
| 0x90 | debug |
| 0x91 - 0xCF | reserved |
| 0xD0 - 0xEF | group addresses |
| 0xF0 - 0xFF | vendor specific |

Fixed tags in the range 0xD0 to 0xEF shall be reserved to permit group screening of connection identifiers.

## 4.4 Services assumed from the PhL

### 4.4.1 General requirements

Subclauses 4.4.2 to 4.4.3 include requirements for interface to a Type 2 PhL that are not required for other Types.

– Transmit Machine.
– Receive Machine.
– Serializer.
– Deserializer.

This is necessary because the Ph-Interface Data Units (PhIDUs) that Type 2 passes across the DLL–PhL interface are individual serial data and non-data symbols, designated on the DLE side as M_Symbols (see Table 5), whereas PhIDUs for other Types are not conceptualized in this way. As a consequence, a Type 2 Data Link Layer requires a Type 2 PhL.

### 4.4.2 Data encoding rules

The M_Symbols transferred at the DLL to PhL interface are of four types designated 0, 1, +, -. They will be encoded inside the PhL into appropriate Ph_Symbols as shown in Table 5. The M_0 and M_1 will be encoded into Ph_Symbols that represent Manchester biphase L data encoding rules. The M_ND symbols are non-data symbols to allow for the creation of unique data patterns used for start and end delimiters. The example of signal voltage waveform as it might be applied to an electrically-conductive medium is shown in an idealized form in Figure 6 to provide an example of the data encoding rules shown in Table 5.

**Table 5 – Data encoding rules**

| Data bits (common name) | M_Symbol representation | Ph_Symbol encoding | Manchester encoded |
|---|---|---|---|
| **data “zero”** | M_0 or {0} | {L,H} | 0 |
| **data “one”** | M_1 or {1} | {H,L} | 1 |
| **“non_data+”** | M_ND+ or {+} | {H,H} | no data |
| **“non_data–”** | M_ND– or {–} | {L,L} | no data |

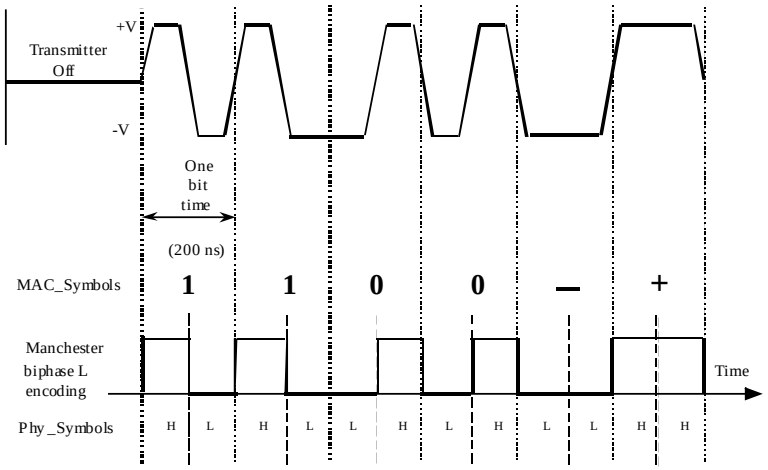



**Figure 6 – M_symbols and Manchester encoding at 5 MHz**

### 4.4.3 DLL to PhL interface

#### 4.4.3.1 General

The DLL to PhL interface need not be exposed in the implementation of any PhL variant. This interface may be internal to the node. If conformance to the DLL to PhL interface is claimed, it shall conform to the requirements of 4.4.3.

#### 4.4.3.2 Ph_lock_indication

ph_lock_indication shall provide an indication of either data lock or Ph_Symbol synchronization by the PhL. Valid states for ph_lock_indication shall be true and false. ph_lock_indication shall be true whenever valid Ph_Symbols are present at the PhL interface and the PhL to DLL interface timing of M_Symbols conforms to the PhL requirements. It shall be false between frames (when no Ph_Symbols are present on the medium) or whenever data synchronization is lost or the timing fails to conform to the PhL requirements. ph_lock_indication shall be true prior to the beginning of the start delimiter.

#### 4.4.3.3 Ph_frame_indication

ph_frame_indication shall provide an indication of a valid data frame from the PhL. Valid states for ph_frame_indication shall be true and false. ph_frame_indication shall be true upon ph_lock_indication = true and reception of the first valid start delimiter. ph_frame_indication shall be false at reception of next M_ND symbol (following the start delimiter) or ph_lock_indication = false.

NOTE This signal provides octet synchronization to the DLL.

#### 4.4.3.4 Ph_carrier_indication

ph_carrier_indication shall represent the presence of a signal carrier on the medium. The ph_carrier_indication shall be true if internal PhL identification of signal carrier has been true during any of the last 4 M_symbol times and it shall be false otherwise.

#### 4.4.3.5 Ph_data_indication

ph_data_indication shall represent the M_Symbols that are decoded from the PhL internal representations such as those shown in Table 5. Valid M_symbols shall be M_0 {0}, M_1 {1}, M_ND+ {+}, and M_ND– {-} as shown in Table 6.

**Table 6 – M Data symbols**

| Data bits (common name) | M_Symbol representation |
|---|---|
| **data "zero"** | M_0 or {0} |
| **data "one"** | M_1 or {1} |
| **"non_data+"** | M_ND+ or {+} |
| **"non_data–"** | M_ND– or {–} |

The ph_data_indication shall represent the M_Symbols as decoded within the PhL whenever ph_lock_indication is true.

#### 4.4.3.6 Ph_status_indication

ph_status_indication shall represent the delimiter status of the frame which was received from the PhL as shown in Table 7. Valid symbols shall be Normal, Abort, and Invalid. ph_status_indication shall indicate Normal after reception of a frame (ph_frame_indication = true) composed of a start delimiter, valid data (no M_ND symbols) and an end delimiter. ph_status_indication shall indicate Abort after reception of a frame (ph_frame_indication = true) composed of a start delimiter, valid data, and a second start delimiter. ph_status_indication shall indicate Invalid after reception of a frame (ph_frame_indication = true) composed of a start delimiter and the detection of any M_ND symbol which was not part of a start or end delimiter.

**Table 7 – Truth table for ph_status_indication**

| Ph_status_indication | Ph_frame_indication | Start delimiters in a single frame | End delimiter detection | Any non-delimiter Manchester violations |
|---|---|---|---|---|
| **Normal** | true | 1 | true | false |
| **Abort** | true | 2 | don't care | false |
| **Invalid** | true | 1 | don't care | true |

#### 4.4.3.7 Ph_data_request

ph_data_request shall present the M_Symbols to be transmitted. Valid symbols shall be M_0, M_1, M_ND+ or M_ND– as shown in Table 6. ph_data_request shall indicate M_0 when no data is to be transmitted (and ph_frame_request = false).

#### 4.4.3.8 Ph_frame_request

ph_frame_request shall be true when ph_data_request presents M_Symbols to be encoded into the appropriate Ph_Symbols by the PhL, and shall be false when no valid M_Symbols are to be transferred to the PhL.

### 4.5 Functional classes

Implementations of this protocol fall into two broad classes:

– connection originators (sometimes called Scanners);

– connection responders or connection targets (sometimes called Adaptors).

Devices with connection origination capability will include functions such as Scheduling object, support for external Configuration Tools and usually a Keeper object.

Connection responders are typically simpler devices which do not need the full set of capabilities.

This specification does not mandate particular groupings of functions for classes of implementation devices.

NOTE Grouping of services and functions into implementation classes can be done by separate specifications for profiles.

# 5 General structure and encoding of PhIDUs and DLPDUs and related elements of procedure

## 5.1 Overview

The DLL and its procedures are those necessary to provide the services offered to DLS user by using the services available from the PhL. The main services of this protocol are

a) provision of timely access opportunities to support scheduled data-transfer DL-services on one link and on an extended multi-link network;

b) the packing of multiple DLS-user transfers into a single PhL transfer unit (DLPDU) to efficiently use each transfer opportunity;

c) the efficient distribution of DLS-user data from a publishing DLS-user to a set of subscribing DLS-users;

d) the provision of a synchronized sense of internal time among the DLEs and available to the DLS-users of the extended link;

e) the standardized availability of local DL-address, buffer and queue management capabilities.

## 5.2 Media access procedure

The primary task of the Data Link Layer is to determine, in co-operation with other Data Link Layers on the same link, the granting of permission to transmit on the medium. At its upper interface, the DLL provides services to receive and deliver service data units for the DLS user and Station Management.

The DLL protocol is based on a fixed, repetitive time cycle called a network update time (NUT). The NUT is maintained in close synchronism among all nodes on the link. A node is not permitted access to transmit on the medium if its NUT does not agree with the NUT currently being used on a link wide basis. Different links may have different NUT duration.

Each node contains its own timer synchronized to the NUT for the local link. Media access is determined by local sub-division of the NUT into access slots. Access to the medium is in sequential order based on the MAC ID of the node. Specific behaviors have been incorporated into the access protocol allowing a node which temporarily assumes a MAC ID of zero to perform link maintenance. The MAC ID numbers of all nodes on a link are unique. Any DLL detecting the presence of a duplicate MAC ID shall immediately stop transmitting.

An implicit token passing mechanism is used to grant access to the medium. Each node monitors the source MAC ID of each DLPDU received. At the end of a DLPDU, each DLL sets an “implicit token register” to the received source MAC ID + 1. If the implicit token register is equal to the local MAC ID, then the node shall transmit one DLPDU containing one or more Lpackets with data or a null DLPDU. In all other cases, the node watches for either a new DLPDU from the node identified by the “implicit token register” or a time-out value if the identified node fails to transmit. In each case, the “implicit token” is automatically advanced to the next MAC ID. All nodes have the same value in their “implicit token register” preventing collisions on the medium.

The time-out period (called the “slot time”) is based on the amount of time required for

– the current node to hear the end of the transmission from the previous node;

– the current node to begin transmitting;

– the next node to hear the beginning of the transmission from the current node.

The slot time is adjusted to compensate for the total length of the medium since the propagation delay of the medium effects the first and last item on the previous list.

NOTE The calculation of slot time is specified in 8.2.

Each NUT is divided into three major parts: scheduled, unscheduled, and guardband as shown in Figure 7. This sequence is repeated in every NUT. The implicit token passing mechanism is used to grant access to the medium during both the unscheduled and scheduled intervals.

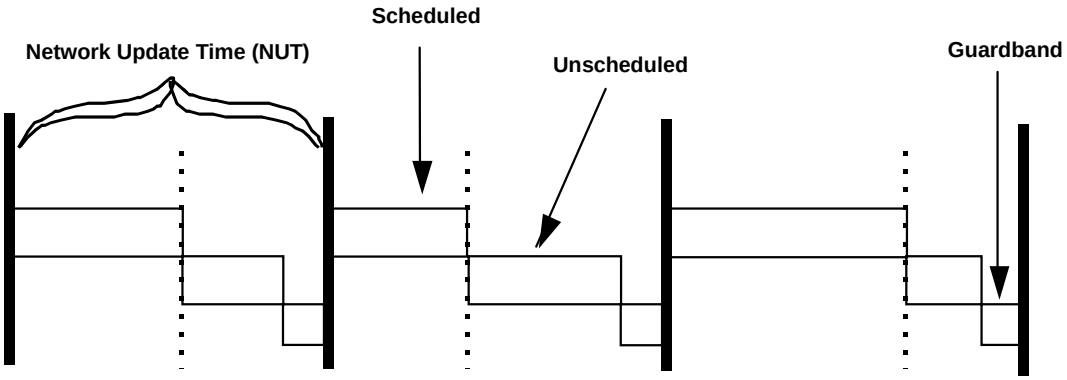



**Figure 7 – NUT structure**

During the scheduled part of the NUT, each node, starting with node 0 and ending with node SMAX, gets a chance to transmit time-critical (scheduled) data. SMAX is the MAC ID of the highest numbered node that has access to the medium during the scheduled part of the NUT. Every node between 0 and SMAX has only one opportunity to send one DLPDU of scheduled data in each NUT. The opportunity to access the medium during the scheduled time is the same for each node in every NUT. This allows data that is transmitted during the scheduled portion of the NUT to be sent in a predictable and deterministic manner. Figure 8 shows how the permission to transmit is granted during the scheduled time. The DLS-user regulates the amount of data that each node may transmit during this scheduled token pass.

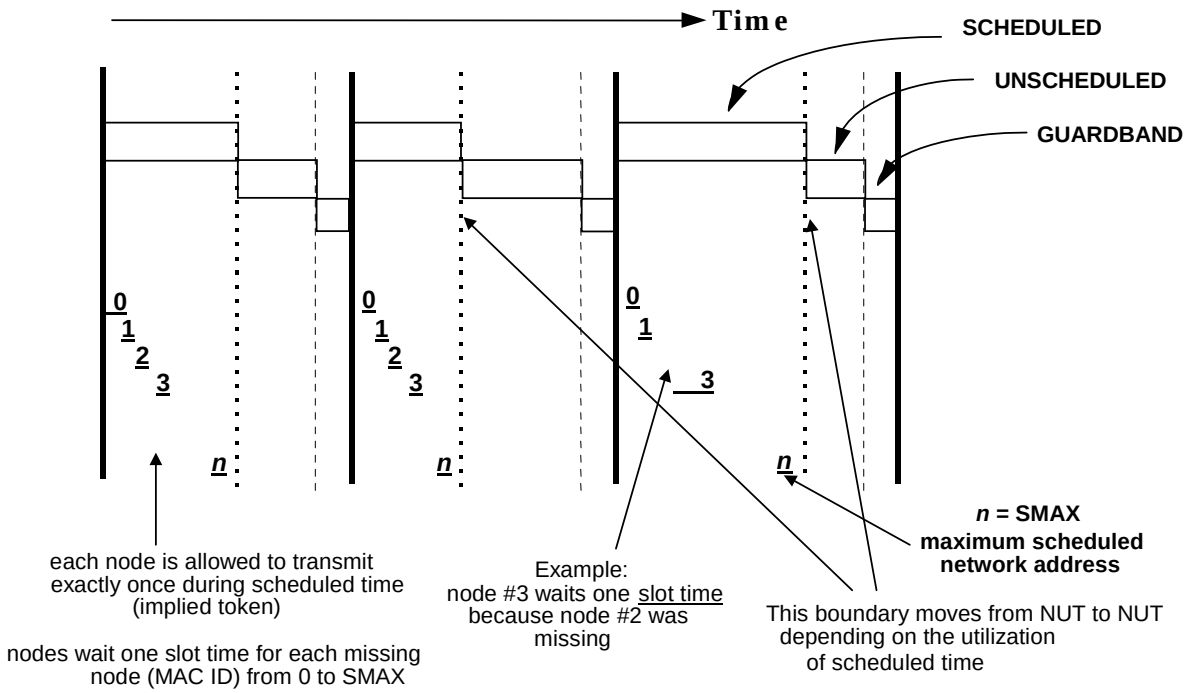



**Figure 8 – Media access during scheduled time**

During the unscheduled part of the NUT, each node from 0 to UMAX shares the opportunity to transmit one DLPDU of non-time critical data in a round robin fashion, until the allocated NUT duration is exhausted. UMAX is the MAC ID of the highest numbered node that has access to the medium during the unscheduled part of the NUT. The round robin method of access opportunity enables every node between 0 and UMAX to have zero, one or many opportunities to send unscheduled data depending on how much of the NUT remains after the completion of the scheduled time. Variations in scheduled traffic means the opportunity to access the medium during the unscheduled time may be different for each node in every NUT. Figure 9 shows how the permission to transmit is granted during the unscheduled time. The MAC ID of the node that goes first in the unscheduled part of the NUT is incremented by 1 for each NUT. The unscheduled token begins at the MAC ID specified in the unscheduled start register (USR) of the previous moderator DLPDU. The USR increments by one modulo (UMAX+1) each NUT. If the USR reaches UMAX before the guardband, it returns to zero and the token pass continues.

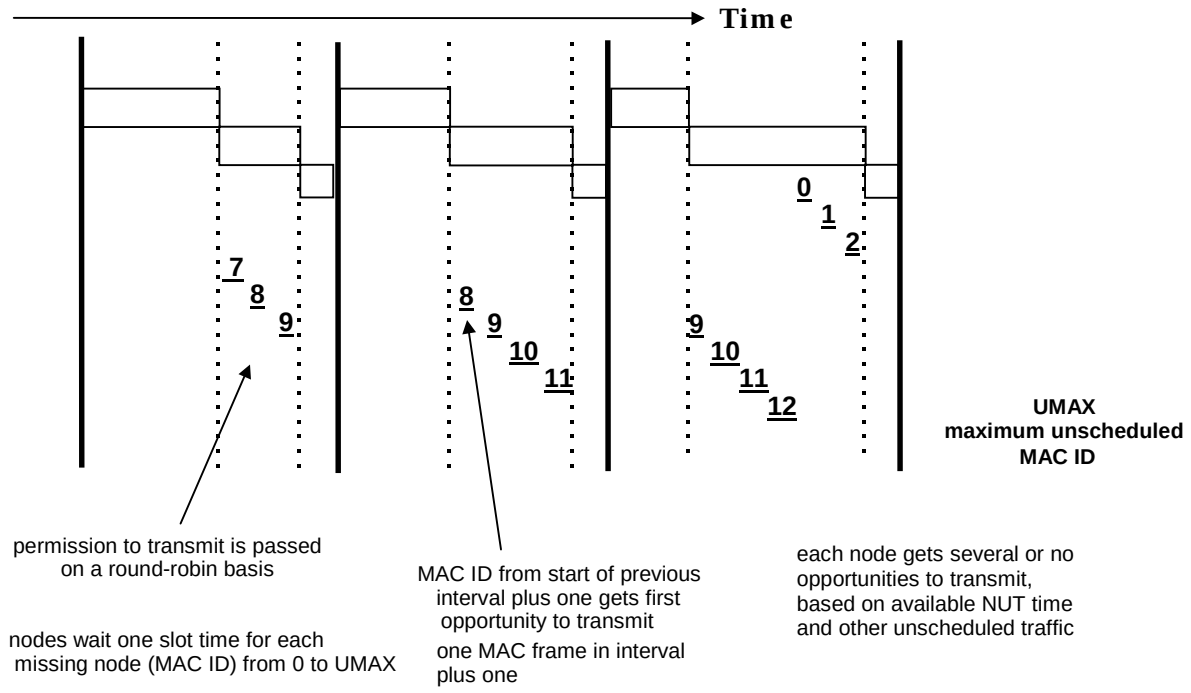



**Figure 9 – Media access during unscheduled time**

When the guardband is reached, all nodes stop transmitting. A node is not allowed to start a transmission unless it can be completed before the beginning of the guardband. During the guardband, the node with the lowest MAC ID (called the “moderator”) transmits a maintenance message (called the “moderator DLPDU”) that accomplishes two things:

– it keeps the NUT timers of all nodes synchronized;

– it publishes critical link parameters enabling all members of the local DLL group to share a common version of important DLL values such as NUT, slot time, SMAX, UMAX, etc.

The moderator transmits the moderator DLPDU, which re-synchronizes all nodes and restarts the NUT. Following the receipt of a valid moderator DLPDU, each node compares its internal values with those transmitted in the moderator DLPDU. A node using link parameters that disagree with the moderator disables itself. If the moderator DLPDU is not heard for 2 consecutive NUTs, the node with the lowest MAC ID assumes the moderator role, and begins transmitting the moderator DLPDU in the guardband of the third NUT. A moderator node that notices another node on-line and transmitting with a MAC ID lower than its own immediately cancels its moderator role.

Typical situations that may cause disruption of the DLL access protocol include

– induced noise on the link;

– poor cable or termination quality;

– physically connecting two links together while the network is operating.

One common consequence of such disruption is that nodes may be caused to disagree as to which node should be transmitting; this is called a network “non-concurrence”. Another potential problem occurs when the nodes do not agree to the same link configuration parameters. A node that disagrees with the link parameters as transmitted by the moderator is called a “rogue” and immediately stops transmitting. The DLL access protocol is designed to recover a rogue node and bring it back on-line.

## 5.3 DLPDU structure and encoding

### 5.3.1 General

DLPDUs are the DLPDUs passed to the Ph layer for sending on the Ph medium. The DLPDUs are formed by the transmit logical link control (TxLLC) protocol in two stages:

– DLSDUs are formed into Link-units (Lpackets);

– Multiple Lpackets are packed into one DLPDU subject to constraints set by the QoS required, the type of access opportunity available and limits on DLPDU overall length.

### 5.3.2 DLPDU components

The general DLPDU structure is shown in Figure 10 and managed by the DLL support units TxFramer, RxFramer, Octet Serializer and Octet Deserializer as shown in Figure 2.

The components of the DLPDU shall be transmitted in the following order: preamble, start delimiter, source MAC ID, zero or more Lpackets, FCS and end delimiter.

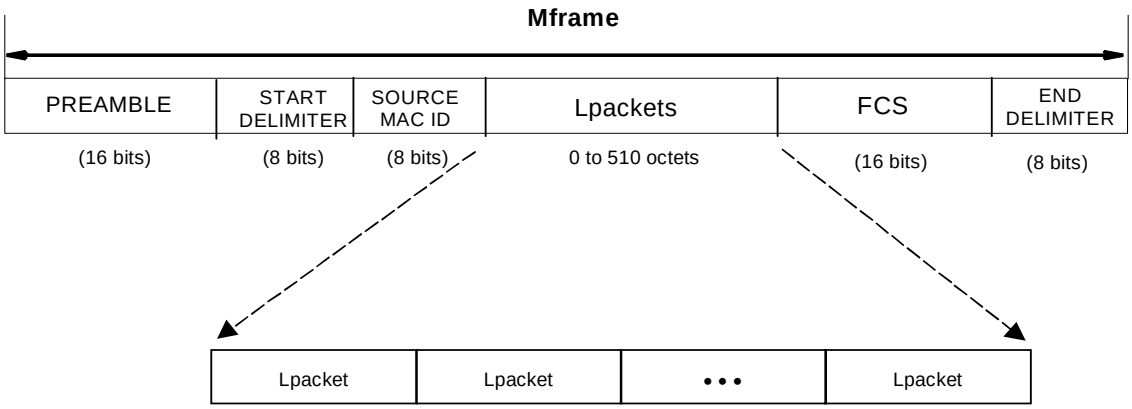



**Figure 10 – DLPDU format**

### 5.3.3 Preamble

The preamble shall be composed of 16 consecutive {1} M_Symbols.

### 5.3.4 Start and end delimiters

The start delimiter shall consist of these M_Symbols, {+, 0, –, +, –, 1, 0, 1}, transmitted from left to right.

The end delimiter shall consist of these M_Symbols, {1, 0, 0, 1, +, –, +, –}, transmitted from left to right.

The use of non-data M_Symbols shall be reserved for the start and end delimiters.

### 5.3.5 DLPDU octets and ordering

The source MAC ID, Lpackets, and FCS shall be composed of octets. An octet shall be composed of exactly eight M_Symbols each of which shall be either {0} or {1}. An octet shall be transferred to the PhL low bit first. Multiple octet fields shall be transmitted low order octet first (little endian format).

### 5.3.6 Source MAC ID

The source MAC ID address shall be a single octet as specified in 4.3.2. A node may also temporarily assume a MAC ID of zero to perform link maintenance.

### 5.3.7 Lpackets field

Zero or more Lpackets shall be concatenated into a single DLPDU subject to the constraint that the maximum number of octets which compose all Lpackets in a DLPDU shall be 510.

An additional constraint is that DLPDUs sent during a scheduled access opportunity shall only contain Lpackets with a QoS parameter of scheduled.

### 5.3.8 Frame check sequence (FCS)

#### 5.3.8.1 Frame check sequence requirements

The FCS shall be a modified CCITT checksum using the 16 bit polynomial: $G(X) = X^{16} + X^{12} + X^5 + 1$. A two octet FCS shall be included in each DLPDU.

The formal concepts for its generation and checking on reception are described in 5.3.8.2. The generation and checking of the FCS are further specified in 9.7 and 9.8.

#### 5.3.8.2 Frame check sequence formal concepts

##### 5.3.8.2.1 Frame check sequence field

Within 5.3.8.2, any reference to bit *K* of an octet is a reference to the bit whose weight in a one-octet unsigned integer is $2^K$.

NOTE This is sometimes referred to as “little endian” bit numbering.

For this standard, as in other international standards (for example, ISO/IEC 3309, ISO/IEC 8802 and ISO/IEC 9314-2), DLPDU-level error detection is provided by calculating and appending a multi-bit frame check sequence (FCS) to the other DLPDU fields during transmission to form a "systematic code word"[3] of length ***n*** consisting of ***k*** DLPDU message bits followed by ***n - k*** (equal to 16) redundant bits, and by calculating during reception that the message and concatenated FCS form a legal (*n*,*k*) code word. The mechanism for this checking is as follows, where the generic form of the generator polynomial for this FCS construction is specified in equation (4) and the polynomial for the receiver’s expected residue is specified in equation (9). The specific polynomials for this standard are specified in Table 8.

**Table 8 – FCS length, polynomials and constants**

| Item | Value | |
|---|---|---|
| *n-k* | 16 | |
| G(X) | $X^{16} + X^{12} + X^5 + 1$ | (Notes 1, 2) |
| R(X) | $X^{12} + X^{11} + X^{10} + X^8 + X^3 + X^2 + X + 1$ | (Notes 2, 3) |

NOTE 1 Code words D(X) constructed from this G(X) polynomial have Hamming distance 4 for lengths ≤ 4 095 octets, and all errors of odd weight are detected.

NOTE 2 These are the same polynomials and method as specified in ISO/IEC 3309 (HDLC).

NOTE 3 The remainder R(x) should be 0001 1101 0000 1111 ($X^{15}$ to $X^0$, respectively) in the absence of errors.

[3] W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes* (2nd edition), MIT Press, Cambridge, 1972.

#### 5.3.8.2.2 At the sending DLE

The original message (that is, the DLPDU without an FCS), the FCS, and the composite message code word (the concatenated DLPDU and FCS) shall be regarded as vectors M(X), F(X), and D(X), of dimension *k*, *n* - *k*, and *n*, respectively, in an extension field over GF(2). If the message bits are $m_1$ ... $m_k$ and the FCS bits are $f_{n-k-1}$ ... $f_0$, where

| | |
|---|---|
| $m_1$ ... $m_8$ | form the first octet sent, |
| $m_{8N-7}$ ... $m_{8N}$ | form the *N*th octet sent, |
| $f_7$ ... $f_0$ | form the last octet sent, and |
| $m_1$ | is sent by the first PhL symbol(s) of the message and $f_0$ is sent by the last PhL symbol(s) of the message (not counting PhL framing information), |

NOTE 1 This “as transmitted” ordering is critical to the error detection properties of the FCS.

then the message vector M(X) shall be regarded to be

$$M(X) = m_1X^{k-1} + m_2X^{k-2} + \ldots + m_{k-1}X^1 + m_k \quad (1)$$

and the FCS vector F(X) shall be regarded to be

$$F(X) = f_{n-k-1}X^{n-k-1} + \ldots + f_0 = f_{15}X^{15} + \ldots + f_0 \quad (2)$$

The composite vector D(X), for the complete DLPDU, shall be constructed as the concatenation of the message and FCS vectors

$$D(X) = M(X)\,X^{n-k} + F(X) = m_1X^{n-1} + m_2X^{n-2} + \ldots + m_kX^{n-k} + f_{n-k-1}X^{n-k-1} + \ldots + f_0 = m_1X^{n-1} + m_2X^{n-2} + \ldots + m_kX^{16} + f_{15}X^{15} + \ldots + f_0 \quad \text{(for the case of k = 15)} \quad (3)$$

The DLPDU presented to the PhL shall consist of an octet sequence in the specified order.

The redundant check bits $f_{n-k-1}$ ... $f_0$ of the FCS shall be the coefficients of the remainder F(X), after division by G(X), of $L(X)\,(X^k + 1) + M(X)\,X^{n-k}$

where G(X) is the degree ***n-k*** generator polynomial for the code words

$$G(X) = X^{n-k} + g_{n-k-1}X^{n-k-1} + \ldots + 1 \quad (4)$$

and L(X) is the maximal weight (all ones) polynomial of degree ***n-k-1***

$$L(X) = (X^{n-k} + 1) / (X + 1) = X^{n-k-1} + X^{n-k-2} + \ldots + X + 1 = X^{15} + X^{14} + X^{13} + X^{12} + \ldots + X^2 + X + 1 \quad \text{(for the case of k = 15)} \quad (5)$$

That is,

$$F(X) = L(X)\,(X^k + 1) + M(X)\,X^{n-k} \ \text{(modulo } G(X)) \quad (6)$$

NOTE 2 The L(X) terms are included in the computation to detect initial or terminal message truncation or extension by adding a length-dependent factor to the FCS.

NOTE 3 As a typical implementation when n-k = 16, the initial remainder of the division is preset to all ones. The transmitted message bit stream is multiplied by $X^{n-k}$ and divided (modulo 2) by the generator polynomial G(X), specified in equation (7). The ones complement of the resulting remainder is transmitted as the (n-k)-bit FCS, with the coefficient of $X^{n-k-1}$ transmitted first.

The equation for D(X) does not apply to the Type 8 protocol because that protocol sends the message and check bits in separate DLPDUs. The equation for F(X) as just given does not apply to the Type 8 protocol because that protocol does not complement the check bits before transmission. The equation corresponding to F(X) for the Type 8 protocol is

$$F_{Type\ 8}(X) = L(X)\ X^{k} + M(X)\ X^{n-k} \ (\text{modulo}\ G(X)) \quad (7)$$

NOTE 4 As a typical implementation for the Type 8 protocol, the initial remainder of the division is preset to all ones. The transmitted message bit stream is multiplied by $X^{n-k}$ and divided (modulo 2) by the generator polynomial G(X), specified in equation (7). The resulting remainder without ones complementation is transmitted as the (n-k)-bit FCS, with the coefficient of $X^0$ transmitted first.

#### 5.3.8.2.3 At the receiving DLE

The octet sequence indicated by the PhE shall be concatenated into the received DLPDU and FCS, and regarded as a vector V(X) of dimension ***u***

$$V(X) = v_1X^{u-1} + v_2X^{u-2} + \ldots + v_{u-1}X + v_u \quad (8)$$

NOTE 1 Because of errors **u** can be different than **n**, the dimension of the transmitted code vector.

A remainder R(X) shall be computed for V(X), the received DLPDU and FCS, by a method similar to that used by the sending DLE (see 5.3.8.2.2) in computing F(X)

$$R(X) = L(X)\ X^{u} + V(X)\ X^{n-k} \ (\text{modulo}\ G(X)) \quad (9)$$
$$= r_{n-k-1}X^{n-k-1} + \ldots + r_0$$

Define E(X) to be the error code vector of the additive (modulo-2) differences between the transmitted code vector D(X) and the received vector V(X) resulting from errors encountered (in the PhS provider and in bridges) between sending and receiving DLEs.

$$E(X) = D(X) + V(X) \quad (10)$$

If no error has occurred, so that E(X) = 0, then R(X) will equal a non-zero constant remainder polynomial

$$R_{ok}(X) = L(X)\ X^{n-k} \ (\text{modulo}\ G(X)) \quad (11)$$

whose value is independent of D(X). Unfortunately R(X) will also equal $R_{ok}(X)$ in those cases where E(X) is an exact non-zero multiple of G(X), in which case there are “undetectable” errors. In all other cases, R(X) will not equal $R_{ok}(X)$; such DLPDUs are erroneous and shall be discarded without further analysis.

NOTE 2 As a typical implementation, the initial remainder of the division is preset to all ones. The received bit stream is multiplied by $X^{n-k}$ and divided (modulo 2) by the generator polynomial G(X), specified in equation (8).

### 5.3.9 Null DLPDU

A DLPDU with an empty Lpacket field, called a null DLPDU, shall be sent by a node to indicate that it has no data to send at the QoS allowed by the access opportunity.

### 5.3.10 Abort DLPDU

To abort a partially transmitted DLPDU, the Data Link Layer shall transmit the following sequence:

– one 0xFF octet;

– start delimiter;

– end delimiter.

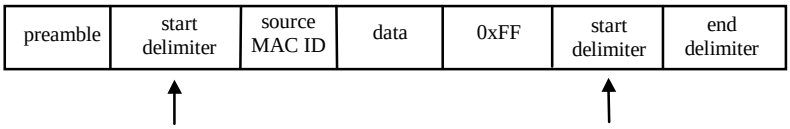



**Figure 11 – Aborting a DLPDU during transmission**

NOTE 1 Some implementations of this specification can build the DLPDU as it is being transmitted. If a higher layer has provided only a partial DLPDU to the Data Link Layer, and the DLL has begun transmission of the DLPDU assuming that the rest will be provided shortly, a transmit underflow occurs if the rest of the DLPDU is not provided to the DLL soon enough to complete the transmission.

NOTE 2 Although the DLL assembles the entire DLPDU including delimiters, the PhL performs start and end delimiter detection. The PhL identifies a transmit abort as a DLPDU which includes two start delimiters which are not separated by an end delimiter, as shown in Figure 11.

NOTE 3 The 0xFF octet is inserted before the start delimiter / end delimiter pair to avoid violating PhL constraints governing consecutive physical symbol relationships on the media.

## 5.4 Lpacket components

### 5.4.1 General Lpacket structure

The Lpacket (or link unit) is the PDU which is used to carry DLSDUs between peer DLS users. The components of the Lpacket shall be transmitted in the following order: size, control, tag, and link data as shown in Figure 12.

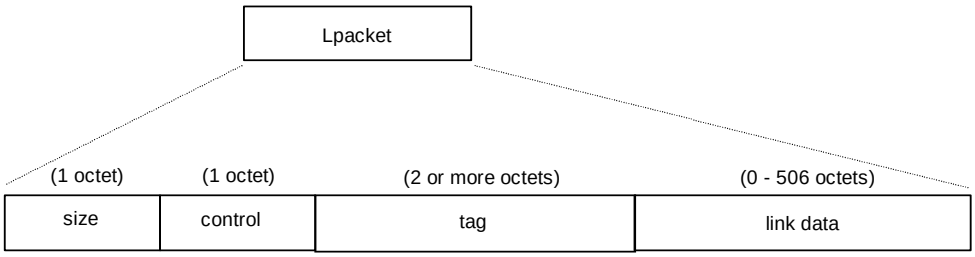



**Figure 12 – Lpacket format**

### 5.4.2 Size

The size field shall specify the number of octet pairs (from 3 to 255) contained in an individual Lpacket. The number of octet pairs shall include the size, control, tag, and link data fields counted in accordance with the following rules:

– combined, the size and control shall count as a single octet pair;

– the number of octet pairs from the tag and link data fields shall be counted separately and odd octet counts shall be rounded up;

– although the counting of octet pairs is done independently, the format of an Lpacket as placed into a DLPDU shall not include the extra octets caused by the round up.

NOTE For example, an Lpacket with a 3 octet tag (2 octet pairs) and a 9 octet link data (5 octet pairs) has a size field of 1+2+5 = 8 octet pairs; however the Lpacket occupies 2+3+9 = 14 octets in the DLPDU.

### 5.4.3 Control

#### 5.4.3.1 Bit 0 and Bit 4 (type of Lpacket)

The bits of the control field shall be numbered 0 to 7, where bit 0 shall be the least significant bit of the control field. Bit 0 shall indicate the type of Lpacket.

– When set (bit 0 = 1), the Lpacket shall be a fixed tag Lpacket (see 4.3.4).
– When clear (bit 0 = 0), the Lpacket shall be a generic tag Lpacket (see 4.3.3).

Bit 4 of the control field shall be the inverse of bit 0. If bit 0 is clear, then bit 4 shall be set. If bit 0 is set, then bit 4 shall be clear.

#### 5.4.3.2 Bit 1 (odd tag size)

Bit 1 of the control field shall indicate whether the tag field contains an even or odd number of octets. When clear (bit 1 = 0), this bit shall indicate that the tag contains an even number of octets. When set (bit 1 = 1), it shall indicate that the tag contains an odd number of octets.

#### 5.4.3.3 Bit 2 (odd link data size)

Bit 2 of the control field shall indicate whether the link data contains an even or odd number of octets. When clear (bit 2 = 0), this bit shall indicate that the link data contains an even number of octets. When set (bit 2 = 1), it shall indicate that the link data contains an odd number of octets.

#### 5.4.3.4 Bits 3, 5, 6 and 7 (reserved bits)

Bits 3, 5, 6 and 7 of the control field are reserved and shall be zero.

### 5.4.4 Generic tag Lpackets

Generic link-units are formed from the DLS-generic-tag and DLS-user-data provided by the DLS user with the addition of size and control field information for generic tags as shown in Figure 13. The 3 octet generic tag address (see 4.3.3) shall identify the connected mode information carried in the link data field.

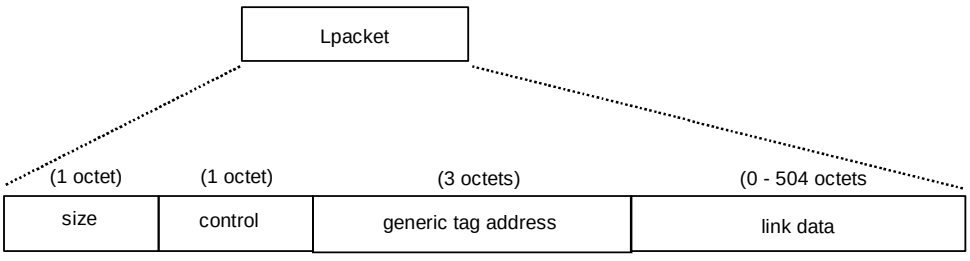



**Figure 13 – Generic tag Lpacket format**

The minimum length generic tag Lpacket shall be 5 octets (minimum link data size of 0 octets). The maximum size of link data shall be 504 octets.

NOTE Generic Lpackets are usually connection mode transfers with the QoS value of scheduled or low.

### 5.4.5 Fixed tag Lpackets

Fixed tag link-units are formed from the DLS-fixed-tag and DLS-user-data provided by the DLS user with the addition of size and control field information for fixed tags as shown in Figure 14. The 2 octet fixed tag address (see 4.3.4) shall identify the service access point in the destination node and the address of the destination node.

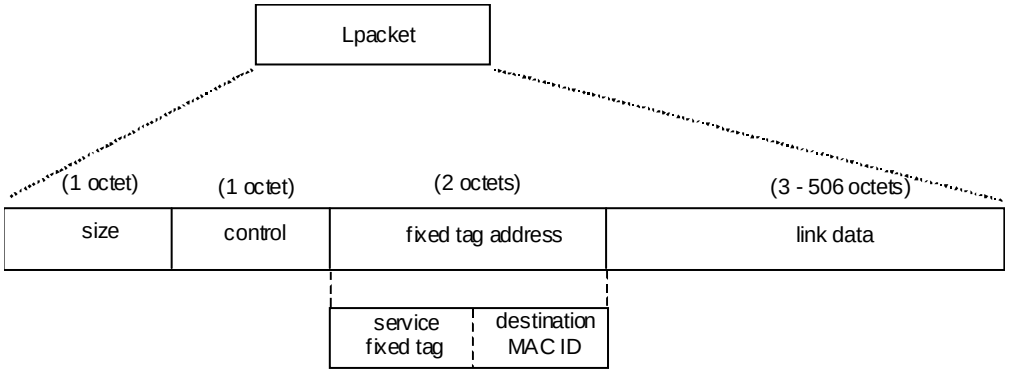



**Figure 14 – Fixed tag Lpacket format**

The minimum length fixed tag Lpacket shall be 7 octets (minimum link data size of 3 octets). The maximum size of link data shall be 506 octets.

NOTE Fixed Lpackets are always connectionless mode transfers. They are not usually sent with the QoS parameter value scheduled.

## 5.5 DLPDU procedures

### 5.5.1 General

The transmission protocol combines one or more Lpackets into single DLPDUs for sending according to the available transmission opportunity. The DLPDU components are passed to the Ph layer by the transmit machine framer (TxM) and Serializer (see 9.3).

A node shall always send one DLPDU at each valid access opportunity for it to transmit, and no DLPDU shall include Lpackets with a lesser QoS parameter than the QoS value applicable to the access opportunity.

If a node has no Lpackets available at or better than the QoS of the access opportunity, then a Null DLPDU shall be sent.

If for internal reasons a node begins a DLPDU and is unable to complete it, the abort sequence shall be sent, transmission shall stop for that access opportunity and the local DLS user shall be notified.

NOTE Null DLPDUs and abort DLPDUs provide an efficiency improvement as they notify other nodes that use of the transmission slot has ended and enable the next node to begin transmission without waiting for slot time out.

Each Lpacket in the queue awaiting transfer has an associated local identifier. At completion of the transfer to the PhL, or an aborted transfer, the local identifier is returned to the local DLS user with a status confirmation of OK or TxABORT.

The DLS user has additional local services to identify unsent Lpackets and request they are flushed from the queue, this action results in the confirmation status of FLUSHED being returned with the local identifier for Lpackets that have been flushed.

### 5.5.2 Sending scheduled DLPDUs

Scheduled DLPDUs are normally used for transfer of connected mode operation using Generic Lpackets. They are formed by concatenating multiple Lpackets (subject to a limit of 510 octets), prepending Ph-control information and source station MAC ID, and appending a defined frame check sequence and further Ph-control information.

Lpackets are packed into a Scheduled DLPDU in FIFO order as received from the DLS user, until the queue is empty of packets with Scheduled QoS or the next such Lpacket size exceeds the remaining space.

### 5.5.3 Sending unscheduled DLPDUs

Unscheduled DLPDUs may contain both Generic Lpackets and Fixed Lpackets. They are formed by concatenating multiple Lpackets (subject to a limit of 510 octets), prepending Ph-control information and source station MAC ID, and appending a defined frame check sequence and further Ph-control information.

Lpackets are packed into an unscheduled DLPDU in QoS priority order (Scheduled, High and Low priorities respectively). Within each QoS, the FIFO order of queuing by the DLS user is followed, until the queue is empty or the next such Lpacket size exceeds the remaining space.

### 5.5.4 Receiving DLPDUs

#### 5.5.4.1 Unpacking

Received DLPDUs are assembled from the incoming Ph stream of symbols and unpacked by the Octet Deserializer, Receive Machine Framer (RxM) and Receive Logical Link Control (RxLLC) functions in two stages:

– a frame check sequence is computed over the stream of received octets, (excluding Ph delimiters) and checked for the proper residual value, if this is correct, then

– each Lpacket contained in the DLPDU is examined to extract its tag. Each extracted tag is then compared with the contents of a local tag filter to determine if the associated Lpacket is of interest to the receiving station. Lpackets whose tags match the tag filter are then processed further and passed to the DL user, DL management or used inside the DLL. Indications passed to the DL user or DL management include Lpacket size, Lpacket tag and Lpacket data. In addition for fixed tag Lpackets, the source ID is also indicated.

#### 5.5.4.2 Tag filtering

Each local DLS provider maintains a list of tags used to identify Lpackets of interest to its DL user or its internal DL management functions. The DL user may access and change this list to specify

– generic tags of interest

– fixed tags of interest

NOTE A generic tag specifies the Lpackets for a single connection-mode object. A fixed tag specifies all connectionless-mode messages in a particular class, for example Time distribution.

## 5.6 Summary of DLL support services and objects

The operation and coordination of participating DLL providers is governed by variables and attributes held in DL management objects or provided by the DL user services listed in Table 9. Access to these items is via defined fixed tag addresses, DL user services or unspecified local services. Details are given in the listed clauses.

**Table 9 – DLL support services and objects**

| DLL item | Description | Location |
|---|---|---|
| **DLS user services** | | see IEC 61158-3-2 |
| connected mode transfer service | a local service for requesting, indicating and confirming generic tag (connected mode) data transfer by Lpackets | see 6.2.2 |
| connectionless mode transfer service | a local service for requesting, indicating and confirming generic tag (connected mode) data transfer by Lpackets | see 6.2.2 |
| queue maintenance service | a local service for flushing unsent Lpacket messages from the internal DL service queue | see 6.2.3 |
| tag filter service | a local service for the DLS user to inform its local DL provider which Lpacket tags and tag types are to be accepted and processed by the DLL or passed to the DL user | see 6.2.4 |
| link synchronization service | a local service for informing the local DLS user of the current NUT number to maintain schedule pointers | see 6.2.5 |
| synchronized parameter change service and tMinus-start-countdown | a local service for loading and changing local lists of current and pending DL configuration data and in conjunction with the tMinus-start-countdown service and moderator DLPDU, to manage an accurately timed change over to new DL link configurations | see 6.2.6 |
| event reports service | local services to report various events for use by management | see 6.2.7 |
| bad FCS service | | see 6.2.8 |
| current moderator | | see 6.2.9 |
| power-up and on-line | | see 6.2.10 |
| enable moderator | | see 6.2.11 |
| listen only | | see 6.2.12 |
| Generic tag service | an Lpacket service for connected mode communication | see 6.3 |
| Fixed tag services | a range of Lpacket service addresses for various connectionless mode communication | see 5.4.5 |
| moderator | an Lpacket broadcast in every NUT for managing time precision and timing for change of link parameters | see 6.4 |
| time distribution service | an Lpacket for publishing time values and offsets used in coordinating the same time sense in all stations | see 6.5 |
| UCMM | an Lpacket service to transfer DLSDUs from a DLS user to the Unconnected Message Manager in a peer DLS-user | see 6.6 |
| Keeper UCMM | an Lpacket service to transfer DLSDUs from a DLS user to the Keeper objects in other nodes | see 6.7 |
| TUI | an Lpacket broadcast to all stations assisting their synchronization to the current link configuration | see 6.8 |
| link parameters | an Lpacket service for publishing pending link parameters in advance of a synchronized change of link operating conditions | see 6.9 |
| tMinus | an Lpacket service to start a count down leading to a change of link parameters and return the result. | see 6.9 |
| I'm alive | an Lpacket broadcast by new and reset devices to assist their entry into an operating system | see 6.10 |
| ping request and reply | an Lpacket service soliciting identification responses from all other active devices on the link | see 6.11 |
| WAMI | an Lpacket allowing temporary devices to identify a local device MAC ID when they are plugged into its network access port | see 6.12 |
| debug | an Lpacket service to trace object state transitions | see 6.13 |

| DLL item | Description | Location |
|---|---|---|
| **Objects** | | |
| ControlNet object | holds and distributes station management data classes and instances for Ph and DL layers, also collects diagnostic information for use by other DLS-user entities | see 7.2 |
| Keeper object | accessed via the Keeper UCMM, or general UCMM fixed tag, to provide a range of support services for link operation, schedules and connections<br>holds link data and attributes for all devices on a link<br>keeps copies of all connection originator data for connection originating devices in a network<br>negotiates with other Keeper objects on a link to agree on master Keeper and backup Keeper roles<br>when master Keeper distributes link attributes to all nodes and keeps backup Keepers synchronized<br>regularly issues TUI Lpacket to ensure all stations are synchronized to the current links configuration<br>manages the network resource semaphore enabling orderly modification of link attribute data | see 7.3 |
| Scheduling object | provides services to external link scheduling software in support of read/write for connection and schedule information from involved devices<br>stores the current schedule details and connection password in the Keeper | see 7.4 |
| TCP/IP Interface object | provides the mechanism to configure the TCP/IP interface of a device | see 7.5 |
| Ethernet Link object | maintains link-specific counters and status information for a physical Ethernet ISO/IEC 8802-3 port | see 7.6 |
| DeviceNet object | holds and distributes station management data classes and instances for Ph and DL layers, also collects diagnostic information for use by other DLS-user entities | see 7.7 |
| Connection Configuration object | provides an interface to create, configure and control connections in a device | see 7.8 |

# 6 Specific DLPDU structure, encoding and procedures

## 6.1 Modeling language

### 6.1.1 State machine description

Some of the components of the Data Link Layer are specified using a state machine description language. The action statement is expressed in C++. Other components do not require a state machine description and are specified using only C++.

A state machine is described using the following syntax, where * indicates repetition of one or more of the preceding component, and {} indicates an optional component:

```
state-machine:=
    <header-statements>
    state*
state:=
    “state:” <state-name>
       transition*
transition:=
    “event:” <event-expression>
    {“condition:” <condition-expression>}
    “destination:” <state-name>
    {“action:” <action-statements>}
```

State-names are legal C++ identifiers, event-expressions and condition-expressions are C++ expressions, and header-statements and action-statements are lists of C++ statements.

Header-statements include

– #include and #define;

– variable and interface declarations;

– subroutine definitions.

An event-expression indicates the specified event when the value of the expression is TRUE. An event is required to trigger a transition. A condition-expression qualifies a transition. A qualified transition occurs when its event occurs if the condition is true at the time of the event. In general, event-expressions are FALSE upon entry into a state. There are a few cases where an event-expression may be true upon entry to a state. In these cases, the transition takes place immediately.

Action statements are executed when the transition is triggered.

All the event triggers and conditions of the current state are evaluated continuously and concurrently until a given transition is enabled. All expressions and statements execute instantaneously. The following model illustrates the precedence of the various parts of a transition:

```
if(the event has occurred && the condition is true)
    {
    do the actions;
    go to the destination;
    }
```

Implementations may use other code or syntax, however the implementation performance shall be equivalent to that specified except for internal execution timing and local variables

### 6.1.2 Use of DLL- prefix

The primitives and parameters described in IEC 61158-3-2 use the following prefixes:

DL- for data link

DLS- for DL-link service

DLMS- for DL-management service.

Within this Type 2: Data Link Protocol Specification, the generic prefix DLL- is used as equivalent to the appropriate service description prefix.

### 6.1.3 Data types

The data types used in the data link variables, parameters, state machines and example code are specified in IEC 61158-5-2.

The specification of data types corresponds to the notation of IEC 61131-3. A list of the elementary data types, their corresponding description and some of their valid values is shown in Table 10. Additional detail is specified in IEC 61158-6-2.

**Table 10 – Elementary data types**

| Keyword | Description | Range | |
|---|---|---|---|
| | | Minimum | Maximum |
| **BOOL** | Boolean | | |
| **SINT** | Short Integer | -128 | 127 |
| **INT** | Integer | -32 768 | 32 767 |
| **DINT** | Double Integer | $-2^{31}$ | $2^{31}-1$ |
| **LINT** | Long Integer | $-2^{63}$ | $2^{63}-1$ |
| **USINT** | Unsigned Short Integer | 0 | 255 |
| **UINT** | Unsigned Integer | 0 | 65 535 |
| **UDINT** | Unsigned Double Integer | 0 | $2^{32}-1$ |
| **ULINT** | Unsigned Long Integer | 0 | $2^{64}-1$ |
| **REAL** | Floating point | | |
| **LREAL** | Long float | | |
| **ITIME** | Duration (short) | | |
| **TIME** | Duration | | |
| **FTIME** | Duration (high resolution) | | |
| **LTIME** | Duration (long) | | |
| **DATE** | Date only | | |
| **TIME_OF_DAY or TOD** | Time of day | | |
| **DATE_AND_TIME or DT** | Date and time of day | | |
| **STRING** | character string (1 octet per character) | | |
| **STRING2** | character string (2 octet per character) | | |
| **STRINGN** | character string (N octets per character) | | |
| **SHORT_STRING** | character string (1 octet per character, 1 octet length indicator) | | |
| **SWORD** | bit string - 8 bits | | |
| **WORD** | bit string - 16-bits | | |
| **DWORD** | bit string - 32-bits | | |
| **LWORD** | bit string - 64-bits | | |

## 6.2 DLS user services

### 6.2.1 General

The DLL user services provided to the DLS user have the following interfaces used for the specified purposes. The related service definitions and time sequence diagrams are given in IEC 61158-3-2.

### 6.2.2 Connected mode and connectionless mode transfer service

#### 6.2.2.1 Service encoding

The request and confirmation services used to queue the sending of Lpackets by the DLL shall be of the following forms.

a) Connection mode transfers use Generic tag request and confirmation.

```
DLL_xmit_generic_request(                    DLL_xmit_generic_confirm(
      IDENTIFIER   id,                             IDENTIFIER   id,
      USINT     Lpacket[],                         TXSTATUS  status );
      UINT      Lpacket_size,
      PRIORITY  priority,
      USINT     gentag[3] );
```

b) Connectionless mode transfers use Fixed tag request and confirmation.

```
DLL_xmit_fixed_request(                      DLL_xmit_fixed_confirm(
      IDENTIFIER   id,                             IDENTIFIER   id,
      USINT     Lpacket[],                         TXSTATUS  status );
      UINT      Lpacket_size,
      PRIORITY  priority,
      USINT     service,
      USINT     destID );
```

c) Connection mode transfers use Generic tag indication.

```
DLL_recv_generic_indication(
      USINT     Lpacket[]
      UINT      Lpacket_size,
      USINT     gentag[3] );
```

d) Connectionless mode transfers use Fixed tag indication.

```
DLL_recv_fixed_indication(
      USINT     Lpacket[],
      UINT      Lpacket_size,
      USINT     service,
      USINT     sourceID );
```

#### 6.2.2.2 Procedures for request and confirmation

The `Lpacket` parameter shall contain the DLSDU as an ordered sequence of octets. The `Lpacket_size` parameter shall specify the number of octets contained in the `Lpacket` array. The `QoS priority` shall specify onto which internal queue the request shall be placed and shall be one of `LOW`, `HIGH` or `SCHEDULED`. Only Lpackets on the `SCHEDULED` queue may be sent during the scheduled portion of the NUT. The unscheduled portion of the NUT shall be used to transmit Lpackets from the `HIGH` and `LOW` QoS priority queues, and may be used to transmit Lpackets from the `SCHEDULED` queue. The queues shall be prioritized such that, during an unscheduled transmit opportunity, the `SCHEDULED` queue is emptied first, then the `HIGH` QoS priority queue is emptied second, and finally the `LOW` QoS priority queue is emptied last.

The IDENTIFIER, `id`, shall correlate a confirmation with a corresponding request. The `status` parameter shall return the status of the attempted transmit and shall be one of `OK`, `TXABORT` or `FLUSHED`.

The `DLL_xmit_fixed_request` service queues an Lpacket with a two octet fixed tag for transmit. The `service` parameter shall specify the first octet of the fixed tag Lpacket. The `destID` parameter shall specify the second octet, which shall determine the MAC ID of the destination node.

The remaining parameter, `gentag`, of the `DLL_xmit_generic_request` service shall specify on which generic tag to transmit the Lpacket.

NOTE 1 The use of one or more queues for different priorities is an internal implementation choice.

NOTE 2 Assembly of the related Lpacket structure and the procedure for its sending on the link are described in their respective clauses (see 5.3 and 5.4).

#### 6.2.2.3 Procedures for indication

These indications shall signal the delivery of an Lpacket with a good FCS. Only fixed tag Lpackets which have had their service enabled by a successful call to `DLL_enable_fixed_request` shall cause a `DLL_recv_fixed_indication`. Only generic tag Lpackets which have had their generic tag enabled by a successful call to `DLL_enable_generic_request` shall cause a `DLL_recv_generic_indication`.

The array of octets, `Lpacket`, shall contain the DLSDU as an ordered sequence of octets. The integer, `Lpacket_size`, shall specify the number of octets contained in the `Lpacket` array.

### 6.2.3 Queue maintenance service

#### 6.2.3.1 Service encoding

Once queued, an Lpacket shall remain in a transmit queue until it is transmitted or de-queued. The request and confirmation services used to de-queue an Lpacket, before it is transmitted, shall be of the form:

```
DLL_flush-requests-by-QoS_request( PRIORITY priority );
DLL_flush-requests-by-QoS_confirm( PRIORITY priority );
DLL_flush_single_request( PRIORITY priority, IDENTIFIER id );
```

#### 6.2.3.2 Procedures for request and confirmation

`DLL_flush-requests-by-QoS_request` shall cancel all pending transmits at the specified QoS priority. Each pending transmit shall immediately terminate with a confirmation containing the FLUSHED status. `DLL_flush_single_request` shall cancel a specific Lpacket identified by the `id` used to transmit the Lpacket. This service need not have a corresponding confirmation since the confirmation of the queued Lpacket shall serve this purpose.

### 6.2.4 Tag filter service

#### 6.2.4.1 Service encoding

By default, the Data Link Layer shall only process the moderator Lpacket (fixed tag 0x00), and all other Lpackets shall be discarded. Other protocol layers may enable or disable reception of specific Lpackets using the following services;

```
DLL_enable_generic_request(
      IDENTIFIER   id,
      USINT     gentag[3] );

DLL_disable_generic_request(
      IDENTIFIER   id,
      USINT     gentag[3] );

DLL_enable_fixed_request(
      IDENTIFIER   id,
      USINT     service );

DLL_disable_fixed_request(
      IDENTIFIER   id,
      USINT     service );
```

```
DLL_enable_generic_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_disable_generic_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_enable_fixed_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_disable_fixed_confirm(
      IDENTIFIER   id,
      BOOL      result );
```

#### 6.2.4.2 Procedures for request and confirmation

The IDENTIFIER, `id`, shall correlate a confirmation with a corresponding request. The parameter, `result`, shall return TRUE if the request was successful and FALSE if

unsuccessful. The gentag parameter shall specify which generic tag to route to the higher layer that enabled the tag; likewise, the service parameter shall specify which fixed tag to route to the higher layer that enabled the tag.

NOTE The DLL_enable_generic_request service can return an unsuccessful result if the implementation of the Data Link Layer is unable to filter any more generic tags.

### 6.2.5 Link synchronization service

#### 6.2.5.1 Service encoding

The indication that a new network update time (NUT) has begun shall be of the form:

```
DLL_tone_indication( USINT cycle );
```

#### 6.2.5.2 Procedures for indication

The parameter cycle shall return the interval counter from the moderator DLPDU just received.

NOTE If a moderator DLPDU is expected but does not arrive, the local node simulates the reception of the moderator DLPDU based on an internal timer of the ACM.

### 6.2.6 Synchronized parameter change service

#### 6.2.6.1 Service encoding

All nodes on a link shall maintain two copies of link parameters: current and pending. The current copy of link parameters shall be used for the on going operation of the Data Link Layer. The pending copy shall be maintained to allow a synchronized change of link parameters. The local DLL user services that read and write these link parameters shall be of the form:

```
DLL_set_pending_request(
      IDENTIFIER      id,
      DLL_config_data params );

DLL_set_current_request(
      IDENTIFIER      id,
      DLL_config_data params );

DLL_get_pending_request(
      IDENTIFIER   id );

DLL_get_current_request(
      IDENTIFIER   id );
```

```
DLL_set_pending_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_set_current_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_get_pending_confirm(
      IDENTIFIER      id,
      DLL_config_data params );

DLL_get_current_confirm(
      IDENTIFIER      id,
      DLL_config_data params );
```

The DLL_config_data shall contain the link parameters and be of the form:

```
class DLL_config_data
{
   public:
   USINT myaddr;       // the MAC ID of this node
   UINT  NUT_length;   // the length of the NUT in 10 µs increments
   USINT smax;         // highest MAC ID allowed to transmit scheduled
   USINT umax;         // highest MAC ID allowed to transmit unscheduled
   USINT slotTime;     // time allowed for line turnaround in 1 µs increments
   USINT blanking;     // time to disable RX after DLPDU in 1,6 µs increments
   USINT gb_start;     // 10 µs intervals from start of guardband to tone
   USINT gb_center;    // 10 µs intervals from start of moderator to tone
   USINT modulus;      // modulus of the interval counter
   USINT gb_prestart;  // transmit cut-off, 10 µs intervals before tone
};                     // may not transmit passed this limit
```

The local DLL user services which request and confirm the start of a tMinus countdown shall be of the form:

```
DLL-tMinus-start-countdown-request(        DLL-tMinus-start-countdown-confirm(
      IDENTIFIER   id,                           IDENTIFIER   id,
      USINT     start_count );                   BOOL      result );
```

NOTE The tMinus fixed tag service (see 6.9) is used by the master Keeper to request all stations on the link to initiate a changeover countdown. This is the local DLL user procedure to request and confirm local participation.

At the correct expiry of the tMinus countdown, each participating station passes a local indication to its ControlNet object requesting a change from current link parameters to pending link parameters.

```
DLL_tminus_zero_indication( );
```

#### 6.2.6.2 Procedures for request, confirmation and indication

The IDENTIFIER, `id`, shall correlate a confirmation with a corresponding request. The `result` shall return TRUE if the request was successful and FALSE if unsuccessful.

The moderator Lpacket contains a field, called `tMinus`, that shall be used to synchronize the update of the current link parameters. The `DLL-tMinus-start-countdown-request` shall cause a node to participate in a `tMinus` countdown, and, if the node is the moderator, shall initialize the `tMinus` field of the moderator Lpacket. The moderator node shall decrement this field before transmitting each moderator until the field equals zero. When the `tMinus` field transitions from 1 to 0, the DLL in each node participating in the countdown shall locally generate a `DLL_tminus_zero_indication` and copy its pending link parameters into its current copy. If the `tMinus` field transitions to 0 from any value except 1, the countdown shall be aborted and no `DLL_tminus_zero_indication` shall be generated.

### 6.2.7 Event reports service

The indication used to alert the Station Management entity of various events internal to the Data Link Layer shall be of the form:

```
DLL_event_indication(
      DLL_event event,
      USINT     mac_id  /*[optional]*/ );
```

The DLL_event, `event`, shall be one of the events enumerated in Table 11. The `mac_id` parameter shall be used in the case of (`event` = DLL_EV_badFrame); it shall indicate the source MAC ID of the node that transmitted the bad DLPDU.

NOTE Since the DLPDU was damaged, the source MAC ID could be incorrect.

**Table 11 – DLL events**

| DLL events | Description |
|---|---|
| **DLL_EV_rxGoodFrame** | A good DLPDU was received. This shall include DLPDUs that contain no data (null DLPDUs), but shall exclude moderators |
| **DLL_EV_txGoodFrame** | A good DLPDU was transmitted. This shall include DLPDUs that contain no data (null DLPDUs), but shall exclude moderators |
| **DLL_EV_badFrame** | A damaged DLPDU was received. The optional parameter, which is the apparent source MAC ID of the offending node, shall also be reported |
| **DLL_EV_errA** | A bad DLPDU was received on channel A of the physical medium, or a good DLPDU was received on channel B and `ph_frame_indication` from channel A stayed false |
| **DLL_EV_errB** | A bad DLPDU was received on channel B of the physical medium, or a good DLPDU was received on channel A and `ph_frame_indication` from channel B stayed false |

| DLL events | Description |
|---|---|
| **DLL_EV_txAbort** | A transmit DLPDU was terminated with an abort sequence |
| **DLL_EV_NUT_overrun** | NUT is not large enough to accommodate all the scheduled traffic |
| **DLL_EV_dribble** | Scheduled Lpackets could not be sent during scheduled time |
| **DLL_EV_nonconcurrence** | An event was detected that indicates that this node is out of step with the access control protocol |
| **DLL_EV_rxAbort** | A DLPDU was received that was terminated with an abort sequence |
| **DLL_EV_lonely** | Have not heard a DLPDU from another node on the link for 8 NUTs |
| **DLL_EV_dupNode** | Another node on the link is using this node's MAC ID |
| **DLL_EV_noisePulse** | `ph_lock_indication` went true then false before `ph_frame_indication` went true, but `ph_lock_indication` was not true long enough to indicate a possibly damaged DLPDU |
| **DLL_EV_collision** | `ph_frame_indication` was true when this node was about to transmit |
| **DLL_EV_invalidModAddress** | A moderator was received from a node that does not have the lowest MAC ID on the link |
| **DLL_EV_rogue** | A moderator DLPDU was received that does not match the link configuration information at this node |
| **DLL_EV_deafness** | Cannot hear the moderator DLPDU even though other link traffic is present |
| **DLL_EV_supernode** | A moderator was received from MAC ID 0 |

### 6.2.8 Bad FCS service

The indication used to alert the Station Management entity that a received DLPDU had an invalid FCS shall be of the form:

```
DLL_badFCS_indication( CHANNEL channel );
```

The `channel` parameter shall indicate which PhL entity provided the DLPDU which was in error. This parameter shall be one of CHA or CHB, corresponding to PhL channel A and PhL channel B, respectively. This indication shall be provided, at most, once per error DLPDU per channel.

### 6.2.9 Current moderator service

The indication used to inform the Station Management entity which node is currently the moderator shall be of the form:

```
DLL_currentMod_indication( USINT mac_ID );
```

The `mac_ID` parameter shall indicate the MAC ID of the node that last transmitted a valid moderator DLPDU.

### 6.2.10 Power up and online services

#### 6.2.10.1 Service encoding

The request and confirmation which places the Data Link Layer on-line shall be of the form:

```
DLL_online_request( BOOL online );          DLL_online_confirm( BOOL online );
```

The indication that specifies that the Data Link Layer has completed its initialization shall be of the form:

```
DLL_powerup_indication( void );
```

#### 6.2.10.2 Procedures for request and confirmation

At power-up, the DLL shall wait until the `online` parameter equals TRUE. The DLL shall then begin the process of going on-line. When the `online` parameter is FALSE, the DLL shall go off-line at the end of the current NUT. When off-line the DLL shall not transmit, and shall ignore any link activity.

### 6.2.11 Enable moderator service

#### 6.2.11.1 Service encoding

The request and confirmation services that enable and disable the ability of a node to assume the moderator role shall be of the form:

```
DLL_enable_moderator_request( BOOL enable );
DLL_enable_moderator_confirm( BOOL enable );
```

#### 6.2.11.2 Procedures for request and confirmation

When the `enable` parameter is TRUE, the DLL shall enable the transmission of moderator DLPDUs. When the `enable` parameter is FALSE, transmission of moderator DLPDUs shall be disabled.

NOTE This request is used by station management entities to give the possibility for a new node to non-disruptively join a working link. The moderator switch over protocol does not tolerate the node with the lowest MAC ID suppressing moderator DLPDUs for an extended period. The Network Attachment Monitor (NAM) re-enables moderator transmission whenever another device is detected on the link.

### 6.2.12 Listen only service

#### 6.2.12.1 Service encoding

The request and confirmation services that allows a device to receive, but disables the ability of a device to transmit, shall be of the form:

```
DLL_listen_only_request( BOOL enable );
DLL_listen_only_confirm( BOOL enable );
```

#### 6.2.12.2 Procedures for request, confirmation and indication

When the `enable` parameter is TRUE, the Data Link Layer shall participate in the access protocol and transmit DLPDUs. When the `enable` parameter is FALSE, transmission of DLPDUs shall be disabled; however, the ability of the node to receive DLPDUs shall not be impaired.

## 6.3 Generic tag Lpacket

### 6.3.1 General

The generic tag Lpacket is used for transferring connection data. The connection ID or Generic Tag is a unique identifier for previously negotiated resources and parameters at its source, destination(s) and any intermediate transit points. Only the generic Tag is necessary to identify the message data.

### 6.3.2 Structure of the generic-tag Lpacket

The size and control fields shall follow the Generic Tag format. The address shall be the DLS-generic-tag provided by the local DL-generic-request service. The link data field shall contain the DLS-user-data provided by the DL-generic-request.

### 6.3.3 Sending and receiving the generic-tag Lpacket

The sending station shall send the Lpacket at the QoS priority specified by the DL-generic-request.

The status of each generic tag Lpacket transmission is confirmed to the DL user by returning the DLS-user-id (an internal handle or identifier) and the DLS Txstatus with one of the following values.

a) "OK" — message successfully sent;

b) "TXABORT" — sending process failed;

c) "FLUSHED" — message has been removed from the pending queue before sending.

NOTE 1 Txstatus is a local variable so the coding is not specified.

NOTE 2 The FLUSHED status is only used in response to a deletion requested by the Queue maintenance service.

NOTE 3 The parameter value OK is not an indication that the message has been received.

All receiving stations whose local Tag Filter service contains an address matching the received Lpacket address, shall pass each correctly received data unit to its local DL user, together with the data unit size and the address (the generic tag for the connection).

## 6.4 Moderator Lpacket

### 6.4.1 General

The moderator MAC Lpacket is used for DLL management, access timing and synchronization, it is sent as a moderator DLPDU containing only one Lpacket, the moderator Lpacket.

### 6.4.2 Structure of the moderator Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the moderator fixed tag and 0xFF the broadcast address for all MAC IC nodes on this link. The link data field shall contain the following in the listed order.

```
class moderatorLpacket: public fixedLpacket
{
    public:
    UINT    NUT_length;     // the length of the NUT in 10 µs increments
    USINT   smax;           // highest MAC ID allowed to transmit scheduled
    USINT   umax;           // highest MAC ID allowed to transmit unscheduled
    USINT   slotTime;       // 1 µs increments allowed for line turnaround
    USINT   blanking;       // 1,6 µs increments to disable RX after DLPDU
    USINT   gb_start;       // 10 µs intervals from start of guardband to tone
    USINT   gb_center;      // 10 µs intervals from start of moderator to tone
    USINT   usr;            // unscheduled start register
    USINT   interval_count; // current NUT number (0 to modulus)
    USINT   modulus;        // modulus of the interval counter
    USINT   tMinus;         // countdown to synchronized link parameter change
    USINT   gb_prestart;    // transmit cut-off, 10 µs intervals before tone
    USINT   reserved;       //
};
```

### 6.4.3 Sending and receiving the moderator Lpacket

The moderator DLPDU containing the moderator Lpacket shall be transmitted once per NUT by the node with the lowest MAC ID. It shall be sent in the guardband, a fixed part of every NUT reserved for the moderator DLPDU.

To support DLL management actions, the sender and receivers of the moderator DLPDU shall use the contents of the moderator link data field as follows.

The NUT_length shall specify the duration of the current NUT. NUT_length shall be a 16 bit integer representing the number of 10 μs ticks and shall be in the range 50 (500 μs) to 10 000 (100 ms). The beginning of the NUT shall be called the tone. At the tone, a counter that decrements every 10 μs shall be loaded with the NUT_length.

NOTE 1 Subclause 8.2 further constrains the value of NUT_length.

To guarantee that the link is quiet during moderator transmission, all nodes shall cease transmitting when their internal counter reaches the value of gb_prestart. All nodes may then assume that the link is quiet between gb_start and gb_center. When the moderator node's counter reaches the value of gb_center, it shall begin transmitting a DLPDU containing only the moderator Lpacket. The moderator Lpacket shall be completely transmitted at least 30 μs before the decrementing counter would have expired.

NOTE 2 Subclause 8.2 calculates the values of gb_prestart, gb_start, and gb_center to guarantee these constraints. Factors that affect the calculation of these values include clock accuracy, missing nodes, and propagation delay between nodes. In all cases, gb_prestart > gb_start > gb_center > 7.

The highest MAC ID transmitting during the scheduled portion of the NUT shall be smax. It shall be in the range 0 to 254. umax shall determine the highest MAC ID that transmits during the unscheduled portion of the NUT. It shall be in the range smax to 254. The unscheduled start register, usr, shall select which node transmits first in the unscheduled portion of the NUT. The usr shall increment each NUT. This counter shall be updated modulo (umax + 1).

A node shall listen to the PhL directly after it finishes transmitting. blanking shall determine the amount of time to suspend listening and shall be in 1,6 μs ticks.

The moderator node shall increment an 8 bit counter, called interval_count, once per NUT. This counter shall be updated modulo (modulus + 1). Since the guardband is at the end of the NUT, the value of interval_count shall correspond to the NUT that just completed.

NOTE 3 Subclause 8.2 and the specifications of the ControlNet object further constrain the value of smax, umax, blanking and modulus.

The tMinus field shall maintain a countdown until all nodes adopt new link parameters (see 6.9). The slotTime field shall determine the time to wait for a missing node and shall be in 1,0 μs ticks. An additional 1,0 μs shall be added to the value of the slotTime. The slotTime field shall be in the range 8 to 254; the other values shall be reserved. The reserved octet shall be set to zero.

## 6.5 Time distribution Lpacket

### 6.5.1 General

The moderator DLPDU provides a common reference marker that is synchronized between all nodes on the network. By distributing and processing time stamps relative to the reference instead of to the time distribution message, implementations are simplified whilst accuracy is improved by several orders of magnitude. Phase and frequency synchronization is inherent in this DL-protocol to a very high level of accuracy. The accuracy of time synchronization using the time distribution format defined in 6.5 is implementation dependent, however it can be better than 10 μs.

### 6.5.2 Structure of the time distribution Lpacket

#### 6.5.2.1 Packet format

The size and control fields shall follow the fixed tag format. The address shall be formed from the time distribution fixed tag and 0xFF the broadcast address. The link data field shall contain the following in the listed order:

```
class TimeDist Lpacket: public fixedLpacket
{
   public:
   USINT     revision;   // revision of time distribution format
   USINT     leap;       // leap second offset
   UINT      goodness;   // time relay control field
   DINT      gse;        // global squared error relative to ultimate master
   LINT      dctz;       // distribution channel time zero
   LINT      ts_ref;     // time stamp of previous reference pulse
   LINT      ts_tx;      // time stamp of this message's transmission
};
```

#### 6.5.2.2 Revision

The revision parameter shall be set to zero. This parameter shall represent the revision of the time distribution format.

#### 6.5.2.3 Leap

The leap parameter shall specify the Universal Coordinated Time (UTC) leap seconds. This number, when added to the system time, shall give actual UTC time. This number can increment or decrement by one twice a year on the solstices, as dictated by the US Naval Observatory, to maintain synchronism between terrestrial and astronomical time. If zero, then the number of leap seconds is unknown.

The leap parameter should not be used in any control situations, but may be needed in some time relays to distribution channels that are based on UTC rather than Global Positioning Satellite (GPS) time.

#### 6.5.2.4 Goodness

##### 6.5.2.4.1 Parameter structure

The goodness parameter shall be partitioned as shown in Figure 15.

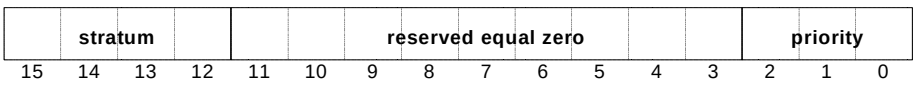



**Figure 15 – Goodness parameter of TimeDist_Lpacket**

##### 6.5.2.4.2 Stratum

The most significant field, called stratum, shall specify the number of time relays between this message, and a source of absolute time. A value of 0 shall signify an exact reference, and the value shall be incremented for every intermediate time relay. If the priority field is set to zero (lock not achieved), or the number of intermediate time relays exceeds 15, the goodness parameter shall be set to 15. Bits 3 through 11 shall be reserved and set to zero.

NOTE A time relay is a link-to-link router which distributes time synchronization messages on its links based on the time synchronization messages received on its other links.

#### 6.5.2.4.3 Source quality

The priority field shall indicate the quality of the time message as shown in Table 12.

**Table 12 – Time distribution priority**

| Value | Meaning |
|---|---|
| 7 | Absolute system time. Acting as master |
| 6 | Absolute system time. Acting as dependent |
| 5 | Human set system time. Acting as master |
| 4 | Human set system time. Acting as dependent |
| 3 | Lock established to node on distribution channel other than this one |
| 2 | Lock established to node on this channel. System time unknown |
| 1 | (invalid) |
| 0 | Not synchronized with any other node |

### 6.5.2.5 gse

The gse parameter shall indicate the cumulative rms stability squared. This parameter shall approximate the node's worst case stability relative to the rest of the system. The units of this parameter shall be $(0{,}1\ \mu s)^2$. When the rms stability is unknown or not yet determined, it shall be 0xFFFFFFFF.

### 6.5.2.6 LINT time parameters

In each of the LINT time parameters, the most significant bit shall be zero – only 63 bits shall be used. The units of these parameters shall be 0,1 μs.

### 6.5.2.7 dctz

The dctz parameter shall indicate the system time offset from the distribution channel's arbitrary time zero, established when the networks are synchronized.

### 6.5.2.8 ts_ref

The ts_ref parameter shall indicate the time stamp of the last tone following a moderator DLPDU which had its interval_count equal to zero. The value of zero shall indicate that this value is not known. System time zero shall be defined as that used for the Global Positioning Satellites: 12:00 midnight, Jan 6, 1980 GMT.

### 6.5.2.9 ts_tx

The ts_tx parameter shall indicate the time stamp at the transmission of this message. The value of zero shall indicate that this value is not known. System time zero shall be defined as that used for the Global Positioning Satellites: 12:00 midnight, Jan 6, 1980 GMT. This parameter shall be set to zero.

NOTE This protocol does not use GPS time, it only uses the time zero point for GPS. So the 1999 roll over of GPS had no relevance for system time.

## 6.5.3 Sending and receiving the time distribution Lpacket

The current master Keeper for the link is responsible to sent the time distribution Lpacket at intervals and QoS values configured by the DL user to match its application requirements.

All nodes are responsible to receive the time distribution Lpacket.

## 6.6 UCMM Lpacket

### 6.6.1 General

The Unconnected Message Manager (UCMM) is a DLS-user entity providing an unconnected request/response message service for at least one outstanding message. It uses the DLL transmit service and the UCMM Lpacket for its messages.

### 6.6.2 Structure of the UCMM Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the UCMM fixed tag and destination MAC ID provided by the DLL transmit service. The link data field shall contain the DLS user data provided by the DLL transmit service.

### 6.6.3 Sending and receiving the UCMM Lpacket

The UCMM Lpacket shall be sent at the QoS priority specified by the DLL data transfer service.

The status of each UCMM transmission is confirmed to the DL user by returning the DLS-user-id (an internal handle or identifier) and the DLS Txstatus with one of the following values.

a) “OK” — message successfully sent;

b) “TXABORT” — sending process failed;

c) “FLUSHED” — message has been removed from the pending queue before sending.

NOTE 1 Txstatus is a local variable so the coding is not specified.

NOTE 2 The FLUSHED status is only used in response to a deletion requested by the Queue maintenance service.

NOTE 3 The parameter value OK is not an indication that the message has been received.

The receiving station whose MAC ID matches that in the address, shall pass each correctly received data unit to its local DL user, together with the data unit size, the Lpacket service type and its source ID obtained from the DLPDU which conveyed the Lpacket.

## 6.7 Keeper UCMM Lpacket

### 6.7.1 General

Devices that implement the Keeper object shall also support sending and receiving of Keeper UCMM Lpackets. Only the master Keeper shall send the Keeper UCMM Lpacket.

NOTE Using a specific address for Keeper objects reduces the decoding activity required by devices which do not include a Keeper object.

### 6.7.2 Structure of the Keeper UCMM Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the Keeper UCMM fixed tag and 0xFF the broadcast address. The link data field shall contain the user data provided by the Keeper object.

### 6.7.3 Sending and receiving the Keeper UCMM Lpacket

The Keeper UCMM Lpacket shall only be sent by a station with its Keeper object in the master state. It shall be requested at the QoS priority specified by the invoking Keeper object.

The status of each Keeper UCMM transmission is confirmed to the DL user by returning the DLS-user-id (an internal handle or identifier) and the DLS Txstatus with one of the following values.

a) "OK" — message successfully sent;

b) "TXABORT" — sending process failed;

c) "FLUSHED" — message has been removed from the pending queue before sending.

NOTE 1 Txstatus is a local variable so the coding is not specified.

NOTE 2 The FLUSHED status is only used in response to a deletion requested by the Queue maintenance service.

NOTE 3 The parameter value OK is not an indication that the message has been received.

The receiving station whose MAC ID matches that in the address, shall pass each correctly received data unit to its local DL user, together with the data unit size, the Lpacket service type and its source ID obtained from the DLPDU which conveyed the Lpacket.

As this is an unconnected message, the source MAC ID is required to enable a response as appropriate.

## 6.8 TUI Lpacket

### 6.8.1 General

A Table Unique Identifier (TUI) is used by all ControlNet objects to reference a consistent set of link configuration attributes. Two separate sets of attributes are held, each with their own TUI, one for current operation and one for pending operation. Changes from TUI data in the current link configuration to the TUI data in the pending link configuration shall occur at the completion of a synchronized parameter change.

The TUI Lpacket is published as a scheduled Lpacket by the master Keeper to enable all attached devices to verify they have current configuration and schedule data.

### 6.8.2 Structure of the TUI Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the TUI fixed tag and 0xFF the broadcast address. The link data field shall contain the listed items shown in Table 13.

**Table 13 – Format of the TUI Lpacket**

| Name | Data type | Description of parameter | Semantics of values |
|---|---|---|---|
| **Size** | USINT | Size of the Lpacket in words | 0x0D |
| **Control** | USINT | Link layer Lpacket control octet | 0x01 (Fixed tag Lpacket) |
| **Fixed_Tag** | USINT | Fixed tag value | 0x84 (TUI Lpacket) |
| **Destination_Mac_Id** | USINT | Who receives the Lpacket | 0xFF (Broadcast) |
| **Unique_Id** | UDINT | CRC of the Keeper's current attributes | Least significant octet first |
| **Status_Flag** | UINT | TUI flag values | See the Keeper object TUI attribute for details |
| **Keeper_Mac_Id** | USINT | MAC ID of the Keeper broadcasting the TUI | See the Keeper object TUI attribute for details |
| **Reserved** | USINT | Reserved for data alignment | |
| **Net_Resource_Vendor_Id** | UINT | Vendor ID of object holding Net Resource for exclusive use | |

| Name | Data type | Description of parameter | Semantics of values |
|---|---|---|---|
| **Net_Resource_Serial_Number** | UDINT | Serial number of object holding Net Resource for exclusive use | |
| **Net_Resource_Class** | UDINT | Class number of object holding Net Resource for exclusive use | |
| **Net_Resource_Instance** | UDINT | Instance number of object holding Net Resource for exclusive use | |

When generating the transmitted TUI Lpacket, any reserved fields shall use the values as specified in the corresponding reserved fields of Keeper object attribute 0x0101. During the configuration process, the programming tool shall set any reserved fields in attribute 0x0101, to zero.

### 6.8.3 Sending and receiving the TUI Lpacket

The requested QoS priority shall be scheduled.

When the Keeper object is in the MASTER_VERIFY, FAULTED_MASTER_VERIFY, MASTER, FAULTED_MASTER, NET_CHANGE_MASTER or NET_CHANGE_FAULTED_MASTER operating state, it shall attempt to transmit a Table Unique Identifier (TUI) Lpacket as scheduled data once every 4 NUTs (on NUT numbers 0, 4, 8, 12, 16 …). All Keeper devices shall reserve 26 octets of scheduled link transmit time once every 4 NUTs for transmitting this Lpacket. (If sent unscheduled, it shall be the highest QoS priority unscheduled information.)

Reception of TUI Lpackets shall be enabled at power up by the ControlNet object in each device. Received TUI Lpackets shall be passed to the local ControlNet object.

## 6.9 Link parameters Lpacket and tMinus Lpacket

### 6.9.1 General

To synchronize the changing of local link parameters, Station Management shall enable the reception of two fixed tag Lpackets: 0x15 (tMinus) and 0x81 (link parameters) via the `DLL_enable_fixed_request` service of the DLL.

All nodes on a link shall maintain two copies of link parameters: current and pending. The current copy of link parameters shall be used for the on going operation of the Data Link Layer. The pending copy shall be maintained to allow a synchronized change of link parameters. The change synchronization is supported by the tMinus service.

### 6.9.2 Structure of link parameters and tMinus Lpackets

The size and control fields shall follow the fixed tag format. The address shall be formed from the fixed tag for the service and 0xFF the broadcast address. The link data field for the appropriate Lpacket shall contain the following contents in the listed order.

a) Contents of the Link parameters Lpacket data field:

```
class LinkParm_Lpacket: public fixedLpacket
{
   public:
   UINT  NUT_length;   // the length of the NUT in 10 µs increments;
   USINT smax;         // highest MAC ID allowed to transmit scheduled
   USINT umax;         // highest MAC ID allowed to transmit unscheduled
   USINT slotTime;     // time allowed for line turnaround in 1 µs increments
   USINT blanking;     // time to disable RX after DLPDU in 1,6 µs increments
   USINT gb_start;     // 10 µs intervals from start of guardband to tone
   USINT gb_center;    // 10 µs intervals from start of moderator to tone
   UINT  zero;         // reserved
   USINT modulus;      // modulus of the interval counter
   USINT gb_prestart;  // transmit cut-off, 10 µs intervals before tone
   UDINT unique_id;    // 32 bit CRC calculated from Keeper configuration table
   UINT  status_flag;  // 16 bit status table set by the Keeper
   UINT  reserved[8];  // all zeros
};
```

NOTE The format of the instance attributes 0x80 and 0x81 of the ControlNet object are exactly the same as the link data of a fixed tag 0x81 Lpacket.

b) Contents of the tMinus Lpacket data field:

```
class tMinus_Lpacket: public fixedLpacket
{
   public:
   USINT     start_count;
   USINT     reserved[3];
};
```

where start count is a number of NUTs to be counted before the transition and shall not be less than 8 and the reserved field shall contain zeros.

### 6.9.3 Sending and receiving the tMinus and Link parameters Lpackets

The Link parameters and tMinus Lpackets are issued by the master Keeper when a change is required to the link parameters. They shall be requested at the QoS priority specified by the invoking Keeper object.

Upon the reception of a `LinkParm_Lpacket`, each node shall load the pending copy of its link parameters using their local `DLL_set_pending_request` service. The pending parameters shall be loaded within 3 unscheduled transmit opportunities or within 1 s, whichever is greater. The pending link parameter attribute of the ControlNet object shall also be updated with the pending copy of the link parameters.

Upon reception of a `tMinus_Lpacket`, the node shall initiate a synchronized parameter change using the `DLL-tMinus-start-countdown-request` service passing the `start_count` parameter. The ControlNet object shall copy its pending link parameter attribute to its current link parameter attribute when the `DLL_tminus_zero_indication` signals the completion of the synchronized parameter change.

NOTE The DLL copies its own pending link parameters to its current copy at the completion of the tMinus countdown and no `DLL_tminus_zero_indication` is signaled at the completion of the countdown unless the completion is preceded by a `DLL-tMinus-start-countdown-request`..

The moderator Lpacket contains a field, called `tMinus`, that shall be used to synchronize the update of the current link parameters. The `DLL-tMinus-start-countdown-request` shall cause a node to participate in a `tMinus` countdown, and, if the node is the moderator, shall initialize the `tMinus` field of the moderator Lpacket. The moderator node shall decrement this field before transmitting each moderator until the field equals zero. When the `tMinus` field transitions from 1 to 0, the DLL in each node participating in the countdown shall locally generate a `DLL_tminus_zero_indication` to its ControlNet object which shall copy its pending link parameters into its current copy. If the `tMinus` field transitions to 0 from any

value except 1, the countdown shall be aborted and no DLL_tminus_zero_indication shall be generated.

### 6.10 I'm-alive Lpacket

#### 6.10.1 General

This service allows all nodes on a link to recognize that another node has just been reset or power cycled.

#### 6.10.2 Structure or the I'm-alive Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the I'm alive fixed tag and 0xFF the broadcast address. The link data field shall contain the following in the listed order.

```
class Im_Alive_Lpacket: public fixedLpacket
{
   public:
   USINT     Lpacket_variant;
   USINT     sourceID;
   USINT     reserved[6];
};
```

The Lpacket_variant and reserved fields shall be set to zero. The sourceID field shall be set to MAC ID of the node which transmits the Lpacket.

#### 6.10.3 Sending and receiving I'm Alive

The QoS priority shall be specified by the invoking DL user.

When a node is first brought on-line before starting full operations, the network attachment monitor (NAM) in the new node shall broadcast at least 3 Im_Alive_Lpackets, subject to the rules for I'm Alive the state processing.

Receiving stations shall notify their management services.

NOTE Among the clients of these broadcasts are the higher layer connection mangers of every other node on the link.The entities will abort any transaction records associated with the newly activated MAC ID and cancel any connections which had passed through the newly activated MAC ID. Since connection IDs (generic tags) are allocated dynamically, this ensures that connection ID's issued by the old device at the MAC ID are not improperly reused.

#### 6.10.4 I'm alive state processing

A node shall only transmit one Im_Alive_Lpacket per NUT. To minimize the processing requirements in each node, a node which is broadcasting its three Im_Alive_Lpackets shall not count as a valid transmission any Im_Alive_Lpacket that is preceded in the NUT by more than seven other such Lpackets. These Lpackets shall be transmitted in the unscheduled portion of the NUT (QoS priority of either LOW or HIGH).

Because the timing of when nodes power up or reset relative to each other cannot be controlled, there is the possibility of an I'm alive Lpacket broadcast storm if a large number of nodes enter the I'm alive state at the same time. To minimize processing requirements, a transmission smearing and back off algorithm is recommended to monitor and control the intensity of any broadcast storm activity. Such an algorithm would spread the I'm alive Lpackets out over longer and longer periods of time, should a large number of nodes power up concurrently. If a small number of nodes power up together, the algorithm should allow them to exit the I'm alive state quickly since there cannot be a broadcast storm. An example of one such algorithm is shown in Figure 16.

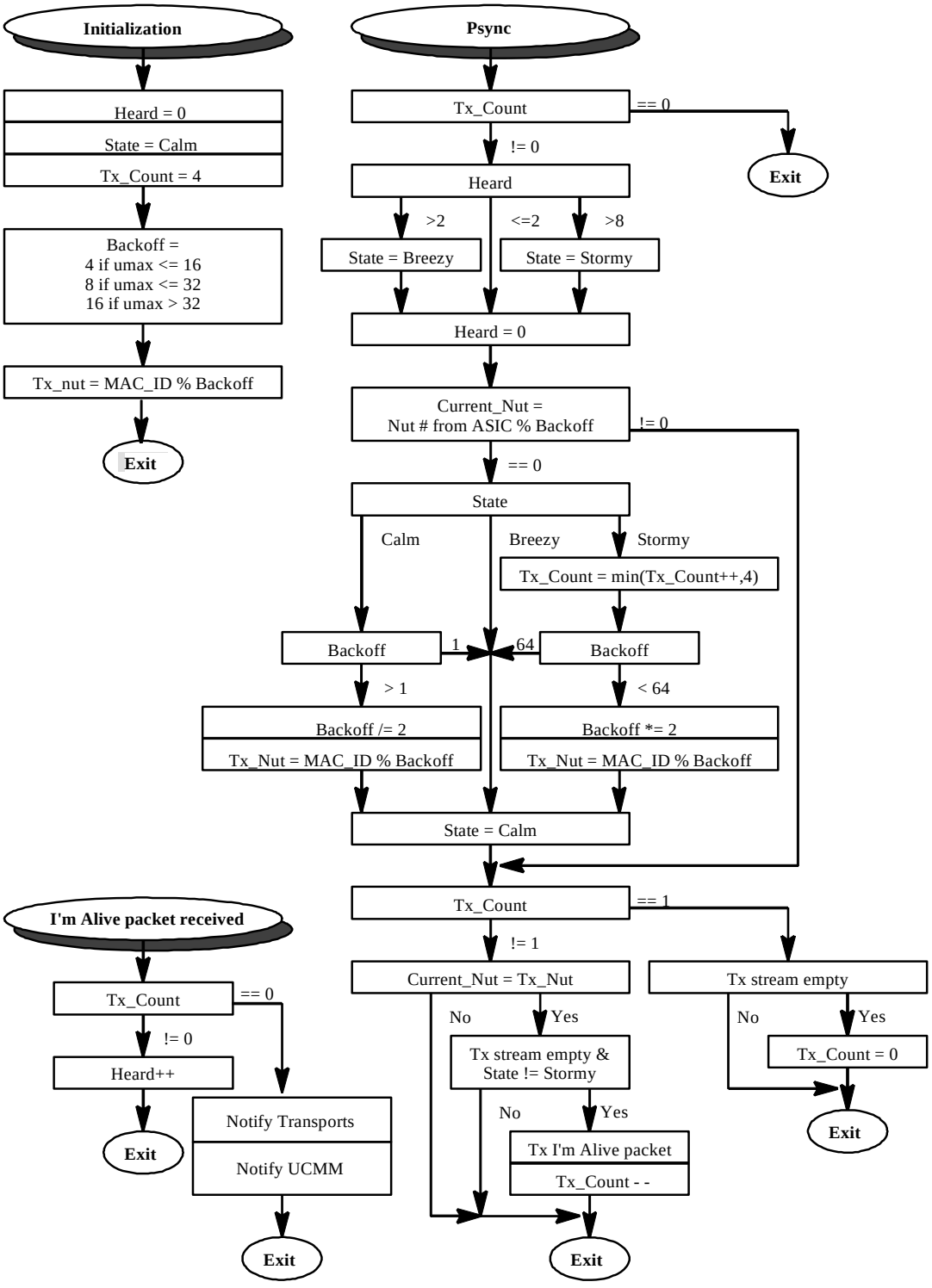



**Figure 16 – Example I'm alive processing algorithm**

## 6.11 Ping Lpackets

### 6.11.1 General

The ping service allows station management to initiate a ping request and receive a ping response from each active station, or one addressed station attached to the DLS provider.

All stations contain the ping service and at power-up the enable-fixed-request service is used by the DLS user to activate reception and response to ping requests within the local DLS provider.

### 6.11.2 Structure of the ping Lpackets

The size and control fields shall follow the fixed tag format.

a) When sending a ping request, the address shall be formed from the ping request fixed tag and an address specified by the DL user. The specified address may be a single MAC ID or 0xFF the broadcast address.

   The link data field shall contain the following in the listed order.

```
class Ping_request: public fixedLpacket
{
   public:
   USINT     client_ID;
   USINT     reserved[3];
};
```

   where client_ID is the MAC ID of the request originator, and the reserved field shall contain zeros.

b) When sending a ping reply, the address shall be formed from the ping reply fixed tag and the client_ID address from the corresponding ping request indication, (used here as destination for the reply).

   The link data field shall contain the following in the listed order;

```
class Ping_reply: public fixedLpacket
{
   public:
   USINT     server_ID;
   USINT     vendor_specific[4];
   USINT     reserved[3];
};
```

   where the server_ID field shall contain the MAC ID of the node responding to the request, the vendor_specific field may contain any data and the reserved field shall contain zeros.

The `DLL_xmit_fixed_request(id, Lpacket, sizeof(Ping_reply), HIGH, 0x29, client_ID)` service shall be used to send the reply to the MAC ID specified in the `client_ID` field of the received `Ping_request` Lpacket.

NOTE A typical use for the vendor_specific field is for debugging or diagnostic information.

### 6.11.3 Sending and receiving the ping Lpackets

The QoS priority for a ping request shall be specified by the invoking DL user. The QoS priority for a ping response shall be High.

Sending of ping-request Lpackets may be initiated by local management in any station.

At power up, the `DLL_enable_fixed_request(id, 0x09)` internal service of the Data Link Layer shall be invoked to enable the reception of fixed tag ping requests.

Upon receipt of a ping-request, each receiving station assembles a ping reply fixed tag Lpacket and sends it to the requesting station.

Upon receipt of a ping response, the requesting node passes the indication to local management.

### 6.12 WAMI Lpacket

#### 6.12.1 General

A “Where Am I?” (WAMI) server shall be present in all permanent nodes which have a Network Access Port (NAP). The server shall not be present on transient nodes.

NOTE The WAMI (where am I) server enables a transient node (a node which connects to the network via the network access port or NAP) to determine the MAC ID of the node to which it is attached. This is a convenient way for software in a transient device to establish communication with the local permanent node.

#### 6.12.2 Structure of the WAMI Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the WAMI fixed tag and 0xFF the broadcast address for all MAC IC nodes on this link. The link data field shall contain the following in the listed order.

```
class WAMI_Lpacket: public fixedLpacket
{
   public:
   USINT     requestID;
   USINT     responseID;
   USINT     reserved[6];
};
```

In a request WAMI, the `requestID` shall contain the MAC ID of the transient node that is sending the Lpacket. The other fields shall contain zeros. To convert a request into a response, the server shall copy its own MAC ID into the `responseID` field and reply. The QoS priority is Low.

NOTE 1 After WAMI fixed tag reception is enabled by the `DLL_enable_fixed_request(id, 0x86)` service of the DLL, fixed tags are received by the `DLL_recv_fixed_indication` service..

NOTE 2 The Network Attachment Monitor (see 8.1) describes procedures to ensure the transient MAC ID does not duplicate an existing link MAC ID.

#### 6.12.3 Sending and receiving the WAMI Lpacket

The QoS priority shall be Low.

The WAMI server is only implemented in devices having a NAP, these devices shall repeat all messages from the main link to the NAP and all messages from the NAP to the main link. Additionally these devices shall include means of detecting messages received via their NAP, and they shall only respond to Lpackets which match each of the following criteria:

– WAMI fixed tag 0x86;

– destination broadcast (0xFF);

– received via the device local NAP.

Lpackets that match these criteria shall be called request WAMI Lpackets. For all such Lpackets, the server shall generate a response WAMI that is addressed to the MAC ID which sent the request WAMI. The WAMI server shall discard all other WAMI fixed tag 0x86 Lpackets.

### 6.13 Debug Lpacket

This service may be used to transmit the state of an object building a trace of object state transitions on the wire.

The size and control fields shall follow the fixed tag format. The address shall be formed from the debug fixed tag and destination MAC ID provided by the DLL transmit service. The link data field shall contain the following in the listed order.

```
class Debug_Lpacket: public fixedLpacket
{
   public:
   UINT      object_class;
   UINT      data1;
   UINT      data2;
};
```

The value of `object_class` shall be the class code (see IEC 61158-6-2) of the object transmitting its state. The meaning of the `data1` and `data2` fields shall be defined in the object definition. Devices shall not generate more than one fixed tag debug Lpacket per 10 s on average so as not to interfere with other unscheduled transmissions.

### 6.14 IP Lpacket

This service (fixed tag 0x85) is used to send raw IP frames. The maximum IP frame size that can be sent in an Lpacket is 506 octets. Broadcast are sent with destination 0xFF. IP frames between two nodes shall have the correct fixed tag destination MAC ID. An address resolution protocol shall be used to determine the MAC ID associated with a particular IP address.

### 6.15 Ethernet Lpacket

This service (fixed tag 0x89) is used to send Ethernet frames other than IP frames. This includes but is not limited to Ethernet Address Resolution Protocol (ARP) frames. This allows simple implementation of IP and an Ethernet style address resolution protocol on Type 2 physical layer since an existing TCP/IP/Ethernet stack can be used with minor packet changes.

The lowest octet of the Ethernet physical address is used to represent the Type 2 MAC ID (Ethernet physical addresses are sent highest octet first). A Type 2 broadcast (MAC ID 0xFF) shall be expanded to an Ethernet broadcast address (FF FF FF FF FF FF).

On start up a TCP/IP stack that includes Ethernet ARP shall know the physical address of the local node. This is reported as “00 00 00 00 00 <MAC ID>”.

The low octet of an Ethernet frame’s “destination address” is used as the Type 2 fixed tag destination. The low octet of an Ethernet frame’s “source address” will be the same as the local node’s MAC ID (reported to the TCP/IP stack at startup) and is sent as the DLPDU source MAC ID. The Ethernet “frame type” (08 06 for ARP) make up the first two octets of the data portion of the Lpacket and up to 504 octets are available for the data portion of the Ethernet frame (ARP request/reply uses 28 octets).

Because the Ethernet “frame type” is included in the Lpacket, other types of Ethernet frames such as RARP can be sent using this Ethernet fixed tag service as well.

## 7 Objects for station management

### 7.1 General

This protocol specification is quite flexible. It can provide deterministic and synchronized I/O transfer at cyclic intervals up to 1 ms and node separations up to 25 km. This performance is adjustable on-line by configuring the link parameters. These parameters, which govern the access to the link, can be tuned as required to match different applications.

Station Management allows these parameters to be changed on-line, as the network is working, it also allows the link to continue functioning while connections to new nodes are added and removed.

The functions of Station Management allow

– access to variables and events within each of the Layers;
– a common user interface;
– coordinating the change of link parameters;
– non-disruptively adding nodes to the link;
– tuning of link parameters;
– clock synchronization between nodes.

The access to variables and events within each of the Layers is accomplished by defining object interfaces for these parameters.

The ControlNet object provides a consistent interface to the Physical and Data Link Layers.

The Keeper object holds the link attributes for all devices on a link and shall be responsible for distributing those attributes to those devices in an orderly fashion. It also holds (for link scheduling software) a copy of the connection originator data for all connection originator devices using a network. If there are multiple Keeper objects on a link, they perform negotiations to determine which Keeper is the master Keeper and which Keeper(s) perform backup Keeper responsibilities. The master Keeper is the Keeper actively distributing link parameters and attributes to the nodes on the network. A backup Keeper is one that monitors Keeper related network activity and can transition into the role of master Keeper should the master Keeper fail to perform.

The Keeper object also contains support for obtaining, holding, and releasing the Network Resource, a network semaphore that is used to eliminate conflicts when modifying the attribute data held by the Keeper object(s) on a link.

The Scheduling object shall exist in connection originator (CO) devices. The Scheduling object shall be used by link scheduling software (LSS) to

– read scheduled connection information;
– write schedule information;
– provide the CO with a signature (software key) which shall be used to authenticate the CO device's access to a specific link.

The TCP/IP Interface object provides the mechanism to configure the TCP/IP interface of a device.

The Ethernet Link object maintains link-specific counters and status information for a physical Ethernet ISO/IEC 8802-3 port.

The DeviceNet object provides a consistent interface to the Physical and Data Link Layers.

The Connection Configuration object provides an interface to create, configure and control connections in a device.

The Device Level Ring (DLR) object provides an interface to configure and monitor the DLR protocol.

The QoS object provides an interface to configure certain QoS-related behaviors.

The Port object describes the Type 2 ports present on the device.

## 7.2 ControlNet object

### 7.2.1 Overview

The ControlNet object shall provide a consistent Station Management interface to the Physical and Data Link Layers. This object shall make diagnostic information from these layers available to client applications. Each node shall support one ControlNet object per link.

A device may contain more than one node.

### 7.2.2 Class attributes

The ControlNet object shall support the class attributes as specified in Table 14.

**Table 14 – ControlNet object class attributes**

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| **0x01** | Required | Get | Revision | UINT | Revision of this object | First revision, value = 1 |
| **0x02** | Required | Get | Max. instance | UDINT | Maximum instance number | Value determined by node specifics |

### 7.2.3 Instance attributes

#### 7.2.3.1 General

The ControlNet object shall support the instance attributes as specified in Table 15.

**Table 15 – ControlNet object instance attributes**

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| 0x80 | Optional | Get | Pending_Link_Config | STRUCT of 34 octets | pending link configuration parameters | see 6.9.2 |
| | | | Link_Config | STRUCT of 12 octets | pending link parameters | |
| | | | NUT_length | UINT | DLL NUT_length | in 10 μs ticks |
| | | | smax | USINT | DLL smax | 0 to 99 |
| | | | umax | USINT | DLL umax | 1 to 99 |
| | | | slotTime | USINT | DLL slotTime | in 1μs ticks |
| | | | blanking | USINT | DLL blanking | in 1,6 μs ticks |
| | | | gb_start | USINT | DLL gb_start | in 10 μs ticks |
| | | | gb_center | USINT | DLL gb_center | in 10 μs ticks |
| | | | reserved | UINT | reserved | |
| | | | modulus | USINT | DLL modulus | 127 required |
| | | | gb_prestart | USINT | DLL gb_prestart | in 10 μs ticks |
| | | | TUI | STRUCT of 22 octets | Keeper TUI | see 7.2.3.3 |
| | | | unique_ID | UDINT | Keeper CRC | see 7.2.3.4 |
| | | | status_flag | UINT | TUI flag | see 7.2.3.5 |

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| | | | reserved | USINT [16] | reserved | |
| 0x81 | Required | Get | current_link_config | STRUCT of 34 octets | current link configuration parameters | same as attribute 0x80 see 6.9.2 |
| 0x82 | Required | Get/ Get_and _Clear | diagnostic_counters | STRUCT of 42 octets | diagnostic counters | see 7.2.3.7 |
| | | | buffer_errors | UINT | buffer event counter | see 7.2.3.8 |
| | | | error_log | SWORD[8] | bad DLPDU log | see 7.2.3.9 |
| | | | event_counters | STRUCT of 32 octets | diagnostic counters | see 7.2.3.10 |
| | | | good_frames_ transmitted | SWORD[3] | good DLPDUs transmitted (LSB first) | |
| | | | good_frames_ received | SWORD[3] | good DLPDUs received (LSB first) | |
| | | | selected_channel_ frame_errors | USINT | framing errors detected on active receive channel | |
| | | | channel_A_ frame_errors | USINT | framing errors detected on channel A | |
| | | | channel_B_ frame_errors | USINT | framing errors detected on channel B | |
| | | | aborted_frames_ transmitted | USINT | DLPDUs aborted during transmission (transmit underflows) | |
| | | | highwaters | USINT | LLC transmit underflow and LLC receive overflow | |
| | | | NUT_overloads | USINT | no unscheduled time in NUT (all time used for scheduled transmissions) | |
| | | | slot_overloads | USINT | more scheduled data queued for one NUT than allowed by sched_max_frame parameter | |
| | | | blockages | USINT | single Lpacket size exceeds sched_max_ frame parameter | |
| | | | non_concurrence | USINT | two or more nodes could not agree whose turn it is to transmit | |
| | | | aborted_frames_ received | USINT | incomplete DLPDUs received | |
| | | | lonely_counter | USINT | number of times nothing heard on network for 8 or more NUTs | |
| | | | duplicate_node | USINT | DLPDU received from node with local node's MAC ID | |

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| | | | noise_hits | USINT | noise detected that locked the modem rx PLL | |
| | | | collisions | USINT | Rx data heard just as we were going to transmit | |
| | | | mod_MAC_ID | USINT | MAC ID of the current moderator node | |
| | | | non_lowman_mods | USINT | Moderator DLPDUs heard from non lowman nodes | |
| | | | rogue_count | USINT | rogue events detected | |
| | | | unheard_moderator | USINT | DLPDUs being heard but no moderators being heard | |
| | | | vendor_specific | USINT | vendor specific | |
| | | | reserved | SWORD[4] | reserved | |
| | | | vendor_specific | USINT | vendor specific | |
| | | | vendor_specific | USINT | vendor specific | · |
| | | | reserved | SWORD | reserved | |
| 0x83 | Required | Get | station_status | STRUCT of 6 octets | station status | see 7.2.3.11 |
| | | | MAC_ver | SWORD | MAC implementation and implementation revision | see 7.2.3.12 |
| | | | vendor_specific | SWORD[4] | vendor specific | |
| | | | channel_state | SWORD | channel state LEDs, redundancy warning, and active channel bits | see 7.2.3.12 |
| 0x84 | Get required, Set-(One Time Only) Optional | Get/Set | MAC_ID | STRUCT of 4 octets | MAC ID switch and current settings | |
| | | | MAC_ID_current | USINT | current MAC ID | Range 1 to 99. See 7.2.3.14 |
| | | | MAC_ID_switches | USINT | MAC ID switch settings | 0 to 99, 0xFF. See 7.2.3.15 |
| | | | MAC_ID_changed | BOOL | MAC ID switches changed since reset | see 7.2.3.16 |
| | | | Reserved | USINT | reserved | |
| 0x85 | Optional | Get/Set | Sched_max_frame | STRUCT of 2 octets | scheduled data limit | |
| | | | Sched_max_frame | USINT | scheduled maximum DLPDU | in octet pairs. See 7.2.3.17 |
| | | | Reserved | USINT | reserved | |
| 0x86 | Required | Get | error_log | STRUCT of 10 octets | driver firmware buffer error counts and troublesome node list | see 7.2.3.18 |
| | | | buffer_errors | UINT | buffer event counter | |
| | | | error_log | SWORD[8] | bad DLPDU log | · |

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| 0x87 | Optional | Get / Get_and_Clear | extended_diagnostic_ counters | STRUCT of 264 octets | additional diagnostic counters | |
| | | | unsched_transmitted | UDINT | number of unscheduled Lpackets transmitted | |
| | | | sched_highwater | UDINT | Max. number of shared/unscheduled Lpackets in transmit queue | |
| | | | sched_data | 128 STRUCTS of 2 octets (256 octets) | schedule information | |
| | | | words_in_use | USINT | number of scheduled words in use by this node this NUT | |
| | | | Lpkt_in_use | USINT | number of scheduled Lpackets in use by this node this NUT | |
| 0x88 | Optional | Get | Active_node_table | ARRAY of 32 octets | one bit for each MAC ID<br>1 = node present<br>0 = node missing | see 7.2.3.19 |
| 0x89 | Optional | Get | New_node_table | ARRAY of 32 octets | one bit for each MAC ID<br>1 = node has joined link recently<br>0 = node has not joined link recently | see 7.2.3.20 |
| where LSB = least significant byte/octet. | | | | | | |

#### 7.2.3.2 pending_link_config

The values within the attribute allow the pending link parameters to be read. This attribute shall change based on the Station Management function called synchronized parameter change, and shall not be set via the services of the ControlNet object.

#### 7.2.3.3 Keeper TUI

The ControlNet object shall maintain separate copies of the TUI sub–attribute for pending_link_config and current_link_config attributes. Changes to the TUI data in the pending_link_config attribute shall occur at the reception of a fixed tag 0x81. Changes to the TUI data in the current_link_config attribute shall occur at the completion of the synchronized parameter change.

#### 7.2.3.4 unique_ID

The unique_ID field shall consist of a 32 bit CRC value calculated from the contents of the network configuration table maintained by the Keeper. This value shall be copied from the fixed tag 0x81 Lpacket.

#### 7.2.3.5 status_flag

The status_flag field of the TUI sub–attribute shall consist of a 16 bit value. Unused bits shall be reserved and set to zero.

As shown in Table 16, the Keeper State bit (bit 4) shall indicate whether this node has ever heard a TUI from a master Keeper. The ControlNet object shall clear this bit at initialization. Since all TUI Lpackets sent by the master Keeper have this bit set, the ControlNet object need not manipulate this bit.

Reception of TUI Lpackets shall be enabled at power up using the `DLL_enable_fixed_request(id, 0x84)` service. The status field of the TUI Lpackets shall be checked. If bit3 = 0, bit4 = 1, and bit5 = 1, then the TUI Lpacket shall be copied into attribute 0x81 and TUI reception shall be disabled using the `DLL_disable_fixed_request(id, 0x84)`. Bits 0 and 1 of the TUI status field shall be used to override the default network redundancy settings for the node. They shall indicate if the network is being operated single channel or redundant channel mode. Only network redundancy information from TUI Lpackets with both the Keeper State and Keeper configured bits set, shall be used.

**Table 16 – TUI status flag bits**

| Bit | Meaning |
|---|---|
| 0 - 1 | Network Redundancy<br>Bit 1 = 0, bit 0 = 0; Illegal combination<br>Bit 1 = 0, bit 0 = 1; Channel B only<br>Bit 1 = 1, bit 0 = 0; Channel A only<br>Bit 1 = 1, bit 0 = 1; Redundant |
| 2 | Holding Network Resource bit<br>0 = Network Resource not being held<br>1 = Network Resource being held for the object identified in the TUI |
| 3 | Network change in progress bit<br>0 = No network change in progress<br>1 = Network change in progress |
| 4 | Keeper State bit<br>0 = TUI transmitted by Keeper not in MASTER state<br>1 = TUI transmitted by Keeper in MASTER state |
| 5 | Keeper configured bit<br>0 = No master Keeper heard since joining link<br>1 = Heard master Keeper since joining link |
| 6 - 15 | reserved ( set to 0 ) |

#### 7.2.3.6 current_link_config

The values within the current_link_config attribute correspond to the link parameters currently used. The format of this attribute shall be identical to that of the pending_link_config attribute.

#### 7.2.3.7 diagnostic_counters

The diagnostic_counters attribute shall maintain counts of events in the Physical and Data Link Layers.

NOTE The diagnostic_counters attribute can be read without effecting their values by using one of the Get_Attribute service requests. The information can be read and the values set to zero by using the Get_and_Clear service request. See 7.2.5.

The event information maintained by the firmware shall also be gettable (but not clearable) via the error_log attribute. This is done for efficiency. The information maintained by the firmware shall be immediately and directly accessible.

#### 7.2.3.8 buffer_errors

The value of the buffer_errors sub–attribute shall increment on every receive overflow event and transmit underflow event. If an implementation never overflows or underflows, buffer_errors shall be zero.

#### 7.2.3.9 error_log

If `DLL_event_indication` returns a value of `DLL_EV_badFrame`, the error_log sub-attribute records the last eight such indications. The `mac_id` parameter of this indication shall be recorded. The order of MAC IDs shall be most recent error first to oldest error last. Values of zero represent unused positions in the error log.

#### 7.2.3.10 event_counters

The event_counters sub-attribute shall contain counts of link events. The counter shall rollover when their values exceed their limits.

#### 7.2.3.11 station_status

This attribute shall be only gettable.

#### 7.2.3.12 MAC_ver

The bits of the MAC_ver octet are described in Table 17.

**Table 17 – Mac_ver bits**

| Bits | Description |
|---|---|
| 0-3 | **MAC revision**[a] – revision of the MAC implementation<br>0x1 through 0xF, 0x0 is reserved |
| 4 | Reserved – set to 0 |
| 5-7 | **MAC implementation**[a] – vendor and implementation name<br>0 - 2 = assigned<br>3 = reserved – unassigned for future implementations<br>4 = assigned<br>5 - 6 = reserved – unassigned for future implementations<br>7 = reserved for extended MAC_ver field expansion |
| [a] MAC revision and MAC implementation are assigned and described by ODVA, Inc. | |

#### 7.2.3.13 channel_state

The channel_state octet shall represent the current state of the network status indicators, if implemented. The bits of the channel_state octet are described in Table 18.

NOTE The network status indicators are described in Annex A.

**Table 18 – Channel state bits**

| Bits | Description |
|---|---|
| 0,1,2 | Channel A LED State<br>0 = Off<br>1 = Solid green<br>2 = Flashing green-off<br>3 = Flashing red-off<br>4 = Flashing red-green<br>5 = Railroad red-off<br>6 = Railroad red-green<br>7 = Solid red |
| 3,4,5 | Channel B LED State<br>Indications same as channel A LED. |
| 6 | **Redundancy Warning** – When warning, the non–active channel in a redundant configuration is unusable by the controller.<br>0 = normal (no warning)<br>1 = warning |
| 7 | **Active Channel** – Indicates which of two channels the receiver is currently listening to.<br>0 = Channel B<br>1 = Channel A |

#### 7.2.3.14 MAC_ID_current

The MAC_ID_current value shall be accessible using the Get_Attribute services. Setting shall be required on devices without address switches. Only the first set, of the MAC_ID_current value, after a ControlNet object reset shall be accepted. Setting shall be optional on devices with address switches.

The range of allowable values for MAC IDs shall be 1 to 99.

#### 7.2.3.15 MAC_ID_switches

The MAC_ID_switches value shall be gettable and not settable on all devices. The returned value shall be 0xFF for auto-address nodes and nodes without address switches.

#### 7.2.3.16 MAC_ID_changed

The MAC_ID_changed value shall be cleared when the device is reset and set whenever the device detects that the address switches have been changed. Once set, it stays set until the device is reset, even if the address switches are returned to their original settings. The frequency at which the address switches are checked shall be device dependent.

When a change in the network address switches is detected, this flag shall be set and a minor error shall be reported to the host.

#### 7.2.3.17 sched_max_frame

The sched_max_frame attribute shall be used to limit the maximum size of the node's scheduled transmissions. A value of 0 shall indicate that a node shall not transmit during the scheduled portion of the NUT.

#### 7.2.3.18 error_log

The instance attribute 0x86 shall be a copy of the error_log sub-attribute contained in instance attribute 0x82.

NOTE The information in this attribute is a copy of the information found in the diagnostic_counters attribute. This copy is only gettable. The values in this attribute can be cleared only by using the Get_and_Clear service request on the diagnostic_counters attribute. See 7.2.5 for more details.

This copy of the error_log shall only be accessible via the Get_Attribute services. This attribute shall only be cleared when the master copy in 0x82 is cleared.

#### 7.2.3.19 Active node table (instance attribute = 0x88)

Instance attribute 0x88 of the ControlNet object shall consist of an array of 256 bits, one per MAC ID. The least significant bit shall correspond to MAC ID = 0; the most significant bit shall correspond to MAC ID = 255. The bit for a specific MAC ID shall be set (1) for a node within 1 s of a node transmitting on the link. The bit shall be cleared (0) within 20 s of the node leaving the link. A node shall be considered to have left the link if it misses three consecutive transmit opportunities.

#### 7.2.3.20 New node table (instance attribute = 0x89)

Instance attribute 0x89 of the ControlNet object shall consist of an array of 256 bits, one per MAC ID. The least significant bit shall correspond to MAC ID = 0; the most significant bit shall correspond to MAC ID = 255. The bit for a specific MAC ID shall be set (1) for a node within 1 s of a node transmitting on the link. The bit shall be cleared (0) between 10 s and 20 s after the node has joined the link.

### 7.2.4 Common services

#### 7.2.4.1 General

The ControlNet object shall support the common services as specified in Table 19.

**Table 19 – ControlNet object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| **0x05** | N/A | Required | Reset | Return ControlNet object to power up state |
| **0x10** | N/A | Conditional | Set_Attribute_Single | Set network or node specific configuration information |
| **0x0E** | Required | Required | Get_Attribute_Single | Get network or node specific configuration information |
| **0x01** | N/A | Optional | Get_Attribute_All | Get network and node specific configuration information |
| **0x02** | N/A | Optional | Set_Attribute_All | Set network and node specific configuration information |
| **0x03** | N/A | Optional | Get_Attribute_List | Get network and/or node specific configuration information |
| **0x04** | N/A | Optional | Set_Attribute_List | Set network and/or node specific configuration information |

#### 7.2.4.2 Reset service

The Reset service shall reinitialize the Network Attachment Monitor (NAM) causing it to monitor the link activity and then attempt to join the link. MAC ID switches shall not be re-evaluated due to the reset.

The ControlNet object need not respond to a Reset request before actually performing the reset operation.

#### 7.2.4.3 Set_Attribute_Single

If any attribute has been implemented as settable, the Set_Attribute_Single service shall be required.

#### 7.2.4.4 Get_Attribute_All response

The object/service specific reply data portion for a successful Get_Attribute response is shown in the Instance Attributes for the ControlNet object table. The size of this response is dependent on the attribute instance(s) being accessed.

Because of the large size of the Get_Attribute_All response, devices with limited resources need not support this service.

The data portion of a Get_Attribute_All service request shall contain the following attributes in the following order:

0x81 - current_net_config

0x82 - diagnostic_counters

0x83 - station_status

0x84 - MAC_ID

0x86 - error_log

### 7.2.5 Class specific services

#### 7.2.5.1 General

The ControlNet object shall support the class specific services as specified in Table 20.

**Table 20 – ControlNet object class specific services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| **0x4C** | N/A | Required | Get_and_Clear | Get then clear the diagnostic counters |
| **0x4D** | N/A | Required | Enter_Listen_Only | Force the node to enter and stay in listen only mode until reset |
| **0x4E** | N/A | Required (See 7.2.5.4) | Where_am_I | Determine the MAC ID of the device this node is attached via the NAP |
| **0x4F** | N/A | Conditional | Auto_Address | Required in all auto-address nodes. Select a new MAC ID using the auto address sequence of the NAM |

#### 7.2.5.2 Get_and_Clear service

This class specific service shall be evoke an identical response to the Get_Attribute_Single service; however, after the response Lpacket is built, the value of the attribute shall be set to zero. Only instance attributes 0x82 and 0x87 shall support this service.

#### 7.2.5.3 Enter_Listen_Only service

The Enter_Listen_Only service shall cause this object to send a `DLL_listen_only_request(FALSE)` to the DLL associated with this instance of the ControlNet object. This service need not generate a response since the node may stop transmitting before the response is sent.

The node should remain in the listen-only state until a reset request is sent to either the ControlNet or Identity object.

While devices are in the listen-only state due to the Enter_Listen_Only service, they are not required to apply the Network Redundancy bits of the TUI (see Table 26), and redundant media devices may enable both channels.

NOTE The Enter_Listen_Only service is intended to be used for diagnostic purposes to force nodes to drop off the link until they are specifically requested to rejoin. This gives the possibility for a node to effectively be removed from the link without having to physically disconnect the network cable.

#### 7.2.5.4 Where_am_I service

All transient nodes shall implement the Where_am_I service. This service shall determine the MAC ID of the node to which the transient node is connected via its network access port (NAP). The service may be implemented by using the WAMI server functionality of the Station Management entity.

NOTE A transient node can use this service to identify the MAC ID of the node into whose NAP it is plugged.

The Where_am_I response shall be a single USINT containing the MAC ID of the permanent node to which the transient node is connected. If the ControlNet object is unable to complete the WAMI query, it shall return a MAC ID of 255.

#### 7.2.5.5 Auto_Address service

For nodes that support automatically selecting a MAC ID, this service shall reinitialize the NAM causing it to monitor the link activity and then attempt to join the link possibly with a new MAC ID. Nodes that do not support automatically selecting a MAC ID shall not implement this service.

The Auto_Address service need not respond prior searching for a new MAC_ID.

### 7.2.6 Behavior

#### 7.2.6.1 Effect of `DLL_tminus_zero_indication`

Upon receipt of this indication, the object shall copy the contents of instance attribute 0x80 (pending_link_config) into instance attribute 0x81 (current_link_config). An internal copy of instance attribute 0x80 shall be kept even if the implementation has not made it accessible via one of the Get_Attribute services.

#### 7.2.6.2 Duplicate node

If a `DLL_event_indicaton(DLL_EV_dupNode)` is received from the Data Link Layer, the ControlNet object shall force the DLL into a listen-only state. The ControlNet object shall latch this state until a reset service is received. A reset to the Identity object shall also end the listen only state.

#### 7.2.6.3 Redundancy

The network redundancy setting for a node shall be initially set at start up to the best capability of the device. Best capability means Channel A enabled for single channel devices, both channels enabled for network redundant devices). The power up redundancy setting shall be reflected in the TUI contained in the instance attribute 0x82.

The network redundancy setting for a node shall be updated to the current network redundancy setting by the Keeper in one of two ways:

– The node receives a TUI Lpacket from the wire with the master Keeper and configured Keeper bits set and the net change in progress bit reset in the Status_Flags word;

– The node receives a fixed tag 0x81 Lpacket.

In either case, the received TUI data shall be copied into attribute 0x80.

If a single channel node joins a redundant link, it shall behave as if it were a redundant node with a faulty channel.

### 7.2.7 Module status indicator

The module status indicator, if implemented, shall provide the following status indications:

a) solid green when connections are active in the *on line* state;
b) flashing green/off in the *check for moderator listen only* state;
c) flashing green/off when rogue conditions are detected;
d) flashing red/off for a recoverable error;
e) flashing red/off when in the DUP_LISTEN_ONLY state;
f) solid red for a irrecoverable error.

## 7.3 Keeper object

### 7.3.1 Overview

The Keeper object shall hold the link attributes for all devices on a link and shall be responsible for distributing those attributes to those devices in an orderly fashion. It also holds (for link scheduling software) a copy of the connection originator data for all connection originator devices using a network. If there are multiple Keeper objects on a link, they perform negotiations to determine which Keeper is the master Keeper and which Keeper(s) perform backup Keeper responsibilities. The master Keeper is the Keeper actively distributing attributes to the nodes on the network. A backup Keeper is one that monitors Keeper related network activity and can transition into the role of master Keeper should the master Keeper fail to perform.

The Keeper object also contains support for obtaining, holding, and releasing the Network Resource, a network semaphore that is used to eliminate conflicts when modifying the attribute data held by the Keeper object(s) on a link.

### 7.3.2 Revision history

The revision history of the Keeper object is described in Table 21.

**Table 21 – Keeper object revision history**

| Class revision | Description of changes |
|---|---|
| **01** | Initial Release |
| **02** | Release for IEC 61158 series |

### 7.3.3 Class attributes

The Keeper object shall support the class attributes as specified in Table 22.

**Table 22 – Keeper object class attributes**

| Number | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| **0x1** | Required | Get | Revision | UINT | Revision of this object | 2 (revision 2) (See Note 2) |

NOTE 1 Revision 1 Keeper devices are restricted to only operate at MAC ID = 1. These devices perform the functions of the Keeper as described in this specification as long as no other Keeper is present on the Network. At any other MAC ID, these devices containing revision 1 Keepers perform as if no Keeper objects were present.

NOTE 2 Revision 2 Keepers devices operate at any legal MAC ID value. These devices perform the functions of the Keeper as described in this specification in the presence of any other Revision 2 Keepers on the Network.

### 7.3.4 Instance attributes

#### 7.3.4.1 General

The Keeper object shall support the instance attributes as specified in Table 23. Only one instance of the Keeper object is allowed. That instance shall be sensitive to the port on which requests are received. This is especially important in the case of a router device where one Keeper instance receives requests from two links.

**Table 23 – Keeper object instance attributes**

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| 0x00 | Required | Get | status | STRUCT of 2 octets | Keeper object status | |
| | | | state | USINT | current Keeper operating state | |
| | | | reserved | USINT | reserved for data alignment | |
| 0x01 through 0x63 | Required | Get/Set | port_status | STRUCT of 10 octets | port status for node 1 through node 99 | |
| | | | port_status | UINT | port status | |
| | | | ID | STRUCT of 8 octets | node type identification | |
| | | | vendor | UINT | vendor code | |
| | | | type | UINT | product type | |
| | | | code | UINT | product code | |
| | | | major | USINT | major revision | |
| | | | minor | USINT | minor revision | |
| 0x64 – 0xFE | Reserved | | | | | Reserved for future expansion |
| 0xFF | Required | Get/Set | net_config | STRUCT of 12 octets | current network parameters | |
| | | | NUT | UINT | Network Update Time | In 10 µs ticks |
| | | | smax | USINT | Scheduled max node ID | 0 to 99 |
| | | | umax | USINT | Unscheduled max node ID | 0 to 99 |
| | | | Slot_Time | USINT | Slot Time | In 1 µs ticks |
| | | | Blank_Time | USINT | Blanking Time | In octets |
| | | | Gb_Start | USINT | Guard Band Start | In 10 µs ticks |
| | | | Gb_Center | USINT | Guard Band Center | In 10 µs ticks |
| | | | Reserved | UINT | Reserved for Data Alignment | |

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| | | | Int_Cnt mod | USINT | Interval Count Modulus (Macrocycle length) | 127 |
| | | | Gb_Prestart | USINT | Guard Band Prestart | In 10 μs ticks |
| 0x100 | Required | Get/Set | name | UINT [33] | current name of this link | 32 unicode characters and a unicode NULL |
| 0x101 | Required | Get/Set | RT_TUI | STRUCT of 22 octets | Table Unique Identifier | |
| | | | unique_ID | UDINT | attribute CRC | |
| | | | status_flag | UINT | TUI flag bits | See 7.2.3.3 |
| | | | reserved | USINT [16] | reserved to allow common format with attribute 0x102 | |
| 0x102 | Required | Get | Link_TUI | STRUCT of 22 octets | Table Unique Identifier (current link only) | |
| | | | unique_ID | UDINT | attribute CRC | |
| | | | status_flag | UINT | TUI flag bits | See 7.2.3.3 |
| | | | Keeper_MAC_ID | USINT | MAC ID of node broadcasting the TUI | Any legal MAC ID |
| | | | reserved | USINT | reserved for data alignment | |
| | | | Net_Resource_ Vendor_Id | UINT | Vendor ID of object holding Net Resource | |
| | | | Net_Resource_ Serial_Number | UDINT | Serial number of object holding Net Resource | |
| | | | Net_Resource_ Class | UDINT | Class of object holding Net Resource | |
| | | | Net_Resource_ Instance | UDINT | Instance of object holding Net Resource | |
| 0x103 | Required | Get/Set | Cable_Config | STRUCT of up to 100 octets | Current cable configuration | |
| | | | Id | USINT | Id value | Always 0xFF |
| | | | Num_Elements | USINT | Number of cable configuration elements in network configuration | Range 1 to 24 |
| | | | Propagation_time | UINT | Number of 100 ns ticks | |
| | | | Physical element | Phy element [24] | | |
| | | | Phy_element | STRUCT of 4 octets | | |
| | | | Vendor_id | UINT | Vendor code | |
| | | | Product_code | USINT | Product code | |
| | | | How_many | USINT | | |
| 0x104 | Required | Get/Set | CO_summary | STRUCT of 204 octets | General information about the *co_data* attribute | |
| | | | data_size | UINT | Size of *co_data* attribute | In words |

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| | | | connection_info_ revision | USINT | | Used by programming tool<br>0 = Format 1<br>1 = Format 2<br>2 - 255 reserved |
| | | | tool_keeper_ revision | USINT | Highest level co_data parse rule supported | Level supported is ≤ Keeper object revision<br>0 = Format 1<br>1 - 255 reserved<br>See 7.3.6.7 |
| | | | 0ffsets | UINT[100] | Array of word offsets into the *co_data* attribute, by MAC ID | In words<br>0xFFFF = no data for this MAC ID<br>node 0 based index |
| 0x105 | Required | Get/Set fragment | CO_data | STRUCT of up to 14 988 octets | connection originator information | Tool Keeper Revision Format 1 |
| | | | Branch | STRUCT of variable size | Information relative to all routers or connection originators on this link | |
| | | | num_devices | UINT | number of devices on this link | |
| | | | device | STRUCT of variable size | details of this device | |
| | | | type | USINT | types of device | 1 = router<br>2 = connection originator |
| | | | Path_Size | USINT | size of path to node | In 16 bit words |
| | | | Path | ARRAY of UINT | path to device | |
| | | | CO_data | STRUCT of variable size | Details of CO data | Present only for CO devices (type = 2) |
| | | | COP | UDINT | CO password, as provided to the CO at the end of a scheduling session (see 7.4.5.8) | |
| | | | Size | UINT | Size of Connection data | In 16 bit words |
| | | | Connection | STRUCT of variable size | Connection data | Connection Info Revision in the format specified by connection_ info_revision of attribute 0x104<br>One per connection for this CO device. See 7.3.4.7 |

#### 7.3.4.2 Operating state definitions

The values for the Keeper operating state shall be as defined in Table 24.

**Table 24 – Keeper operating state definitions**

| Value | Description |
|---|---|
| 0 | Power up - Not on line |
| 1 | Power up - TUI wait |
| 2 | Power up - TUI poll |
| 3 | Backup |
| 4 | Master verify |
| 5 | Master |
| 6 | Faulted Backup |
| 7 | Faulted Master verify |
| 8 | Faulted Master |
| 9 | Net Change - Backup |
| 10 | Net Change - Master |
| 11 | Net Change - Faulted Backup |
| 12 | Net Change - Faulted Master |

#### 7.3.4.3 Port status flag bit definitions

The values for the Port status flag shall be as defined in Table 25.

**Table 25 – Port status flag bit definitions**

| Bit | Meaning |
|---|---|
| 0 | Node accepts scheduled connections bit<br>0 = Node does not accept scheduled connections<br>1 = Node accepts scheduled connections |
| 1 – 15 | reserved (set to 0) |

#### 7.3.4.4 status_flag

The Keeper configured bit cleared in the TUI status word shall identify a Keeper with an invalid configuration or cleared memory.

Referring to Table 26, the status_flag field of the TUI sub–attribute shall consist of a 16 bit value. Unused flag bits shall be reserved and set to 0.

The holding network resource bit shall only be used by the Keeper object.

The Master Keeper bit shall indicate whether or not this node has ever heard from a master Keeper. The ControlNet object shall clear this bit at initialization. All TUIs sent by the master Keeper shall have this bit set. If a TUI Lpacket is received with the Master Keeper bit not set, the Lpacket shall be retained as a received TUI, but the redundancy information shall not be used.

The redundancy bits shall be used to override the default redundancy settings for the node. They shall indicate if the link is being operated in the single channel mode or redundant. The

redundancy information shall not be used if a TUI Lpacket is received with the Master Keeper bit clear.

The net change in progress bit shall indicate that a synchronized network change sequence is in progress and that new nodes shall delay going on-line until the change sequence is completed and the bit is cleared. This bit shall be cleared in the case of a network resource timeout situation or if the network resource is released.

**Table 26 – TUI status flag bits**

| Bit | Meaning | Used in: | | |
|---|---|---|---|---|
| | | RT_TUI | Network_TUI | ControlNet object TUI |
| 0 - 1 | Network Redundancy<br>Bit 1 = 0, bit 0 = 0; Illegal combination<br>Bit 1 = 0, bit 0 = 1; Channel B only<br>Bit 1 = 1, bit 0 = 0; Channel A only<br>Bit 1 = 1, bit 0 = 1; Redundant | X | X | X |
| 2 | Holding Network Resource bit<br>0 = Network Resource not being held<br>1 = Network Resource being held for the object identified in the TUI | | X | |
| 3 | Network change in progress bit<br>0 = No network change in progress<br>1 = Network change in progress | X | X | X |
| 4 | Keeper State bit<br>0 = TUI transmitted by Keeper not in MASTER state<br>1 = TUI transmitted by Keeper in MASTER state. | | X | X |
| 5 | Keeper configured bit<br>0 = Keeper attributes not configured (RT_TUI)<br>1 = Keeper attributes configured (RT_TUI) | X | X | X |
| 6 - 15 | reserved ( set to 0 ) | | | |

#### 7.3.4.5 unique_ID

The unique_ID field of the TUI attribute shall consist of a 32-bit value calculated from the contents of the Keeper attribute table (excluding the TUI attribute) by the software configuration tool. The following attributes shall be used, in the given order to calculate the TUI unique_ID:

0x1 through 0x63 (port parameters for nodes 1 through 99)

0xFF (link parameters

0x100 (link name);

0x101 (RT TUI) redundancy bits only;

0x103 (cable configuration);

0x105 (CO device password data only);

The TUI unique identifier shall be calculated as follows:

– the TUI unique identifier shall be initialized to 0;

– the data for attributes 0x1 through 0x63, attribute 0xFF, attribute 0x100, redundancy setting (attribute 0x0101 RT_TUI status flag bits 0–1), 0x103 and each CO password (in address order) shall be run through the CRC polynomial.

The polynomial used to calculate the CRC shall be

$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + 1$

as shown in Figure 17.

```
UDINT CRC(                    // (r) the updated CRC
      unsigned long CRC,  // (i) the CRC to start with
      SWORD* buffer,          // (i) pointer to buffer of SWORD / octets
      WORD size)              // (i) the number of octets in the buffer

#  define KEEPER_CRC_POLY 0xEDB88320L
   WORD octetIndex;
   WORD bitIndex;
   SWORD oneOctet;

// loop through the octets passed, including them in the CRC
   for (octetIndex=0; octetIndex < size; octetIndex++)
   {
      oneOctet = buffer[octetIndex];

// loop through the bits passed, including them in the CRC
      for (bitIndex = 0; bitIndex < 8; bitIndex++)
      {
         if ((oneOctet & 0x1) ^ (CRC & 0x1))
         {
            CRC = (CRC >> 1) ^ KEEPER_CRC_POLY;
         } else {
            CRC = (CRC >> 1);
         }
         oneOctet >>= 1;
      }
   }

// return the new CRC
   return(CRC);
}
```

**Figure 17 – Keeper CRC algorithm**

#### 7.3.4.6 Keeper MAC_ID

The Keeper_MAC_ID should be the MAC ID of the current master Keeper that is broadcasting the TUI. If no TUI has been received in the last 12 NUTS, a value of 0xFF shall be returned.

#### 7.3.4.7 Connection

Various formats exist for connection data. Each format is described below. The entire collection of connection data entries shall be an integral number of words.

Format 1 does not specify the size of the path; therefore, its path[ ] shall be one of two forms:

– class segment, instance segment, then two connection point segments; or

– an ANSI extended symbol segment.

```
// Format 1
  USINT  O2T_Size     // in 16 bit words
  UINT   O2T_Rpi      // in 1 ms ticks from 2 to 32 767
  USINT  T2O_Size     // in 16 bit words
  UINT   T2O_RPI      // bit 15 = connection point, 0 = point to point, 1 = multipoint
                      // bit 14 = API in ms from 2 to 32 767
  USINT  O2T_Nui_Api  // The highest bit set indicates the API;
                      //   bit 7 = 128*NUT, bit 6 = 64*NUT, etc.
                      // Remaining bits indicate the O2T_NUI.  If API = 1, NUI = 0
  USINT  T_Node       // Range 1 – 99
  USINT  T2O_Nui_Api  // The highest bit set indicates the API;
                      //   bit 7 = 128*NUT, bit 6 = 64*NUT, etc.
                      // Remaining bits indicate the O2T_NUI.  If API = 1, NUI = 0
  UINT   path[]       // forward open connection path from this
```

```
// Format 2
  UINT   O2T_net_params  // copied from forward open
  UINT   T2O_net_params  // copied from forward open
  UINT   O2T_RPI         // RPI in floating point format
  UINT   T2O_RPI         // RPI in floating point format
  USINT  O2T_schedule    // from schedule segment in forward open
  USINT  MAC_ID
  USINT  T2O_schedule    // from schedule segment in forward open
  USINT  path_size       // in 16-bit words
  UINT   path[]          // forward open connection path from this
```

```
UDINT convert_to_10us_ticks (UINT RPI)
{
   return (RPI <= 0x4000) ? RPI:
          (RPI <= 0xE000) ? (RPI & 0x1FFF) + 0x2000 << (RPI>>13) - 1:
                          : (RPI & 0x0FFF) + 0x1000 << (RPI>>12 & 1) + 7;
}

UINT convert_to_Keeper_format (UDINT ticks)
{
   if (ticks < 0x00004000) return RPI;                        //       0 - 16383 by 1
   if (ticks < 0x00008000) return RPI>>1 & 0x1FFF | 0x4000; //   16384 - 32766 by 2
   if (ticks < 0x00010000) return RPI>>2 & 0x1FFF | 0x6000; //   32768 - 65532 by 4
   if (ticks < 0x00020000) return RPI>>3 & 0x1FFF | 0x8000; //   65536 - 131064 by 8
   if (ticks < 0x00040000) return RPI>>4 & 0x1FFF | 0xA000; //  131072 - 262128 by 16
   if (ticks < 0x00080000) return RPI>>5 & 0x1FFF | 0xC000; //  262144 - 524256 by 32
   if (ticks < 0x00100000) return RPI>>7 & 0x0FFF | 0xD000; //  524288 - 1048448 by 128
   if (ticks < 0x00200000) return RPI>>8 & 0x0FFF | 0xE000; // 1048576 - 2096896 by 256
   return 0xFFFF;
)
```

#### 7.3.4.8 Attribute data

A Keeper maintains the link and node configuration information for its link in an attribute data structure. The Keeper shall keep two copies of the attribute structure: pending copy in RAM and a current copy in non-volatile memory. When new attributes are set in the Keeper, they shall be set in the pending RAM copy. The RAM copy shall be written to the currently active copy in non-volatile storage only upon receipt of a special request to do so.

The attributes shall use the following rules for getting and setting:

– Set service requests set the pending copy in RAM;

– Get service requests get the current copy in non-volatile storage.

Keeper shall maintain the collection of attributes specified in Table 27:

**Table 27 – Keeper attributes**

| Attribute number | Keeper attributes |
|---|---|
| 0x01 - 0x63 | (port parameters for nodes 1 through 99) |
| 0xFF | (link parameters) |
| 0x100 | (link name) |
| 0x101 | (Keeper TUI) |
| 0x103 | (cable configuration) |
| 0x104 | (node offset information into CO data) |
| 0x105 | (CO path and password information) |

All Keeper attribute definitions assume a maximum of 99 nodes on the link. The Keeper object shall have memory to store the CO_summary and CO_data attributes.

Memory requirements (in octets) for the Keeper attributes are as specified in Table 28.

**Table 28 – Memory requirements (in octets) for the Keeper attributes**

| Octets | Keeper attributes |
|---|---|
| 2 | Keeper status |
| 990 | Port parameters (99 nodes at 10 octets each) |
| 12 | link parameters |
| 66 | link name |
| 22 | TUI |
| 100 | Cable configuration |
| 204 | CO summary |
| 14 988 | CO data (7 494 words) |
| 16 384 | **Total** (Exactly 16K octets) |

The sequence of events required to properly and safely update the Keeper attributes shall be as follows:

a) obtain the Network Resource;

b) issue a Change_Start service request to the Keeper;

c) issue the appropriate Set_Attribute service request(s) to update the pending copy of the Keeper attributes;

d) issue a Change_Complete service request to the Keeper. Doing this shall initiate a synchronized parameter change;

e) release the Network Resource.

### 7.3.5 Common services

The Keeper common services shall all operate with an internal time out to prevent a hardware error condition from indefinitely stalling operations. The actual time out values shall be device dependent.

The Keeper object shall support the common services as specified in Table 29.

**Table 29 – Keeper object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x03 | N/A | Required | Get_Attribute_List | Get network and node specific configuration information |
| 0x04 | N/A | Required | Set_Attribute_List | Set network and node specific configuration information |
| 0x0E | Required | Required | Get_Attribute_Single | Get network and node specific configuration information |
| 0x10 | N/A | Required | Set_Attribute_Single | Set network and node specific configuration information |

The Keeper object is capable of receiving these service requests via fixed tags 83 and 88. When received via fixed tag 83, the request is processed and responded to as required independent of Keeper state. When received on fixed tag 88, the Keeper will only respond to the request if it is in the Master, Net Change Master, Faulted Master or Net Change Faulted Master states. Keeper nodes in other states shall not respond to the request.

### 7.3.6 Class specific services

#### 7.3.6.1 General

Any device that has implemented the Keeper object shall also implement the Keeper UCMM.

The Keeper object shall support the following class specific as specified in Table 30.

**Table 30 – Keeper object class specific services**

| Service code | Need in implementation | | Service name | Description of service | Parameter(s) |
|---|---|---|---|---|---|
| | Class | Instance | | | |
| 0x4B | N/A | Required | Obtain_ Network_Resource (ONR) | Attempt to obtain the Network Resource for this link.<br>If successful, hold for 60 s | UINT vendor ID<br>UDINT serial number<br>UDINT class<br>UDINT instance |
| 0x4C | N/A | Required | Hold_ Network_Resource (HNR) | Continue to hold the previously obtained Network Resource for this link for 60 s | UINT vendor ID<br>UDINT serial number<br>UDINT class<br>UDINT instance |
| 0x4D | N/A | Required | Release_ Network_Resource (RNR) | Release the Network Resource for this link | UINT vendor ID<br>UDINT serial number<br>UDINT class<br>UDINT instance |
| 0x4E | N/A | Required | Change_Start (CS) | Copy current attributes in non-volatile storage to pending attributes in RAM.<br>Enter the appropriate *net change* operating state | <none> |

| Service code | Need in implementation | | Service name | Description of service | Parameter(s) |
|---|---|---|---|---|---|
| | Class | Instance | | | |
| 0x4F | N/A | Required | Change_Complete (CC) | Copy pending attribute data in RAM to current data in non-volatile storage.<br>Restart normal Keeper operation.<br>Initiate synchronized network change if net attributes changed.<br>If number of sets does not match, do *change_abort* processing | 16 bit number of “sets” performed since *Change_Start* |
| 0x50 | N/A | Required | Change_Abort (CA) | Discard changes made to the pending attribute data, release the network resource and return to normal Keeper operation | <none> |
| 0x51 | N/A | Required | Get_Signature (GS) | Get signature value for a specific connection originator (CO password) | Request: path size (octet) (size is in words) path<br>Response: 32 bit cop |
| 0x52 | N/A | Required | Get_Attribute_ Fragment (GAF) | Get a portion of an attribute specifically designed to be fragmented, that is too large to fit in a single Lpacket.<br>For use only on the co_data attribute | UINT size; //words<br>UINT offset; //offset |
| 0x53 | N/A | Required | Set_Attribute_ Fragment (SAF) | Set a portion of an attribute specifically designed to be fragmented, that is too large to fit in a single Lpacket.<br>For use only on the CO_data attribute | UINT size; //words<br>UINT offset; //offset<br>UINT data [ ] |

The error codes for these services shall be as specified in Table 31, where the service codes are the abbreviations for the service names specified in Table 30.

**Table 31 – Service error codes**

| Error code | Meaning | Error code use by service | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | ONR | HNR | RNR | CS | CC | CA | GS | GAF | SAF |
| 0x02 | resource_unavailable_err | X | X | X | X | | | | | |
| 0x03 | invalid_parameter_err | | | | | X | | X | X | X |
| 0x04 | path_segment_err | | | | | | | X | X | X |
| 0x05 | path_dest_unknown_err | X | X | X | X | X | X | X | X | X |
| 0x08 | unimplemented_service_err | X | X | X | X | X | X | X | X | X |
| 0x09 | invalid_attr_err | | | | | | | | X | X |
| 0x0C | wrong_object_state_err | | | | | X | X | | | X |
| 0x0D | already_exists_err | X | | | X | | | | | |
| 0x0F | no_permission_err | | X | X | | | | | | |
| 0x11 | reply_too_large_err | | | | | | | X | | |
| 0x13 | not_enough_data_err | X | X | X | | X | | X | X | X |
| 0x14 | undefined_attr_err | | X | | | | | | | |
| 0x15 | too_much_data_err | | | | | | | X | | X |
| 0x16 | nonexistant_object_err | X | X | X | X | X | X | X | X | X |
| 0x26 | invalid_path_size_err | X | X | X | X | X | X | X | X | X |

#### 7.3.6.2 Network resource

The Network Resource (NR) shall be a link semaphore that is used to eliminate conflicts when modifying attribute information in Keeper objects.

In operation, the NR shall appear as a status flag bit and other data fields in the TUI Lpacket that shall be repetitively broadcast by the master Keeper.

A device that wishes to update the Keeper attribute information first shall submit a request to the master Keeper to obtain the NR on its behalf (via an Obtain_Network_Resource service). This request shall be broadcast since the identity of the master Keeper is not generally known. Once obtained, the device shall periodically ask (every 15 s) the master Keeper to continue holding the NR via a Hold_Network_Resource service request. The master Keeper shall maintain a 60 s timer that is started when the NR is initially obtained, and retriggered when a subsequent Hold_Network_Resource request from the same node is received. If the timer ever times out, the master Keeper shall automatically release the NR.

A backup Keeper shall respond to the Network Resource service requests as follows. When an *Obtain_Network_Resource* service request is received, it shall start a 60 s timer that shall be retriggered when a *Hold_Network_Resource* service request is received. The timer shall also be started when a Keeper powers up on an existing network and senses that the network resource is currently being held. The timer shall be stopped when a *Release_Network_Resource* service is received or the timer expires.

When a backup Keeper becomes the master Keeper, it shall use the data from the most recently received TUI Lpacket to hand off the holding of the NR from the old master Keeper.

In both master and backup Keepers, the 60 s timer shall be used to determine if the node requesting the NR has failed or has been prematurely removed from the network. When a Keeper's 60 s NR timer expires, the Keeper shall

a) abort any attribute changes in progress;

b) reset the net change in progress TUI status bit;

c) exit the net change operating state;

d) return to normal operation.

Since the NR can be held over a fairly long period if the user is performing substantial edits, the NR shall be cleanly handed off to the new master Keeper if the master Keeper changes. This handoff shall be invisible to the node that the master Keeper is holding the NR for. Since all information regarding the NR is contained in the TUI Lpacket being broadcast by the master Keeper, and since backup Keepers are receiving the broadcast TUI and are using the broadcast TUI for determining if and when they should take over as master Keeper, the backup Keepers shall have all the information needed to cleanly transition holding the NR when the appropriate backup Keeper takes over as master Keeper.

Faulted backup and master Keeper shall process Network Resource requests in the same way as non-faulted backup and master Keeper.

When the network resource is not being held, the related fields in the network TUI Lpackets shall be zeroed:

– NR VENDOR_ID;

– NR SERIAL_NUMBER;

– NR CLASS;

– NR_INSTANCE.

#### 7.3.6.3 Obtain_Network_Resource service

The Obtain_Network_Resource service request shall cause the master Keeper to hold the Network Resource for 60 s for itself or another node, if it is not already being held. The time period that the Network Resource is held may be extended indefinitely through use of the Hold_Network_Resource service request.

If the Network Resource is already being held for any node, an error status shall be returned.

The Obtain_Network_Resource service shall return the error codes as defined in Table 31, as designated in column ONR for this service.

#### 7.3.6.4 Hold_Network_Resource service

The Hold_Network_Resource service request shall cause the master Keeper to continue holding the Network Resource for the next 60 s if the node requesting the hold is the same node for which the master Keeper is currently holding the Network Resource.

If the Network Resource is being held for another node or is not being held, an error status shall be returned.

The Hold_Network_Resource service shall return the error codes as defined in Table 31, as designated in column HNR for this service.

#### 7.3.6.5 Release_Network_Resource service

The Release_Network_Resource service request shall cause the master Keeper to stop holding the Network Resource if the node requesting the release is the same node for which the master Keeper is currently holding the Network Resource. If a network change is in process and the network resource is released, the network change will be aborted.

Whether the Network Resource is being held for another node or is not being held, an error status shall be returned.

The Release_Network_Resource service shall return the error codes as described in Table 31, as designated in column RNR for this service.

#### 7.3.6.6 Change_Start service

The Change_Start service request shall cause the Keeper object to:

a) copy the current copy of the attributes in non-volatile storage into the pending copy in RAM;

b) clear the count of Set_Attribute service requests (including set single, list and fragment) received by the ControlNet object;

c) put the Keeper object into the appropriate net_change operating state.

This service request shall only be processed if the Network Resource is being held by the master Keeper as indicated by the net_resource bit in the broadcast TUI Lpacket.

The Change_Start service shall return the error codes as defined in Table 31, as designated in column CS for this service.

#### 7.3.6.7 Change_Complete service

The Change_Complete service request shall cause the Keeper object to:

a) copy its pending copy of the attributes from RAM to the current copy of the attributes in non-volatile storage;

b) perform a synchronized network change;

c) verify tool_keeper_revision in attribute 105 is ≤ Keeper object revision (if not true, mark attribute table invalid);

d) exit whatever network change operating state it is in;

e) return to normal Keeper operation.

If the Keeper object is not in one of the net change operating states, the Change_Complete service request shall be ignored and an incorrect state error response shall be generated.

If the number of Set_Attribute service requests (including: Set_Attribute_Single, Set_Attribute_List and Set_Attribute_Fragment) received by the Keeper object since the Change_Start service request does not match the number in the Change_complete parameter, a Change_Abort shall be executed instead and an error response shall be generated.

The reply to this service shall be returned after at least one TUI is broadcast with the net_change_in_progress bit cleared. A synchronized change shall take place prior to the TUI transmission with the bit reset.

The Change_Complete service shall return the error codes as defined in Table 31, as designated in column CC for this service.

#### 7.3.6.8 Change_Abort service

The Change_Abort service shall cause the Keeper object to

a) discard changes made to the pending attribute data;

b) release the network resource;

c) return to normal Keeper operation.

The reply to this service shall be returned after at least one TUI is broadcast with the net_change_in_progress bit and holding_network_resource bit cleared.

The Change_Abort service shall return the error codes as defined in Table 31, as designated in column CA for this service.

#### 7.3.6.9 Get_Signature service

The Get_Signature service shall cause the master Keeper to look up the CO password values for a connection originator in the CO_data attribute. The path to the device shall be given as the attribute in the request. This path shall be parsed and used to search the tree structure of the CO_data one branch at a time, to locate the appropriate COP (CO password) value, which is returned as a parameter in the response.

The path shall be parsed one branch at a time since the path entries in the CO_data attribute only specify the portion of the path between one branch and the next, not the entire path to that branch.

The Get_Signature service shall return the error codes as specified in Table 31, as designated in column GS for this service.

#### 7.3.6.10 Get_Attribute_Fragment service/response

The Get_Attribute_Fragment service shall collect and return some portion of the CO_data attribute data managed by the Keeper object. Any part of this attribute may be read. The

CO_data attribute shall be the only Keeper attribute which responds to fragment services. Accessing other attributes shall cause an error to be returned.

When getting attribute data, the current attribute data from non-volatile storage, not the pending copy of the attribute should be returned.

The Get_Attribute_Fragment service shall return the error codes as defined in Table 31, as designated in column GAF for this service.

#### 7.3.6.11 Set_Attribute_Fragment service

The Set_Attribute_Fragment service shall be used to set any portion of the CO_data attribute managed by the Keeper object. Only the CO_data attribute shall be accessed via this service. Accessing other attributes shall cause an error to be returned.

The Set_Attribute_Fragment service shall be processed only when the Keeper object is in one of the net change operating states. The Keeper object shall return an invalid mode error if it is not in one of the net change operating states when a Set_Attribute_Fragment service request is received.

Setting attribute data, shall set the pending copy of the attribute, not the current copy in non-volatile storage. The pending attributes shall be copied to the current attributes in non-volatile storage upon reception of a class specific Change_Complete service request.

The Keeper object shall not perform attribute value checking during any set operation.

The Set_Attribute_Fragment service shall return the error codes as defined in Table 31, as designated in column SAF for this service.

#### 7.3.6.12 Table unique identifier (broadcast), fixed tag 0x84

When the Keeper object is in the master verify, faulted_master_verify, master, faulted_master, net change master or net_change_faulted_master operating state, it shall attempt to transmit a special Table Unique Identifier (TUI) Lpacket as scheduled data once every 4 NUTs (that is, on NUT numbers 0, 4, 8, 12, 16 ...). All Keeper devices shall reserve 26 octets of scheduled link transmit time once every 4 NUTs for transmitting this Lpacket. If sent unscheduled, it shall be the highest QoS priority unscheduled information.

When generating the transmitted TUI Lpacket, any reserved fields shall use the values as specified in the corresponding reserved fields of attribute 0x0101. The programming tool shall set any reserved fields in attribute 0x0101, to zero.

The format of the TUI Lpacket link data shall be as defined in Table 32.

**Table 32 – Wire order format of the TUI Lpacket**

| Name | Data type | Description of parameter | Semantics of values |
|---|---|---|---|
| Size | USINT | Size of the Lpacket in words. | 0x0D |
| Control | USINT | Link layer Lpacket control octet. | 0x01 (Fixed tag Lpacket) |
| Fixed_Tag | USINT | Fixed tag value. | 0x84 (TUI Lpacket) |
| Destination_Mac_Id | USINT | Who receives the Lpacket. | 0xFF (Broadcast) |
| Unique_Id | UDINT | CRC of the Keeper's current attributes. | Least significant octet first. |
| Status_Flag | UINT | TUI flag values. | See the TUI attribute for details. |

| Name | Data type | Description of parameter | Semantics of values |
|---|---|---|---|
| Keeper_Mac_Id | USINT | MAC ID of the Keeper broadcasting the TUI. | See the TUI attribute for details. |
| Reserved | USINT | Reserved for data alignment | |
| Net_Resource_Vendor_Id | UINT | Vendor ID of object holding Net Resource for exclusive use | |
| Net_Resource_Serial_Number | UDINT | Serial number of object holding Net Resource for exclusive use | |
| Net_Resource_Class | UDINT | Class number of object holding Net Resource for exclusive use | |
| Net_Resource_Instance | UDINT | Instance number of object holding Net Resource for exclusive use | |

### 7.3.7 Service error codes

Table 33 lists possible error codes and the most likely condition under which the code would be returned.

**Table 33 – Service error codes**

| Error code | Meaning | Return condition |
|---|---|---|
| 0x02 | resource_unavailable_err | The requested network resource is not available |
| 0x03 | invalid_parameter_err | An invalid parameter was provided to the service |
| 0x04 | path_segment_err | Whenever an path over and above the attribute number is specified |
| 0x05 | path_dest_unknown_err | Object instance does not exist |
| 0x08 | unimplemented_service_err | Whenever the service is not supported for an attribute |
| 0x09 | invalid_attr_err | Whenever an attribute is specified which is not supported in this object |
| 0x0C | wrong_object_state_err | The object is in a state which does not allow the service to be performed |
| 0x0D | already_exists_err | The service has already been obtained by the requesting node |
| 0x0E | not_settable_err | Attribute is not settable |
| 0x0F | no_permission_err | Node requesting this service does not currently own the resource |
| 0x11 | reply_too_large_err | Not enough room in the response buffer to reply |
| 0x13 | not_enough_data_err | Whenever the request does not contain enough data |
| 0x14 | undefined_attr_err | Attempts to access an undefined attribute |
| 0x15 | too_much_data_err | More data has been encountered than expected for this service request |
| 0x16 | nonexistant_object_err | Keeper object not available |
| 0x26 | invalid_path_size_err | Whenever an invalid path segment for this service is specified |

### 7.3.8 Behavior

The numeric indication of Keeper state shall be as defined in Table 34.

**Table 34 – Keeper object operating states**

| State | Description |
|---|---|
| 0,1,2<br>– Power up | The Keeper object is either waiting for the node to come on-line or is determining if it is a legitimate Keeper for this link |
| 3<br>– Backup | The Keeper object has determined that it is a legitimate Keeper for this link but either has determined it is a backup Keeper or has not yet determined if it is the master Keeper for this link |
| 4<br>– Master Verify | The Keeper object has determined that it is a legitimate Keeper for this link but has not heard another active Keeper on the link or has received a good TUI from a matching Keeper at a higher node number or has heard a TUI from a faulted Keeper. It starts broadcasting the TUI Lpacket but does not process synchronized parameter changes or respond to requests of the master Keeper |

| State | Description |
|---|---|
| 5<br>– Master | The Keeper object has determined that it is a legitimate Keeper for this link and that it is the master Keeper for this link. It broadcasts the TUI Lpacket, processes synchronized link changes when required to change link parameters and shall respond to requests of the master Keeper |
| 6<br>– Faulted Backup | The Keeper object has determined that it is not a legitimate Keeper for this link, and is not the Keeper if there are only faulted Keepers on the link |
| 7<br>– Faulted Master Verify | The Keeper object is faulted but has not heard another active Keeper on the network for 12 NUT times. It starts broadcasting the TUI Lpacket but does not process synchronized link changes or respond to requests of the master Keeper |
| 8<br>– Faulted Master | The Keeper object is faulted and has assumed faulted master Keeper duties. There are no unfaulted Keeper's on the network. It broadcasts the TUI Lpacket, processes synchronized link changes when required to change network parameters and shall respond to requests of the master Keeper |
| 9,10,11,12<br>– Net Change | The Keeper is having its attributes updated. There is a Net *Change* substate for the *backup, master, fault* and *faulted master* states previously discussed. The separate substates are needed so the Keeper object can exit the *Net Change* state properly when updates complete normally and when updates are aborted. The pending attributes may be set only when the Keeper is in a *Net Change* state. Nodes in the *net change master* and *net change faulted master* shall respond to requests of the master Keeper. Nodes in the *net change backup* or *net change faulted backup* states shall not respond to requests of the master Keeper |

### 7.3.9 Miscellaneous notes

Keepers shall maintain parameters only for nodes on the same link.

Each Keeper shall be capable of distributing link parameters to all nodes on that link. The master Keeper shall be the Keeper with the lowest MAC ID of all the legitimate Keepers on the link. A faulted master Keeper on a link may or may not be the lowest MAC ID.

The Keeper shall be responsible for configuring those devices that appear in its attributes. The master Keeper shall be the only Keeper that issues responses to service requests via fixed tag 0x88 that are broadcast to all Keeper objects on a net.

All legitimate Keeper objects on a link shall have identical copies of the network attributes for all devices on the link. Keepers that do not agree with the master Keeper shall stay in the faulted backup state until they agree (agree means their TUI *unique_id* parameters are identical).

Only one programming terminal may modify Keeper attributes at a time. The programming terminal shall acquire the Network Resource prior to starting the Keeper modification and shall retain possession of the Network Resource during the entire modification procedure. If a programming terminal makes a change to a Keeper, it shall update all of the other Keepers on the link as well by broadcasting the Change_Start, Set_Attribute, and Change_Stop service requests.

The Keeper shall check the "tool Keeper revision" in attribute 0x0104 for a valid attribute table. If the Keeper is unable to understand that revision, the Keeper shall clear bit 5 in any TUI transmissions so that other more capable Keepers can assume the master Keeper responsibilities. Keeper revisions 0 and 1 shall be identical to revision 2.

The "connection info revision" value is independent on the Keeper revision, and shall only be utilized by programming tools.

The Keeper may, for network diagnostic purposes, broadcast to all nodes a debug fixed tag (0x90) Lpacket with three words of data. The first word identifies that the Keeper object is sending the diagnostic. The second word is the current "Keeper state" and the third word is the next "Keeper state".

### 7.3.10 Keeper power up sequence

#### 7.3.10.1 General

The Keeper power up sequence shall allow the Keeper object to begin operations under a variety of conditions, from a normal configured network power on state to one where many network devices are new and unconfigured.

The Keeper object power up sequence shall not begin until the NAM has brought the Keeper node on-line. The Keeper power up shall determine if this particular Keeper device is a legitimate Keeper for the local link and to bring the Keeper object into either the backup or faulted backup operating state.

If this device is a legitimate Keeper for the link, the Keeper object shall determine if it is the master Keeper or a backup Keeper for the link. If this device is not a legitimate Keeper for the link, the Keeper object enters a fault state. An unconfigured Keeper shall enter the fault state at power up.

The 12*NUT timer shall be retriggered whenever a TUI Lpacket is received that indicates the NR is being held. This shall indicate that the Keeper attributes are being updated.

The Keeper object's power up sequence is illustrated in Figure 18.

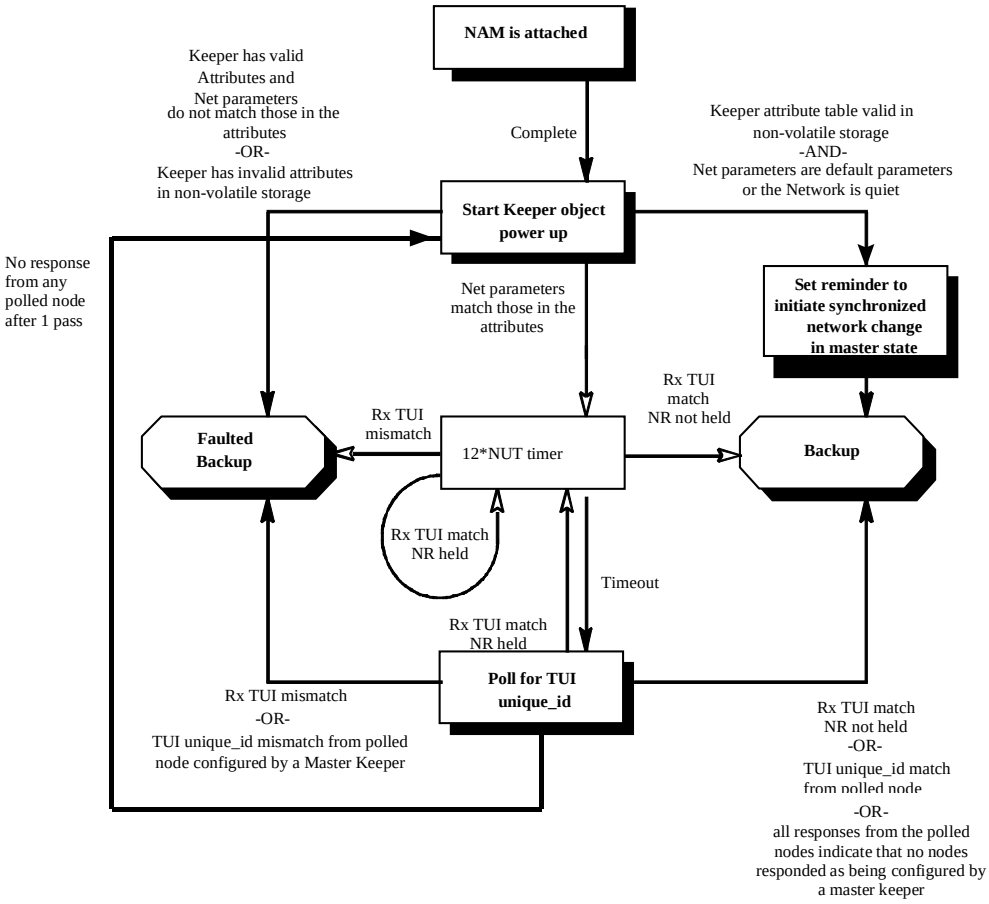



**Figure 18 – Keeper object power-up state diagram**

#### 7.3.10.2 Poll for TUI unique_id

The Poll for TUI unique_id state in Figure 18 shall require the following operation be performed by the Keeper object:

a) Issue a UCMM request to the addressed node's ControlNet object to read the object's current configuration parameters. Should the request fail, proceed on to the next MAC ID until UMAX is reached. Multiple concurrent requests may be permitted to reduce the poll time.

b) Each MAC ID in the range 1 through UMAX (excluding the MAC ID of the Keeper node) may be polled singly or concurrently with a Get_Attribute_Single request on the current_link_config attribute of the ControlNet object of the node. This attribute contains a copy of the current TUI attribute for the node. Concurrent access to multiple nodes at one instant will serve to accelerate the polling process.

c) If no response is heard from any polled node and UMAX has been reached, the poll is aborted and the Keeper object shall transition to the "Start Keeper object power up state", since at least one other node on a non-default network is present but unable to respond. This will allow the power up poll process to repeat until a node is capable of responding.

d) If a response is received from a polled node and the "Keeper State" bit in the returned TUI status_flag was set, the polled node was configured by a Keeper in the "Master" operating state. Compare the TUI unique_id value returned by the polled node to the one in this Keeper's attribute table. If it matches, the poll is aborted and the Keeper object shall transition to the "backup" operating state. If it mismatches, the poll is aborted and the Keeper object shall transition to the "faulted backup" operating state.

e) If a response is received from a polled node and the "Keeper State" bit in the returned TUI status_flag was not set, the polled node was not configured by a Keeper in the "Master" operating state. In this case, the information is discarded and the poll continued. If all nodes respond in this way, indicating that none of them were configured by a Keeper in the "Master" operating state, the poll is terminated and the Keeper shall transition to the "backup" operating state since in this situation, all nodes are either new or have lost their stored configuration.

#### 7.3.10.3 Operating states

The Keeper object operating states shall be defined as in Figure 19, Table 35, and in 7.3.10.3 to 7.3.10.9.

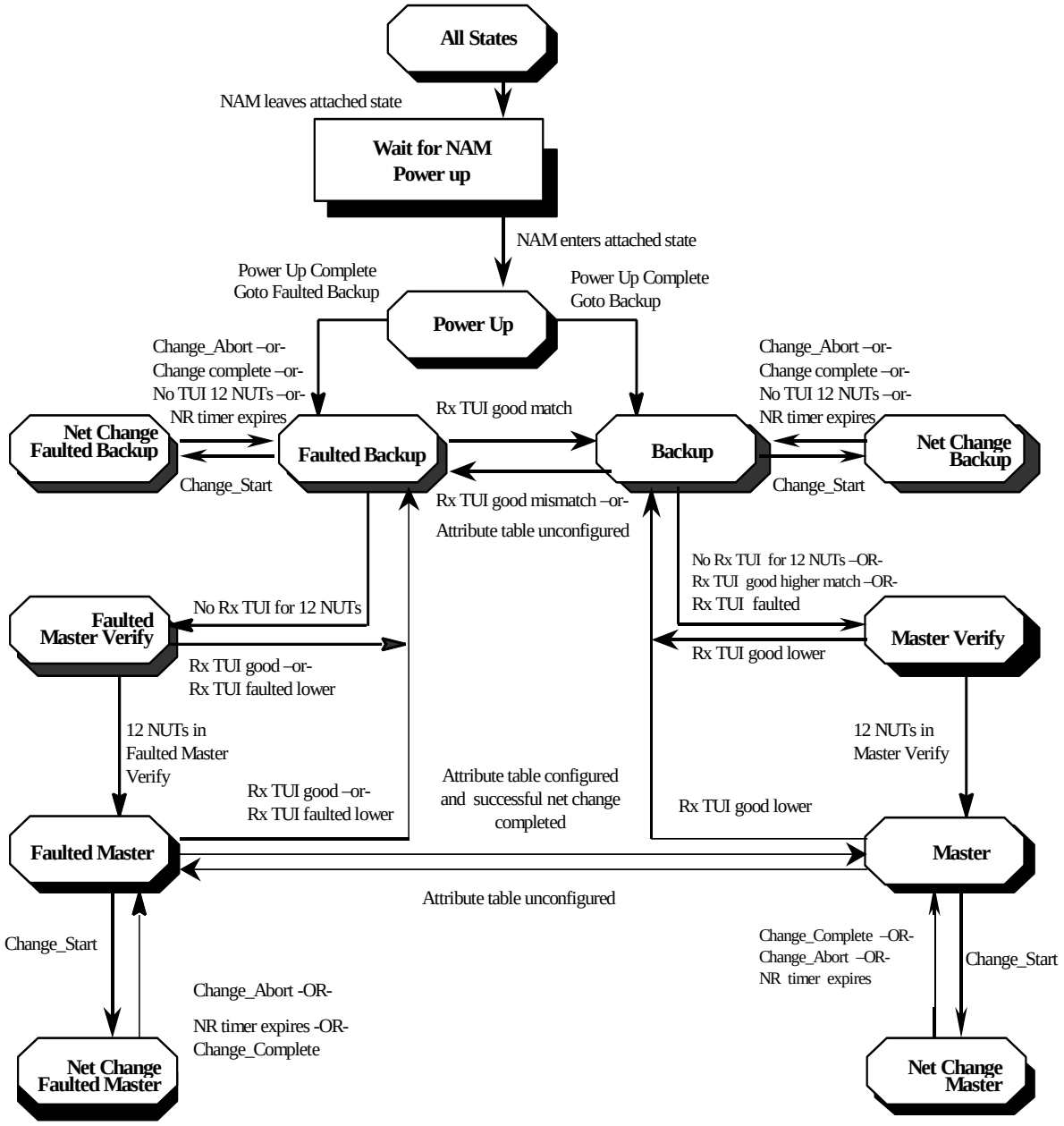



**Figure 19 – Keeper object operating state diagram**

**Table 35 – Keeper object state event matrix**

| Event | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Power Up | Backup | Master Verify | Master | Faulted Backup | Faulted Master Verify | Faulted Master | Net Change (Back-up) | Net Change (Master) | Net Change (Faulted Backup) | Net Change (Faulted Master) |
| Rx TUI from a good node | | | | | | Goto Faulted Backup | Goto Faulted Backup | | | | |
| Rx TUI from a good lower node | | | Goto Backup | | | | | | | | |
| Rx TUI from a good higher node (match) | | Goto Master verify | | | | | | | | | |

| Event | State | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Rx TUI from a good node (mismatch) | | Goto Faulted Backup | | | | | | | | | |
| Rx TUI from a good node (match) | | | | | Goto Backup | | | | | | |
| Rx TUI from a faulted node | | Goto Master Verify | | | | | | | | | |
| Rx TUI from a faulted lower node | | | | | | Goto Faulted Backup | | | | | |
| No TUI Rx for 12*NUT | | Goto Master Verify | | | Goto Faulted Master Verify | | | Abort change. Goto Backup | | Abort change. Goto Faulted Backup | |
| 12 Nut times elapse | | | Goto Master | | | Goto Faulted Master | | | | | |
| Change_Start | | Goto Net Change Backup | | Goto Net Change Master | Goto Net Change Faulted Backup | | Goto Net Change Faulted Master | | | | |
| Change_Complete | | | | | | | | Goto Backup | Goto Master | Goto Backup | Goto Master verify |
| Change_Abort | | | | | | | | Do Change Abort. Goto Backup | Do Change Abort. Goto Master | Do Change Abort. Goto Faulted Backup | Do Change Abort Goto Faulted Master |
| 60 s Network Resource timer expires | | | | | | | | Abort change. Goto Backup | Abort change. Goto Master | Abort change. Goto Faulted Backup | Abort change. Goto Faulted Master |
| NAM leaves attached state | Goto wait for NAM powerup | | | | | | | | | | |
| Power up complete to Backup | Goto Backup | | | | | | | | | | |
| Power up complete to Faulted Backup | Goto Faulted Backup | | | | | | | | | | |

NOTE Blank cells indicate that this event cannot occur, or if it does occur, no action shall be taken by the Keeper object.

#### 7.3.10.4 Network resource

Access to changing Keepers shall be controlled through the use of the Network Resource. Only one device can acquire the Network Resource for exclusive access at a time.

The Network Resource shall be held by the master Keeper for the node performing the attribute updates. If the master Keeper does not hear from the node performing the update at least once every 60 s, the master Keeper shall automatically release the Network Resource, abort the change in progress, and return to normal operation.

#### 7.3.10.5 Faulted Keeper

Any Keeper whose *TUI unique_id* attribute does not match that of the broadcast TUI from a Keeper shall enter a fault state unless a net change operation is in progress.

#### 7.3.10.6 MAC ID chooses master Keeper

The Keeper node MAC ID included in the TUI message shall be used by the Keeper to determine which Keeper on a link is the master Keeper.

#### 7.3.10.7 Master verify

The Keeper shall wait for 12*NUT after the first TUI broadcast begins before assuming master Keeper status. This shall be sufficient time for a Keeper with a lower MAC ID to become master Keeper. Other Keepers become backup Keepers.

#### 7.3.10.8 Rogue

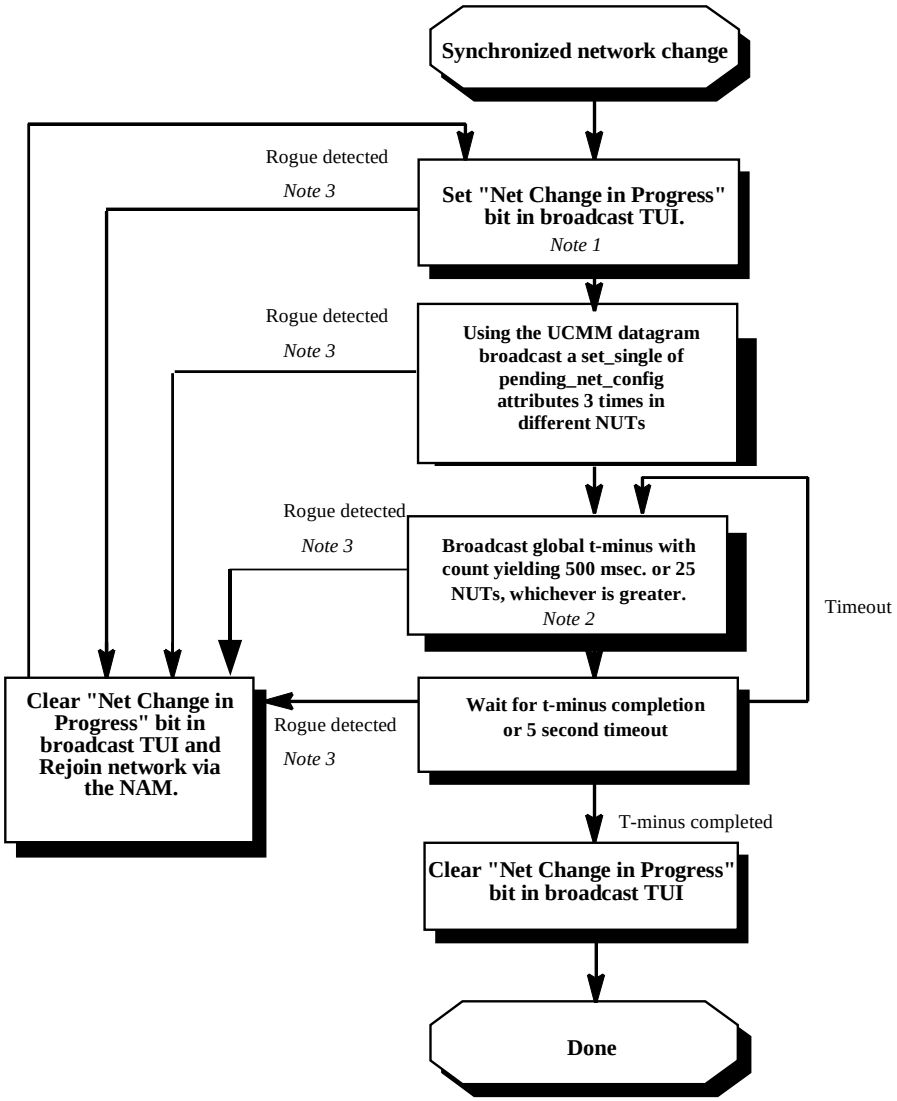
If the node containing the Keeper rogues during a synchronized network change sequence, the “Net Change in Progress” bit shall be cleared and control shall be returned back to the start of the synchronized network change processing immediately after the NAM brings the node back on-line. The normal Keeper object power-up sequence shall be bypassed.

#### 7.3.10.9 Synchronized network change processing algorithm

Although the moderator is used by the link to distribute operating parameters, it shall be the responsibility of the Keeper object to initiate changes to the link parameters.

All link parameter changes shall be accomplished through the use of a synchronized network change algorithm. This shall provide a means for all nodes on the network to change their current link parameters in unison without disrupting network operation.

The algorithm for the synchronized network change operation shall be as defined in Figure 20.




NOTE 1 Broadcasting a TUI with the net change in progress bit set forces nodes that are just powering up to hold off from going on-line until after the network change completes and the net change in progress bit in the broadcast TUI is cleared. This reduces the chances of a rogue event occurring.

NOTE 2 The tMinus delay count provides noise immunity and insures that all nodes will hear at least one moderator with the new tMinus count. It also provides to nodes a reasonable amount of time to process the Set_Single packets with new network attributes. The delay count is determined by taking the maximum of 25 or the result of dividing 500 ms by the current NUT time in ms.

NOTE 3 The rogue condition will occur only if the moderator node and at least one other node did not receive one of the three broadcast set network attribute service request packets. This would only occur if there was a serious noise event (lasting at least 3 NUTs in duration).

**Figure 20 – Synchronized network change processing**

## 7.4 Scheduling object

### 7.4.1 Overview

The Scheduling object shall exist in connection originator (CO) devices. The Scheduling object shall be used by link scheduling software (LSS) to

a) read scheduled connection information;

b) write schedule information;

c) provide the CO with a password (software key) which shall be used to authenticate the CO device's access to a specific link.

A LSS session with the Scheduling object schedules a particular link. If a CO device has connections on multiple links, multiple LSS sessions shall be needed to schedule all the connections. The results of the scheduling session shall be saved by the CO device (including the CO password computed by the LSS).

Since the LSS may be integrated with the programming software (PS) for a CO device, the Scheduling object shall provide commands which support integrated or a separate PS. Specifically, when the LSS writes Scheduling data to the Scheduling object, it may write conditionally or it may force the write. A conditional write of Scheduling data shall be used if the PS and LSS were not integrated and the Scheduling object shall only use the provided Scheduling data if a separate PS has not changed the connection information during the LSS session. A forced write may be used with an integrated LSS/PS if the integrated tool can guarantee that no other PS has changed connection information since it was last read during the current Scheduling session.

All communication to the Scheduling object shall use the Unconnected Message Manager (UCMM) fixed service, or an active Generic tag connection service to the Message Router object.

### 7.4.2 Class attributes

The Scheduling object shall support the class attributes as specified in Table 36. The Get_Attribute_All service shall return the class level attributes in numerical order, smallest to largest.

**Table 36 – Scheduling object class attributes**

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute |
|---|---|---|---|---|---|
| 0x01 | Required | Get | Revision | UINT | First revision, value = 1 |
| 0x02 | Required | Get | MaxInstances | UDINT | maximum number of Scheduling objects supported |
| 0x03 | Required | Get | NumInstances | UDINT | number of instantiated Scheduling objects |

### 7.4.3 Instance attributes

The Scheduling object shall support the instance attributes as specified in Table 37. The Get_Attribute_All service shall return the attribute level attributes in numerical order, smallest to largest.

**Table 37 – Scheduling object instance attributes**

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute |
|---|---|---|---|---|---|
| 0x01 | Required | Get/Set | Route | STRUCT of variable size | route from CO to the link being scheduled |
| | | | NumSegments | UINT | number of segments in path |
| | | | Segments | ARRAY of UINT | array of segments |
| 0x02 | Required | Get/Set | TimeOut | UDINT | watchdog time-out μs<br>(default = 60 s) |

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute |
|---|---|---|---|---|---|
| 0x03 | Optional | Get/Set | Controller State | UINT | bit 0 (run/program)<br>= 0, may be programmed<br>= 1, currently controlling I/O<br>bit 1 (remote/local)<br>= 0, bit 0 shall not be changed remotely<br>= 1, bit 0 may be changed remotely<br>bit 2-15 reserved and shall be zero |

### 7.4.4 Common services

#### 7.4.4.1 General

In addition to the services described in 7.4.4, the Scheduling object shall support the standard Get_Attribute_All service addressed to the class instance (instance zero). A specific Scheduling object implementation may optionally support other standard attribute services.

The Scheduling object shall support the common services as specified in Table 38.

**Table 38 – Scheduling object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x01 | Required | Optional | Get_Attribute_All | Returns a predefined listing of this objects attributes Modifies all settable attributes |
| 0x02 | N/A | Optional | Set_Attribute_All | Modifies all settable attributes |
| 0x03 | Optional | Optional | Get_Attribute_List | Returns the contents of a list of specified attributes |
| 0x04 | N/A | Optional | Set_Attribute_List | Modifies a list of settable attributes |
| 0x08 | Required | Optional | Create | Used to instantiate a Scheduling object |
| 0x09 | N/A | Required | Delete | Used to conclude a LSS Scheduling session |
| 0x0E | Optional | Optional | Get_Attribute_Single | Returns the contents of the specified attribute |
| 0x10 | N/A | Optional | Set_Attribute_Single | Modifies a single attribute |

An implementation may support setting the route instance-level attribute, but if it does, the internal behavior shall be indistinguishable from a delete service followed by a create service containing the new route.

#### 7.4.4.2 Create

The create service shall be used to instantiate a Scheduling object on the CO. Within the create message, the client (LSS) shall include a route pointing from the CO to the link to be scheduled. The route to the link shall consist of port segments and shall not contain network segments. The Scheduling object shall reply with an instance number to which all further communication shall be addressed for this Scheduling session, and with the currently stored CO password and path.

The new instance shall delete itself if communications to its client are severed. Communications to an instance shall be monitored via a watchdog timer whose time-out value (default = 60 s) can be set via one of the optional set attribute services. The watchdog timer

for an instance shall be reset when it receives any message. If the timer expires, the instance shall be deleted aborting all pending changes. The time-out value shall be a 32-bit integer with units of µs.

If the create service is addressed to the class level (instance zero), the instance number of the newly created instance shall be chosen automatically by the Scheduling object. For debug purposes, the Scheduling object may optionally allow the create service to be addressed to a specific instance number.

The Scheduling object shall support at least one instance. If the Scheduling object allows more than one instance to exist, it shall ensure the integrity of all its instances. The value of instance attribute 0x01 shall be provided with the create service.

```
class Scheduling_Create_Request: public MR_Request
{
   UINT   number_of_attributes;     // set to 1
   InstanceAttribute1 attribute1;
}

class Scheduling_Create_Response: public MR_Response
{
   UDINT  instance_number;
   UDINT  cop;             // Connection Originator Password (COP)
   UINT   path_size;          // in words, range 0 to 255
   UINT   path[];             // array of port segments from the scheduled link
                           //   to the connection originator
                           //   no key segments, no network segments
                           //   include the MAC ID of the link hop
};
```

The response shall include one of the status codes shown in Table 39.

**Table 39 – Status error descriptions for Create**

| General status | Extended status | Error description |
|---|---|---|
| 0x02 | 0x0001 | Insufficient resource — maximum number of Scheduling object instances already exist |
| | 0x0002 | Insufficient resource — not enough memory on CO device |
| 0x04 | N/A | Path syntax error |
| 0x08 | N/A | Unimplemented service (since non-zero instances are optional) |
| 0x10 | N/A | Cannot open another instance due to internal conflicts |
| 0x13 | N/A | Insufficient request data — request was too short or truncated |
| 0x0D | N/A | Object already exists (when non-zero instance specified) |
| 0x1C | N/A | Attribute list shortage — required attribute was missing |
| 0xDO | N/A | No scheduled connections on this link |

#### 7.4.4.3 Delete

The delete service shall be used to conclude a LSS Scheduling session. It deallocates all resources for a specified instance of the Scheduling object. Any pending changes which have not yet been committed shall be lost. Connections which have been broken by commands earlier in the LSS session shall not necessarily be restored. The delete service shall not be supported at the class level. The response of this service shall be as specified in Table 40.

**Table 40 – Status error descriptions for Delete and Kick_Timer**

| General status | Extended status | Error description |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Path syntax error |
| 0x05 | N/A | Instance undefined |
| 0x08 | N/A | Unimplemented service (if directed to instance 0) |

### 7.4.5 Class specific services

#### 7.4.5.1 General

The Scheduling object shall support the following class specific services as specified in Table 41.

**Table 41 – Scheduling object class specific services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x4B | N/A | Required | Kick_Timer | Reset the watchdog timer within the Scheduling object |
| 0x4C | N/A | Required | Read | Read information about the scheduled connections |
| 0x4D | N/A | Required | Conditional_Write | Write the schedule information back to the CO device |
| 0x4E | N/A | Optional | Forced Write | Write the schedule information back to the CO device |
| 0x4F | N/A | Required | Change Start | Initiate a Scheduling data change |
| 0x50 | N/A | Required | Break Connections | Break all affected connections after new Scheduling data |
| 0x51 | N/A | Required | Change Complete | Indicate that all scheduled connections have been broken |
| 0x52 | Required | N/A | Restart Connections | Break all connections which use a specific link |

#### 7.4.5.2 Kick_Timer

The kick_timer service shall reset the watchdog timer within an instance of the Scheduling object without otherwise affecting the state of the Scheduling object. The LSS shall issue a command at a rate of four per time-out period. For example, the LSS shall issue a service request every 15 s if the time-out value is set to the default 60 s. All other Scheduling object services, directed to this Scheduling instance, shall also reset the watchdog timer so the kick timer service need only be used if the client (LSS) wants to keep a Scheduling object instance alive but has nothing to say.

The watchdog timer for an instance shall be reset when it receives any message; however, it shall not start its countdown until the corresponding response is sent. This shall be to allow for a single task implementation. The response of this service shall be as specified in Table 40.

#### 7.4.5.3 Read

The read service shall be used by the LSS to read information about the scheduled connections that the CO device desires to create on a specific link. The connection information shall come from an application object within the CO device.

A read service request shall contain a required UDINT parameter that specifies which connection index on which to start the read. The Scheduling object shall reply to a read service with as much connection information as can be fit into a reply Lpacket. The connection indexes within an Lpacket shall be ordered from smallest to largest (gaps shall be allowed). The extended status field within the reply shall indicate whether connections with higher indexes exist in the CO device. A client (LSS) shall continue to submit read requests with higher starting indexes until it reads all the connection information for the chosen link. An extended status of 0 shall indicate that more connections exist in the device. An extended status of 1 shall indicate that all connections have been read.

The reply shall consist of a UINT indicating the number of connections described in this response followed by a series of entries each describing a connection that meets the connection request criteria (the route to the link). Each entry shall have a unique Scheduling data index that is chosen by and shall primarily be used by the CO. The last one of these can be incremented by the LSS to continue the request if it is too long for a reply.

Other parameters provided in the connection entry shall describe the connection sizes, connection types (multi-cast/point to point) and connection update rates (RPIs). The connection entry route information shall consist of the route from the link to the target module for the connection including the connection points within the target module. Port information need not actually be used by the LSS.

This route shall also contain the current values assigned for the connection in the network segments. The network segments shall contain both API (Lpacket interval) and starting NUT information.

```
class Scheduling_Read_Request: public MR_Request
{
   UDINT  first_connection_index;
}

class Scheduling_Read_Response: public MR_Response
{
   UINT   number_of_connections;
   Scheduling_Connection_Description desc[];
}

class Scheduling_Connection_Description
{
   UDINT  connection_index;
   UINT   O2T_parameters;
   UINT   T2O_parameters;
   UDINT  O2T_RPI;
   UDINT  T2O_RPI;
   UINT   path_size;      // in words, range 0 to 255
   UINT   path[];         // array of path segments
                     //   first segment is always O=>T schedule segment
                     //   second is always port segment onto the link
                     //   third is always T=>O schedule segment
                     //   remaining path including logical segments
                     //   and other port segments if multihop connection
};
```

The scheduling tool, LSS, shall use the logical segments of the target path to determine whether more than one connection request actually uses one multipoint producer. The rule for all objects except the Assembly object shall be that a match of class, instance, T⇒O connection point, and any finer granularity (for example attribute or member) constitutes a unique producer and it shall be scheduled once for both connections. Since the Assembly

object equates instance and connection point, a match on connection path instance shall not determine whether the producer is unique.

For example, these two connection paths specify the same producer (instance 0x88 of the Assembly object):

"20 04 24 03 2C 99 2C 88";

"20 04 24 AA 2C 77 2C 88".

These two connection paths specify different producers since the logical class segment do not match:

"20 77 24 03 2C 99 2C 88";

"20 77 24 AA 2C 99 2C 88".

The status response of this service shall be as specified in Table 42.

**Table 42 – Status error descriptions for Read**

| General status | Extended status | Error description |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Path syntax error |
| 0x05 | N/A | Instance undefined |
| 0x08 | N/A | Unimplemented service (if directed to instance 0) |
| 0x13 | N/A | Insufficient request data — request was too short or truncated |

#### 7.4.5.4 Conditional_Write

Conditional_Write service shall be used by the LSS to write the schedule information back to the CO device. Receipt of this message shall indicate that specific indexes of the connection information are being rescheduled by the LSS. A CO device may start breaking the current connections associated with the connection indexes, or it may wait until the receipt of a subsequent break connections service request.

If the new schedule provided by the LSS exactly matches the current schedule for a given connection, the CO device may ignore that part of the write request. A Scheduling object implementation may reply with an out of resource error if it receives updates for more connections than it has been queried about (via read requests). An implementation may also check for duplicate write requests; thereby, allowing retries — more write requests than read requests. The connection indexes may not be in order.

The difference between the conditional write service and the forced write service shall be that, for the conditional case, the CO device shall be required to verify the freshness of the connection information it previously sent to the LSS. If the data is not fresh, the CO device shall not use the Scheduling data for that connection index. In the forced case, the CO device need not check. “Fresh” connection information shall mean that nothing has changed the connection information since it was sent to the LSS via a read command.

The request shall be an array of connection schedules.

```
class Scheduling_Write_Request: public MR_Request
{
   UINT   number_of_connection_schedules;
   struct {
      UDINT  connection_index;
      USINT  O2T_schedule;
      USINT  T2O_schedule;
   } schedules[];
};
```

The response status of this service shall be as specified in Table 43.

**Table 43 – Status error descriptions for Conditional_Write**

| General status | Extended status | Error description |
|---|---|---|
| 0x00 | N/A | Success |
| 0x03 | N/A | Invalid value — bad index contained in request |
| 0x04 | N/A | Path syntax error |
| 0x05 | N/A | Instance undefined |
| 0x08 | N/A | Unimplemented service (if directed to instance 0) |
| 0x0C | N/A | Wrong state (***change start*** not received) |
| 0x10 | N/A | Device state conflict (device cannot break connection) |
| 0x13 | N/A | Insufficient request data — request was too short or truncated |
| 0x15 | N/A | Instance is unable to receive any more write requests |

#### 7.4.5.5 Forced_Write

Forced_Write service shall be similar to the conditional write service except that the LSS shall guarantee that it has provided valid connection information. The request parameters shall be the same as for the Conditional_Write service. The response status of this service shall be as specified in Table 44.

**Table 44 – Status error descriptions for Forced_Write**

| General status | Extended status | Error description |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Path syntax error |
| 0x05 | N/A | Instance undefined |
| 0x08 | N/A | Unimplemented service (if directed to instance 0) |
| 0x0C | N/A | Wrong state (***change start*** not received) |
| 0x10 | N/A | Device state conflict (device cannot break connection) |
| 0x13 | N/A | Insufficient request data — request was too short or truncated |
| 0x15 | N/A | Instance is unable to receive any more write requests |

#### 7.4.5.6 Change_Start

The Change_Start service shall be used to initiate a Scheduling data change in the controller. The controller shall allocate enough memory for the new data and for the CO password and path. The ***conditional write*** or ***forced write*** service shall be then used to write the data to the controller.

The CO password shall be used when a controller first is powered up and begins to remake scheduled connections. The CO shall ensure the path associated with the CO password (see Create response or Change_Complete request) matches the path from the network to the CO device. If the path matches the CO shall ensure this CO password matches the one stored in the Keeper object of the link prior to making any scheduled connections on that link.

The Change_Start request shall have only one parameter. The parameter shall be a UINT in the range 0 to 255 that specifies the size of the CO password and path to be provided in the Change_Complete service. Implementations may use this to allocate a receive buffer for the CO password and path.

The response status of this service shall be as specified in Table 45.

**Table 45 – Status error descriptions for Change_Start**

| General status | Extended status | Error description |
|---|---|---|
| 0x00 | N/A | Success |
| 0x02 | 0x0002 | Insufficient resource — not enough memory on CO device |
| 0x04 | N/A | Path syntax error |
| 0x05 | N/A | Instance undefined |
| 0x08 | N/A | Unimplemented service (if directed to instance 0) |
| 0x10 | N/A | Device state conflict (for example, device cannot allocate sufficient memory) |
| 0x13 | N/A | Insufficient request data — request was too short or truncated |

#### 7.4.5.7 Break_Connections

Break_Connections service shall be used after the new Scheduling data has been written to the controller to break all the affected connections. The service shall reply only after all the connections have been broken. If any connections are in the process of being connected, they shall also be closed before replying to this service.

The Scheduling object break connections service shall break all connections that were specified in the Scheduling data write services. An LSS may always write Scheduling data for all connections (causing all connections over the link to be broken) but another LSS may be optimized to minimize the number of connections that get broken.

When the Scheduling object break connections service is received, in addition to breaking the changed connections, the CO shall NULL out (zero out) the currently active scheduled data values for the connections it breaks to prevent connections from being re-established until the Scheduling data change complete service is received. If the change complete service is never received (due to loss of communications with LSS) these connections shall not be able to be opened until they are rescheduled. The response of this service shall be as specified in Table 46.

**Table 46 – Status error descriptions for Break_Connections**

| General status | Extended status | Error description |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Path syntax error |
| 0x05 | N/A | Instance undefined |
| 0x08 | N/A | Unimplemented service (if directed to instance 0) |

| General status | Extended status | Error description |
|---|---|---|
| 0x0C | N/A | Object state conflict |
| 0x10 | N/A | Device state conflict (device cannot break connections) |
| 0x13 | N/A | Insufficient request data — request was too short or truncated |

#### 7.4.5.8 Change_Complete

Change_Complete service shall be used to indicate that all scheduled connections have been broken and it is permitted to start making the new scheduled connections. The request contains the new CO password and path to be used for the link that has been scheduled. The CO shall ensure the path associated with the CO password (see Create response or Change_Complete request) matches the path from the network to the CO device. If the path matches the CO shall ensure this CO password matches the one stored in the Keeper object of the link prior to making any scheduled connections on that link.

```
class Scheduling_Change_Complete_Request: public MR_Request
{
   UDINT  cop;             // Connection Originator Password (COP)
   UINT   path_size;          // in words, range 0 to 255
   UINT   path[];             // array of port segments from the scheduled link
                           //   to the connection originator
                           //   no key segments, no network segments
                           //   include the MAC ID of the link hop
};
```

NOTE The reverse path included in this request gives the possibility for the LSS to detect changes in the network position of the scheduled CO during later scheduling sessions.

The response status of this service shall be as specified in Table 47.

**Table 47 – Status error descriptions for Change_Complete**

| General status | Extended status | Error description |
|---|---|---|
| 0x00 | N/A | Success |
| 0x03 | N/A | Invalid value (new CO password and path is not formatted correctly) |
| 0x04 | N/A | Path syntax error |
| 0x05 | N/A | Instance undefined |
| 0x08 | N/A | Unimplemented service (if directed to instance 0) |
| 0x10 | N/A | Device state conflict |
| 0x13 | N/A | Insufficient request data — request was too short or truncated |
| 0x0C | N/A | Wrong state |

#### 7.4.5.9 Restart_Connections

The Restart_Connections service may only be sent to the class instance of the Scheduling object (instance zero). It shall be used to break all connections which use a specific link. After all the connections have been broken, the CO device shall attempt to re-establish them. The request parameter for this service shall be structured as is instance attribute 0x01. The response status of this service shall be as specified in Table 48.

**Table 48 – Status error descriptions for Restart_Connections**

| General status | Extended status | Error description |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Path syntax error |
| 0x08 | N/A | Unimplemented service (if addressed to a non-zero instance) |
| 0x10 | N/A | Device state conflict (cannot break connections) |
| 0x13 | N/A | Insufficient request data — request was too short or truncated |
| 0xDO | N/A | No scheduled connections on this link |

### 7.4.6 Typical scheduling session

A typical scheduling session shall use these services:

a) create;

b) read;

c) write;

d) change_start;

e) break_connections;

f) conditional_write;

g) forced_write;

h) change_complete.

The following steps describe the usage of services in a typical Scheduling session:

1) The LSS shall begin by sending a ***create*** message to instance zero of the Scheduling object. Contained in this message shall be a path from the CO to the link. This path shall determine which connections are of interest to the LSS.

2) The Scheduling object shall create an instance of itself to service this LSS and reply with the instance number to which all future communications in this session shall be sent.

3) The newly created instance shall reset a 60 s timer upon receipt of any message. If this timer ever expires, the instance shall delete itself aborting any change in progress. In normal operation, the LSS shall be required to send a command to the Scheduling object at least once every 15 s allowing a few dropped messages without unnecessarily deleting an instance.

4) The LSS may optionally send ***read*** commands to the newly created instance to transfer the connection information from the CO device. The transmitted connection information shall contain indexes which are used to uniquely identify the connections internally within the CO device.

5) The LSS shall reflect back these indexes on subsequent ***write*** commands so that the CO may determine for which connections the ***write*** applies. The issuing of ***read*** commands shall be optional since the connection information can be obtained for the LSS by other means. For example, in an integrated LSS/PS, the PS shall provide this information to the LSS.

6) Once the connection information is acquired from each CO desiring scheduled data on the link, the LSS shall calculate a tentative schedule.

7) The LSS then shall issue the following sequence of commands to each of the CO devices: ***change start***, one or more ***writes***, and ***break connections***. This sequence can be done in parallel to each of the CO devices. The type of ***write*** command (forced or conditional) shall depend on how the connection information was obtained. If the LSS has independently ensured that the connection information is valid, it may use the ***forced write*** command; otherwise, it shall use a ***conditional write*** command. The LSS can know

that the connection information is valid by obtaining the edit resource for the CO before the connection information.

8) Upon receipt of a ***conditional write*** command, the Scheduling object shall check that the parameters of the connection have not changed since they were last read. If the connection has been changed, the Scheduling object shall not use the Scheduling information for the index whose connection parameters have changed. The Scheduling object within the CO can do this through any means it chooses.

9) The Scheduling object shall be absolved of responsibility to check the validity of ***forced writes***. This responsibility shall be assumed by the LSS.

10) Once the ***break connection*** commands to each of the CO devices has completed successfully, the LSS shall issue a ***change complete*** to each of the CO devices indicating that the schedule transmitted earlier via ***writes*** is now valid. The ***change complete*** command shall include the new CO password and path.

11) The session shall end by deleting the instance of the Scheduling object.

## 7.5 TCP/IP Interface object

### 7.5.1 Overview

The TCP/IP Interface object provides the mechanism to configure the TCP/IP interface of a device.

NOTE 1 Examples of configurable items include the device's IP Address, Network Mask, and Gateway Address.

The physical port associated with the TCP/IP Interface object shall be any port supporting the TCP/IP protocol.

NOTE 2 For example, a TCP/IP Interface object can be associated any of the following: an Ethernet ISO/IEC 8802-3 port, an ATM port, a serial port running SLIP, a serial port running PPP.

The TCP/IP Interface object provides an attribute that identifies the link-specific object for the associated physical port. The link-specific object is generally expected to provide link-specific counters as well as any link-specific configuration attributes.

Each device shall support exactly one instance of the TCP/IP Interface object for each TCP/IP-capable port on the device.

### 7.5.2 Revision history

Table 49 shows the revision history for the TCP/IP Interface object.

**Table 49 – Revision history**

| Revision | Reason for object definition update |
|---|---|
| 1 | Initial revision of this object definition |
| 2 | Added ACD instance attributes 10 (SelectACD) and 11 (LastConflictDetected)<br>Added instance attribute 12, QuickConnect<br>Added bits 5 (Interface Configuration Pending) and 6 (AcdStatus) to attribute 1, Status<br>Added bits 6 (Interface Configuration Change Requires Reset) and 7 (AcdCapable) to attribute 2, Configuration Capability |
| 3 | Added bit 7 (AcdFault) to attribute 1, Status |

### 7.5.3 Class attributes

The TCP/IP Interface object shall support the class attributes as specified in Table 50.

**Table 50 – TCP/IP Interface object class attributes**

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x01** | Required | Get | NV | Revision | UINT | Revision of this object | Third revision, value = 3 |
| **0x02** | Conditional[a] | Get | NV | Max instance | UINT | Maximum instance number of an object currently created in this class level of the device | The largest instance number of a created object at this class hierarchy level. |
| **0x03** | Conditional[a] | Get | NV | Number of instances | UINT | Number of object instances currently created at this class level of the device | The number of object instances at this class hierarchy level |
| **0X04 – 0x07** | These class attributes are optional | | | | | | |
| [a] Required if the number of instances is greater than 1. | | | | | | | |

### 7.5.4 Instance attributes

#### 7.5.4.1 General

The TCP/IP Interface object shall support the instance attributes as specified in Table 51.

**Table 51 – TCP/IP Interface object instance attributes**

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x01** | Required | Get | V | Status | DWORD | Interface status | See 7.5.4.2 |
| **0x02** | Required | Get | NV | Configuration capability | DWORD | Interface capability flags | Bit map of capability flags. See 7.5.4.3 |
| **0x03** | Required | Get<br>Set is conditional[a] | NV | Configuration control | DWORD | Interface control flags | Bit map of control flags. See 7.5.4.4 |
| **0x04** | Required | Get | NV | Physical link object | STRUCT of variable size | Path to physical link object | See 7.5.4.5 |
| | | | | Path size | UINT | Size of path | Number of UINTs in Path |
| | | | | Path | Padded EPATH | Logical segments identifying the physical link object | The path is restricted to one logical class segment and one logical instance segment. The maximum size is 12 octets. |
| **0x05** | Required | Get<br>Set is recommended | NV when configuration method is 0.<br>V when obtained via BOOTP or DHCP | Interface configuration | STRUCT of variable size | TCP/IP network interface configuration | See 7.5.4.6 |

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| | | | | IP address | UDINT | Device's IP address | Value of 0 indicates no IP address has been configured. Otherwise, the IP address shall be set to a valid class A, B, or C address and shall not be set to the loopback address (127.0.0.1) |
| | | | | Network mask | UDINT | Device's network mask | Value of 0 indicates no network mask address has been configured |
| | | | | Gateway address | UDINT | gateway address | Value of 0 indicates no IP address has been configured. Otherwise, the IP address shall be set to a valid class A, B, or C address and shall not be set to the loopback address (127.0.0.1) |
| | | | | Name server | UDINT | Primary name server | Value of 0 indicates no name server address has been configured. Other-wise, the name server address shall be set to a valid class A, B, or C address |
| | | | | Name server 2 | UDINT | Secondary name server | Value of 0 indicates no secondary name server address has been configured. Otherwise, the name server address shall be set to a valid class A, B, or C address |
| | | | | Domain name | STRING | Default domain name | ASCII characters. Maximum length is 48 characters. Shall be padded to an even number of characters (pad not included in length). A length of 0 shall indicate no Domain Name is configured |
| **0x06** | Required | Get<br>Set is conditional[b] | NV | Host name | STRING | Host name | ASCII characters. Maximum length is 64 characters. Shall be padded to an even number of characters (pad not included in length). A length of 0 shall indicate no Host Name is configured. See 7.5.4.6.2 |
| **0x07** | Conditional[c] | | | Safety network number | USINT[6] | | See IEC 61784-3-2 |

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x08** | Conditional[d] | Get<br>Set is conditional[e] | NV | TTL value | USINT | TTL value for CP 2/2 multicast packets | Time-to-live value for IP multicast packets. Default value is 1, minimum is 1, maximum is 255. See 7.5.4.8 |
| **0x09** | Conditional[d] | Get<br>Set is conditional[e] | NV | Mcast config | STRUCT of 8 octets | IP multicast address configuration | See 7.5.4.9 |
| | | | | Alloc control | USINT | Multicast address allocation control word. Determines how addresses are allocated | Determines whether multicast addresses are generated via algorithm or are explicitly set. See 7.5.4.9 for details |
| | | | | Reserved | USINT | Reserved for future use | Shall be 0 |
| | | | | Num mcast | UINT | Number of IP multicast addresses to allocate for CP 2/2 | The number of IP multicast addresses allocated, starting at “Mcast start addr”.<br>Maximum value is device specific, however shall not exceed the number of CP 2/2 multicast connections supported by the device |
| | | | | Mcast start addr | UDINT | Starting multicast address from which to begin allocation | IP multicast address (Class D). A block of “Num mcast” addresses is allocated starting with this address |
| **0x0A** | Conditional[f] | Set | NV | SelectAcd | BOOL | Activates the use of ACD | Enable ACD (1, default), disable ACD (0).<br>See 7.5.4.10 |
| **0x0B** | Conditional[f] | Set | NV | LastConflictDetected | STRUCT of 35 octets | Structure containing information related to the last conflict detected | ACD Diagnostic Parameters<br>See 7.5.4.11 |
| | | | | AcdActivity | USINT | State of ACD activity when last conflict detected | ACD activity<br>Default = 0 |
| | | | | RemoteMAC | USINT[6] | MAC address of remote node from the ARP PDU in which a conflict was detected | MAC from Ethernet packet header<br>Default =0 |
| | | | | ArpPdu | USINT[28] | Copy of the raw ARP PDU in which a conflict was detected. | ARP PDU<br>Default = 0 |
| **0x0C** | Optional | Set | NV | QuickConnect | BOOL | Enable/Disable of QuickConnect feature | 0 = Disable (default)<br>1 = Enable<br>See 7.5.4.12 |

NOTE ACD is specified in 7.5.7.

| | |
|---|---|
| a | Set is required unless the configuration method is selected exclusively via hardware settings. |
| b | Set is optional when the Interface Configuration attribute is not settable. |
| c | This attribute is required for CPF 2 safety devices. Non-safety devices shall not implement this attribute. |
| d | If either “TTL value” or “Mcast config” is implemented, both shall be implemented. |
| e | If either “TTL value” or “Mcast config” is implemented as settable, both shall be implemented as settable. |
| f | Required if device implements ACD. |

#### 7.5.4.2 Status

The status attribute is a bitmap that indicates the status of the TCP/IP network interface, as specified in Table 52. Refer to the state diagram in Figure 21 for a description of object states as they relate to the status attribute.

**Table 52 – Status bits**

| Bit(s) | Name | Definition |
|---|---|---|
| **0-3** | Interface configuration status | Indicates the status of the interface configuration attribute.<br>0 = The interface configuration attribute has not been configured<br>1 = The interface configuration attribute contains valid configuration obtained from BOOTP, DHCP or non-volatile storage<br>2 = The IP address member of the interface configuration attribute contains valid configuration, obtained from hardware settings (e.g. pushwheel, thumbwheel,...)<br>3-15 = Reserved for future use |
| **4** | Mcast pending | Indicates a pending configuration change in the TTL value and/or Mcast config attributes. This bit shall be set when either the TTL value or Mcast config attribute is set, and shall be cleared the next time the device starts |
| **5** | Interface configuration pending | Indicates a pending configuration change in the Interface Configuration attribute. This bit shall be 1 (TRUE) when Interface Configuration attribute are set and the device requires a reset in order for the configuration change to take effect (as indicated in the Configuration Capability attribute).<br>The intent of the Interface Configuration Pending bit is to allow client software to detect that a device’s IP configuration has changed, but will not take effect until the device is reset. |
| **6** | AcdStatus | Indicates when an IP address conflict has been detected by ACD.<br>This bit shall default to 0 (FALSE) on startup. If ACD is supported and enabled, then this bit shall be set to 1 (TRUE) any time an address conflict is detected as defined by the [ConflictDetected] transitions in Figure 23 (ACD Behavior). |
| **7** | AcdFault | Indicates when an IP address conflict has been detected by ACD or the defense failed, and that the current Interface Configuration cannot be used due to this conflict. This bit SHALL be 1 (TRUE) if an address conflict has been detected and this interface is currently in the Notification & FaultAction or AcquireNewIpv4Parameters ACD state as defined in 7.5.7, and SHALL be 0 (FALSE) otherwise.<br>Notice that when this bit is set, then this CPF 2 port will not be usable. However, for devices with multiple ports, this bit provides a way of determining if the port has an ACD fault and thus cannot be used. |
| **8-31** | Reserved | Reserved for future use and shall be set to zero |

#### 7.5.4.3 Configuration capability

The configuration capability attribute is a bitmap that indicates the device’s support for optional network configuration capability. Possible configuration capability items are listed in

Table 53. Devices are not required to support any one particular item, however they shall support at least one method of obtaining an initial IP address.

**Table 53 – Configuration capability bits**

| Bit | Name | Definition |
|---|---|---|
| **0** | BOOTP client | 1 (TRUE) shall indicate the device is capable of obtaining its network configuration via BOOTP |
| **1** | DNS client | 1 (TRUE) shall indicate the device is capable of resolving host names by querying a DNS server |
| **2** | DHCP client | 1 (TRUE) shall indicate the device is capable of obtaining its network configuration via DHCP |
| **3** | DHCP-DNS update | Shall be 0, behavior to be defined in a future edition of this standard |
| **4** | Configuration settable | 1 (TRUE) shall indicate the interface configuration attribute is settable. Some devices, for example a PC or workstation, may not allow the interface configuration to be set via the TCP/IP Interface object |
| **5** | Hardware configurable | 1 (TRUE) shall indicate the IP Address member of the Interface Configuration attribute can be obtained from hardware settings (e.g., pushwheel, thumbwheel,...).<br>If this bit is FALSE the Status instance attribute (0x01), Interface Configuration Status field value shall never be 2 (the Interface Configuration attribute contains valid configuration, obtained from hardware settings) |
| **6** | Interface configuration change requires reset | 1 (TRUE) shall indicate that the device requires a restart in order for a change to the Interface Configuration attribute to take effect.<br>If this bit is FALSE a change in the Interface Configuration attribute will take effect immediately. |
| **7** | AcdCapable | (1) TRUE shall indicate that the device is ACD capable |
| **8-31** | Reserved | Reserved for future use and shall be set to zero |

#### 7.5.4.4 Configuration control

##### 7.5.4.4.1 Configuration control structure

The configuration control attribute is a bitmap used to control network configuration options, as specified in Table 54.

**Table 54 – Configuration control bits**

| Bit | Name | Definition | |
|---|---|---|---|
| **0–3** | Configuration method | Determines how the device shall obtain its IP-related configuration | 0 = The device shall use statically-assigned IP configuration values<br>1 = The device shall obtain its interface configuration values via BOOTP.<br>2 = The device shall obtain its interface configuration values via DHCP<br>3-15 = Reserved for future use |
| **4** | DNS Enable | If 1 (TRUE), the device shall resolve host names by querying a DNS server | |
| **5–31** | Reserved | Reserved for future use and shall be set to zero | |

##### 7.5.4.4.2 Configuration method

The Configuration Method determines how a device shall obtain its IP-related configuration.

- If the Configuration Method is 0, the device shall use statically-assigned IP configuration contained in the Interface Configuration attribute (or assigned via non-Type 2 methods, as noted below).
- If the Configuration Method is 1, the device shall obtain its IP configuration via BOOTP. The BOOTP client behavior shall be as defined in the relevant IETF RFCs (IETF RFC 951, IETF RFC 1542, IETF RFC 2132, or their successors).
- If the Configuration Method is 2, the device shall obtain its IP configuration via DHCP. The DHCP client behavior shall be as defined in the relevant IETF RFCs (IETF RFC 2131, IETF RFC 2132, or their successors).
- Devices that optionally provide hardware means (e.g., rotary switch) to configure IP addressing behavior shall set the Configuration Method to reflect the configuration set via hardware: 0 if a static IP address has been configured, 1 if BOOTP has been configured, 2 if DHCP has been configured.

If a device has been configured to obtain its configuration via BOOTP or DHCP it shall continue sending requests until a response from the server is received. Devices that elect to use default IP configuration in the event of no response from the server shall continue issuing requests until a response is received, or until the Configuration Method is changed to static.

Once the device receives a response from the server it shall stop sending the BOOTP/DHCP client requests (DHCP clients shall follow the lease renewal behavior per the IETF RFC). It is recommended that devices implement the means to detect a link up and upon a link up detection restart the initial BOOTP or DHCP sequence. For multiport devices the restart of the initial BOOTP or DHCP sequence shall only be triggered if all external links have been down and when the first link up is detected.

Setting the Configuration Method to 0 (static address) shall cause the Interface Configuration to be saved to non volatile storage.

It is recommended that setting the Configuration Method to 1 (BOOTP) or 2 (DHCP) cause the device to start the BOOTP / DHCP client to obtain new IP address configuration. If the device requires a reset in order to start the BOOTP / DHCP client, it shall set the Interface Configuration Pending bit, and upon device reset start the BOOTP / DHCP client.

#### 7.5.4.4.3 DNS enable

For originator devices that support resolving target host names via DNS, the DNS Enable bit shall enable (1) and disable (0) the DNS client.

### 7.5.4.5 Physical link object

This attribute identifies the object associated with the underlying physical port. There are two components to the attribute: a path size (in UINTs) and a path. The path shall contain two logical segments (a class segment and an instance segment) that identify the physical port object. The maximum path size is 6 (assuming a 32 bit logical segment for each of the class and instance). An example path is shown in Table 55.

**Table 55 – Example path**

| Path | Meaning |
|---|---|
| **[20][F6][24][01]** | [20] = 8 bit class segment type;<br>[F6] = Ethernet Link object class;<br>[24] = 8 bit instance segment type;<br>[01] = instance 1 |

The physical link object itself typically maintains link-specific counters as well as any link-specific configuration attributes. If the port associated with the TCP/IP Interface object has an

Ethernet physical layer, this attribute shall point to an instance of the Ethernet Link object (see 7.6). When there are multiple physical interfaces that correspond to the TCP/IP interface, this attribute shall either contain a path size of 0, or shall contain a path to the object representing an internal communications interface (often used in the case of an embedded switch).

#### 7.5.4.6 Interface configuration

##### 7.5.4.6.1 Interface configuration contents

The Interface Configuration attribute contains the configuration parameters required for a device to operate as a TCP/IP node.

The contents of the Interface Configuration attribute shall depend upon how the device has been configured to obtain its IP parameters.

- If configured to use a static IP address (Configuration Method value is 0), the Interface Configuration values shall be those which have been statically assigned and stored in non-volatile storage.
- If configured to use BOOTP or DHCP (Configuration Method value is 1 or 2), the Interface Configuration values shall contain the configuration obtained from the BOOTP or DHCP server. The Interface Configuration attribute shall be 0 until the BOOTP/DHCP reply is received.
- Some devices optionally provide additional, non-CIP mechanisms for setting IP-related configuration (e.g., a web server interface, rotary switch for configuring IP address,...). When such a mechanism is used, the Interface Configuration attribute shall reflect the IP configuration values in use.

Components of the interface configuration attributes are described in Table 56.

**Table 56 – Interface configuration components**

| Name | Meaning |
|---|---|
| **IP address** | The device's IP address |
| **Network mask** | The device's network mask. The network mask is used when the IP network has been partitioned into subnets. The network mask is used to determine whether an IP address is located on another subnet |
| **Gateway address** | The IP address of the device's default gateway. When a destination IP address is on a different subnet, packets are forwarded to the default gateway for routing to the destination subnet |
| **Name server** | The IP address of the primary name server. The name server is used to resolve host names. For example, that might be contained in a fieldbus connection path |
| **Name server 2** | The IP address of the secondary name server. The secondary name server is used when the primary name server is not available, or is unable to resolve a host name |
| **Domain name** | The default domain name. The default domain name is used when resolving host names that are not fully qualified. For example, if the default domain name is "network.org", and the device needs to resolve a host name of "plc", then the device will attempt to resolve the host name as "plc.network.org" |

##### 7.5.4.6.2 Set Attributes behavior

In order to prevent incomplete or incompatible configuration, the parameters making up the Interface Configuration attribute cannot be set individually. To modify the Interface Configuration attribute, client software should first Get the Interface Configuration attribute, change the desired parameters, and then Set the attribute.

An attempt to set any of the parameters of the Interface Configuration attribute to invalid values (see Table 51) shall result in an error response with status code 0x09 "Invalid Attribute Value" to be returned. In this scenario, all of the parameters of the Interface Configuration attribute retain the values that existed prior to the invocation of the set service.

If the device has an active I/O connection, it is recommended that the device rejects the set attributes request by returning an error response with status code 0x10 "Device State Conflict".

When the value of the Configuration Method (Configuration Control attribute) is 0, the Set Attribute service shall store the new Interface Configuration values in non-volatile memory. If the device requires reset in order for new parameters to take effect (Configuration Capability bit 6 set), the device shall set the Interface Configuration Pending bit (Status attribute bit 5).

After storing the new values the device shall send a response using its current IP address (i.e., the IP address to which the set attributes request was sent).

If the device does not require reset (Configuration Capability bit 6 clear) in order for new IP configuration to take effect, after responding to the set service the device shall initiate application of the new IP parameters. While implementation-dependent, this activity is generally initiated by the TCP/IP Interface Object set service and then completed asynchronously by the TCP/IP stack.

In order to achieve consistency of device configuration behavior, it is recommended that devices support setting the Interface Configuration attribute, and support application of new attribute values without requiring device reset. If a device does not support setting the Interface Configuration attribute, the device shall return an error response with status code 0x0E "Attribute Not Settable".

#### 7.5.4.6.3 Application of new configuration parameters

If the device does not require reset (Configuration Capability bit 6 clear), after responding to the set service the device shall initiate application of the new IP parameters. When applying new IP configuration, the device shall maintain the consistency of the IP configuration context in which it operates. For example, the device shall not mix old and new IP address / network mask / gateway address values, and shall not use the new IP configuration on Type 2 connections established with the previous IP address.

When a TCP/IP stack is configured to use a particular set of IP parameters, a context around these IP parameters is built. This context includes relationships to the TCP/IP stack, the Type 2 communications environment, and the control application among other entities. To allow a different set of IP parameters to be used a new IP context shall be built. The device shall properly manage its IP context and maintain its consistency. For example a new IP Address shall not be used with old Network Mask or Gateway Address; an old IP address shall not be used for Type 2 or other communication once the new one is applied.

### 7.5.4.7 Host name

The Host Name attribute contains the device's host name, which can be used for informational purposes. The set access is optional when the Interface Configuration attribute is not settable.

### 7.5.4.8 TTL value

TTL value is the value the device shall use for the IP header Time-to-live field when sending CP 2/2 packets via IP multicast. By default, TTL Value shall be 1. The maximum value for TTL is 255. Unicast packets shall use the TTL as configured for the TCP/IP stack, and not the TTL value configured in this attribute.

When set, the TTL Value attribute shall be saved in non-volatile memory. If a device does not support applying the TTL Value immediately, the Mcast pending bit in the Interface status attribute shall be set, indicating that there is pending configuration. For devices that support applying the TTL Value immediately, if there are existing multipoint connections, an Object State Conflict error (0xC) shall be returned and the Mcast Pending bit shall not be set. When a new TTL value is pending, Get_Attribute_Single or Get_Attribute_All requests shall return the pending value. The Mcast pending bit shall be cleared the next time the device starts.

Users should exercise caution when setting the TTL value greater than 1, to prevent unwanted multicast traffic from propagating through the network.

NOTE IEC 61158-6-2 includes a discussion on user considerations when using multicast.

#### 7.5.4.9 Mcast config

The Mcast config attribute contains the configuration of the device's IP multicast addresses to be used for CP 2/2 multicast packets. There are three elements to the Mcast config structure: Alloc control, Num mcast, and Mcast start addr.

- Alloc control determines how the device shall allocate IP multicast addresses (e.g., whether by algorithm, whether they are explicitly set,...). Table 57 shows the details for alloc control.

**Table 57 – Alloc control values**

| Value | Definition |
|---|---|
| 0 | Multicast addresses shall be generated using the default allocation algorithm specified in IEC 61158-6-2. When this value is specified on a Set_Attribute_Single or Set_Attribute_All, the values of Num mcast and Mcast start addr in the Set_Attribute request shall be 0 |
| 1 | Multicast addresses shall be allocated according to the values specified in Num mcast and Mcast start addr |
| 2 | Reserved |

- Num mcast is the number of IP multicast addresses allocated. The maximum number of multicast addresses is device specific, but shall not exceed the number of CP 2/2 multicast connections supported by the device.
- Mcast start addr is the starting multicast address from which Num mcast addresses are allocated.

When set, the Mcast config attribute shall be saved in non-volatile memory. If a device does not support applying the MCast Config attribute immediately, the Mcast Pending bit in the Interface status attribute shall be set, indicating that there is pending configuration. For devices that support applying the Mcast Config attribute immediately, if there are existing multipoint connections, an Object State Conflict error (0xC) shall be returned and the MCast Pending bit shall not be set. When a new Mcast Config value is pending, Get_Attribute_Single or Get_Attribute_All requests shall return the pending value. The Mcast pending bit shall be cleared the next time the device starts.

When the multicast addresses are generated using the default algorithm, Num mcast and Mcast start addr shall report the values generated by the algorithm.

#### 7.5.4.10 SelectAcd

SelectAcd is an attribute used to Enable/Disable ACD.

If SelectAcd is 0 then ACD is disabled. If SelectAcd =1 then ACD is enabled.

The default value of SelectAcd shall be 1 indicating that ACD is enabled.

When the value of SelectAcd is changed by a Set_Attribute service, the new value of SelectAcd shall not be applied until the device executes a restart.

#### 7.5.4.11 LastConflictDetected

The LastConflictDetected attribute is a diagnostic attribute presenting information about the ACD state when the last IP Address conflict was detected. This attribute shall be updated by the device whenever an incoming ARP packet is received that represents a conflict with the device's IP address as described in IETF RFC 5227.

To reset this attribute the Set_Attribute_Single service is invoked with an attribute value of all 0. Values other than 0 shall result in an error response (status code 0x09, Invalid Attribute Value).

There are three elements to the LastConflictDetected structure: AcdActivity, RemoteMac and ArpPdu.

- AcdActivity contains the state of the ACD algorithm when the last IP address conflict was detected. The ACD activities are defined in Table 58.

**Table 58 – AcdActivity values**

| Value | AcdMode | Description |
|---|---|---|
| 0 | NoConflictDetected (Default) | No conflict has been detected since this attribute was last cleared |
| 1 | ProbeIpv4Address | Last conflict detected during ProbeIpv4Address state |
| 2 | OngoingDetection | Last conflict detected during OngoingDetection state or subsequent DefendWithPolicyB state |
| 3 | SemiActiveProbe | Last conflict detected during SemiActiveProbe state or subsequent DefendWithPolicyB state |

- RemoteMac contains the ISO/IEC 8802-3 source MAC address from the header of the received Ethernet packet which was sent by a device reporting a conflict.
- ArpPdu contains the ARP Response PDU in binary format. The ArpPdu shall be a copy of the ARP message that caused the address conflict. It shall be a raw copy of the ARP message as it appears on the Ethernet network, i.e.: ArpPdu[1] contains the first octet of the ArpPdu received (see Table 59).

**Table 59 – ArpPdu – ARP Response PDU in binary format**

| Field size (octets) | Field description | Field value |
|---|---|---|
| 2 | Hardware Address Type | 1 for Ethernet H/W |
| 2 | Protocol Address Type | 0x800 for IP |
| 1 | HADDR LEN | 6 for Ethernet H/W |
| 1 | PADDR LEN | 4 for IP |
| 2 | OPERATION | 1 for Request or 2 for Response |
| 6 | SENDER HADDR | Sender's H/W address |
| 4 | SENDER PADDR | Sender's proto address |
| 6 | TARGET HADDR | Target's H/W address |
| 4 | TARGET PADDR | Target's proto address |

#### 7.5.4.12 QuickConnect™ [4]

The QuickConnect attribute shall enable (1) or disable (0) the QuickConnect feature. The default value of the attribute shall be 0.

NOTE The QuickConnect™ implementation profile is specified by ODVA, Inc (see <www.odva.org>).

### 7.5.5 Common services

#### 7.5.5.1 General

The TCP/IP Interface object shall support the common services as specified in Table 60.

**Table 60 – TCP/IP Interface object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| **0x01** | Optional | Optional | Get_Attribute_All | Returns a predefined listing of this objects attributes (see the Get_Attribute_All response definition in 7.5.5.2) |
| **0x02** | N/A | Optional | Set_Attribute_All | Modifies all settable attributes |
| **0x0E** | Required | Required | Get_Attribute_Single | Returns the contents of the specified attribute |
| **0x10** | N/A | Required | Set_Attribute_Single | Modifies a single attribute |

#### 7.5.5.2 Get_Attribute_All response

At the class level, the Get_Attribute_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

For instance attributes, attributes shall be returned in numerical order up to the last implemented attribute. The Get_Attribute_All reply shall be as specified in Table 61.

**Table 61 – Get_Attribute_All reply format**

| Attribute ID | Size (octets) | Contents |
|---|---|---|
| **1** | 4 | Status |
| **2** | 4 | Configuration capability |
| **3** | 4 | Configuration control |
| **4** | 2 | Physical link object, path size |
| | Variable, 12 octets max | Physical link object, path (if path size is non-zero) |

[4] QuickConnect™ is a trade name of ODVA, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance to this profile does not require use of the trade name QuickConnect™. Use of the trade name QuickConnect™ requires permission from ODVA, Inc.

| Attribute ID | Size (octets) | Contents |
|---|---|---|
| **5** | 4 | IP address |
| | 4 | Network mask |
| | 4 | Gateway address |
| | 4 | Name server |
| | 4 | Secondary name server |
| | 2 | Domain name length |
| | Variable, equal to domain name length | Domain name |
| | 1 | Pad octet only if Domain Name Length is odd |
| **6** | 2 | Host name length |
| | variable, equal to host name length | Host name |
| | 1 | Pad octet only if Host Name Length is odd |
| **7** | 6 | See IEC 61784-3-2.<br>Not present if attribute 7 is not implemented. Shall be 0 when additional attributes greater than attribute ID 7 are included |
| **8** | 1 | TTL value.<br>Not present if attribute 8 is not implemented. Shall be 0 when additional attributes greater than attribute ID 8 are included |
| **9** | 8 | Mcast config.<br>Not present if attribute 9 is not implemented. Shall be 0 when additional attributes greater than attribute ID 9 are included |
| **10** | 1 octet | SelectACD value.<br>Not present if attribute 10 is not implemented. Shall be 0 when additional attributes greater than attribute ID 10 are included. |
| **11** | 1 octet | AcdActivity<br>Not present if attribute 11 is not implemented. Shall be 0 when additional attributes greater than attribute ID 11 are included. |
| | 6 octets | RemoteMAC<br>Not present if attribute 11 is not implemented. Shall be 0 when additional attributes greater than attribute ID 11 are included. |
| | 28 octets | ArpPdu<br>Not present if attribute 11 is not implemented. Shall be 0 when additional attributes greater than attribute ID 11 are included. |
| **12** | 1 octet | QuickConnect<br>Not present if attribute 12 is not implemented. |

The lengths of the physical link object path, domain name, and host name are not known before issuing the Get_Attribute_All service request. Implementers shall be prepared to accept a response containing the maximum sizes of the physical link object path (6 UINTs), the domain name (48 USINTs), and host name (64 USINTs).

#### 7.5.5.3 Set_Attribute_All request

The instance Set_Attribute_All request shall contain the configuration control attribute, followed by the interface configuration attribute.

### 7.5.6 Behavior

The behavior of the TCP/IP Interface object shall be as illustrated in the State Transition Diagram shown in Figure 21 and Figure 22.

Note that the act of obtaining an initial executable image via BOOTP/TFTP shall not be considered within the scope of the TCP/IP Interface object behavior. Devices are free to implement such behavior, however it shall be considered to have occurred in the "Non-existent" state.

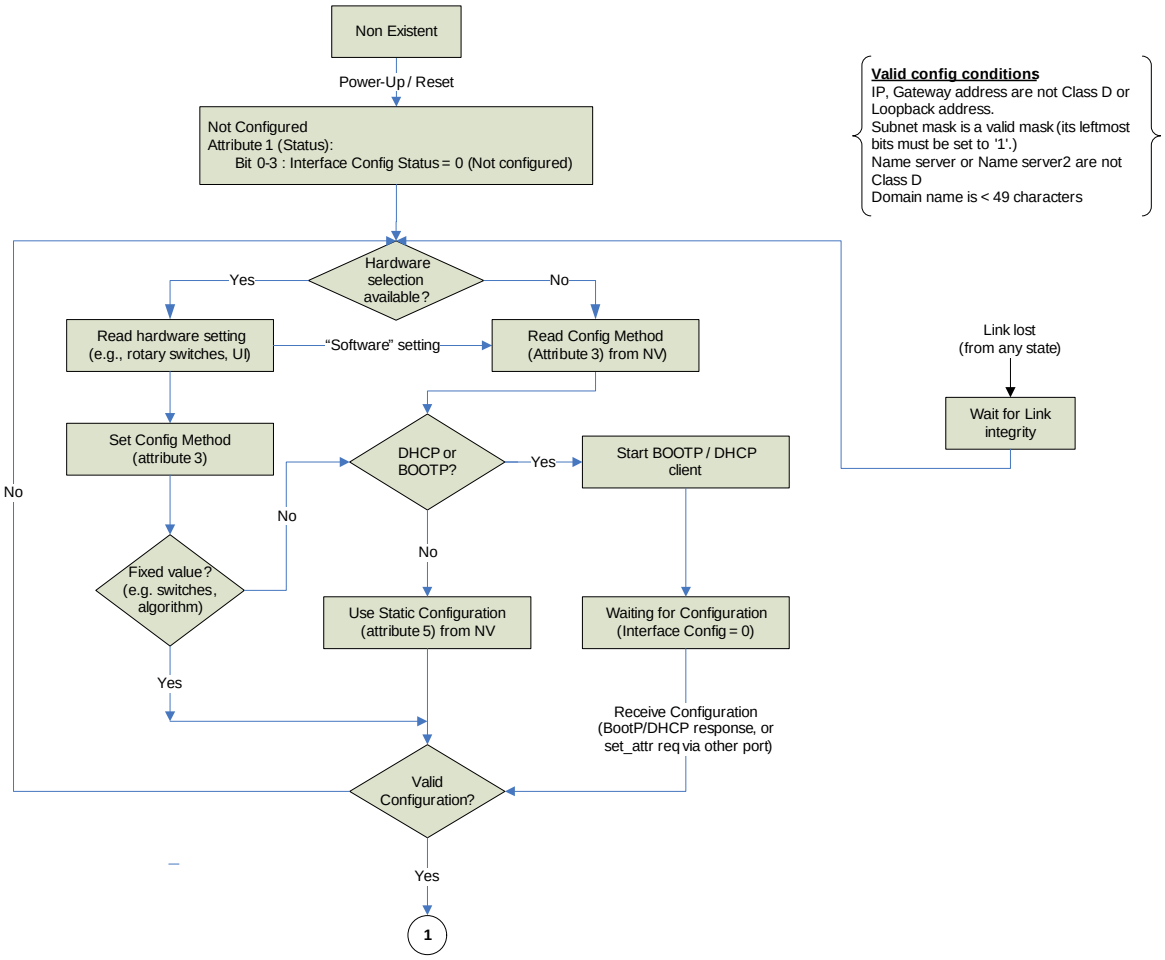



**Figure 21 – State transition diagram for TCP/IP Interface object**

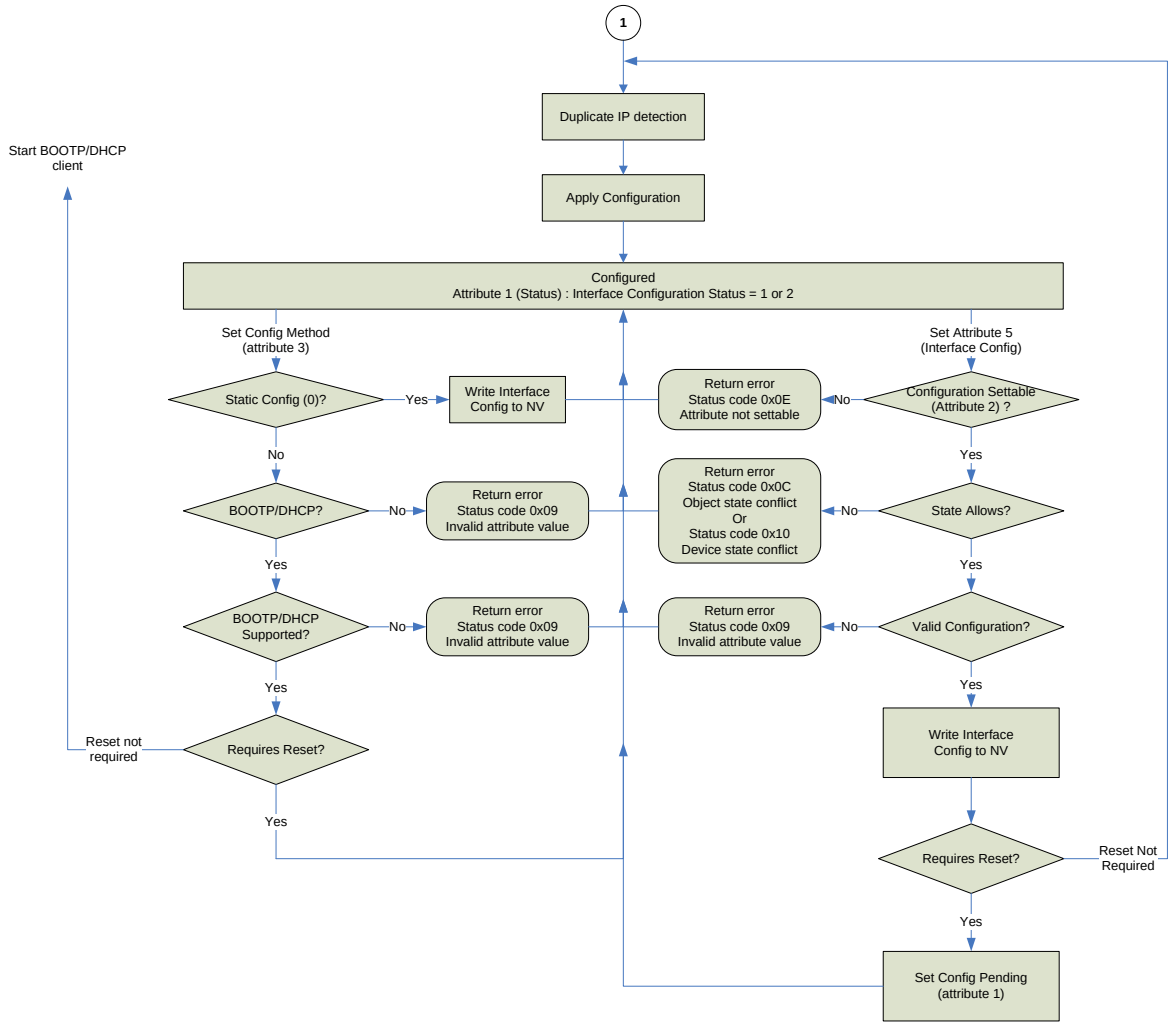



**Figure 22 – State transition diagram for TCP/IP Interface object**

### 7.5.7 Address Conflict Detection (ACD)

#### 7.5.7.1 General

##### 7.5.7.1.1 ACD requirements for Type 2 devices

Address conflict detection (ACD) is a mechanism that devices can use to detect and act upon IPv4 address conflicts.

The ACD mechanism specified in this Subclause 7.5.7 conforms to IETF RFC 5227. The requirements specified in IETF RFC 5227 are included by reference except where superseded by normative requirements in 7.5.7. This Subclause 7.5.7 specifies additional requirements for Type 2 devices with respect to the IPv4 ACD mechanism.

The following ACD requirements apply to Type 2 devices:

- ACD is recommended;
- a device implementing ACD shall conform to the ACD mechanism specified in IETF RFC 5227 except where superseded by normative requirements in this Annex;
- a device executing ACD shall execute the ACD mechanism regardless of the method used by the device for obtaining its IP parameter set;
- a device, executing ACD, and not executing the QuickConnect algorithm shall execute the ACD mechanism before using an IP parameter set;

- devices shall respond to ARP Probes.

#### 7.5.7.1.2 Overview of ACD mechanism in IETF RFC 5227

The IPv4 ACD mechanism described by IETF RFC 5227 involves the following activities:

- initial address probing & conflict detection – before using an IP address the device issues ARP Probes to detect whether the address is in use by another device;
- address announcement – an ARP Request packet is sent after determination that there are no IP address conflicts;
- ongoing conflict detection – ongoing process that is in effect for as long as a device is using an IP Address;
- address defense – procedure used to resolve an address conflict.

### 7.5.7.2 ACD behavior

The principles of operation for the ACD mechanism are specified in IETF RFC 5227.

In particular, IETF RFC 5227, 2.1, requires that "... a host implementing this specification must test to see if the address is already in use ... when a link-state change signals that an Ethernet cable has been connected ...".

In addition, this Subclause 7.5.7 further refines the ACD behavior description in the following ways.

– The activities of ACD are grouped into regions, namely:
  - active phase;
  - passive phase:
  - semi-active phase.
– A more complete set of link_up transitions are included in the diagram.
– Both single port and multi-port devices are accommodated.

A device shall implement the ACD mechanism with the behavior specified in Figure 23.

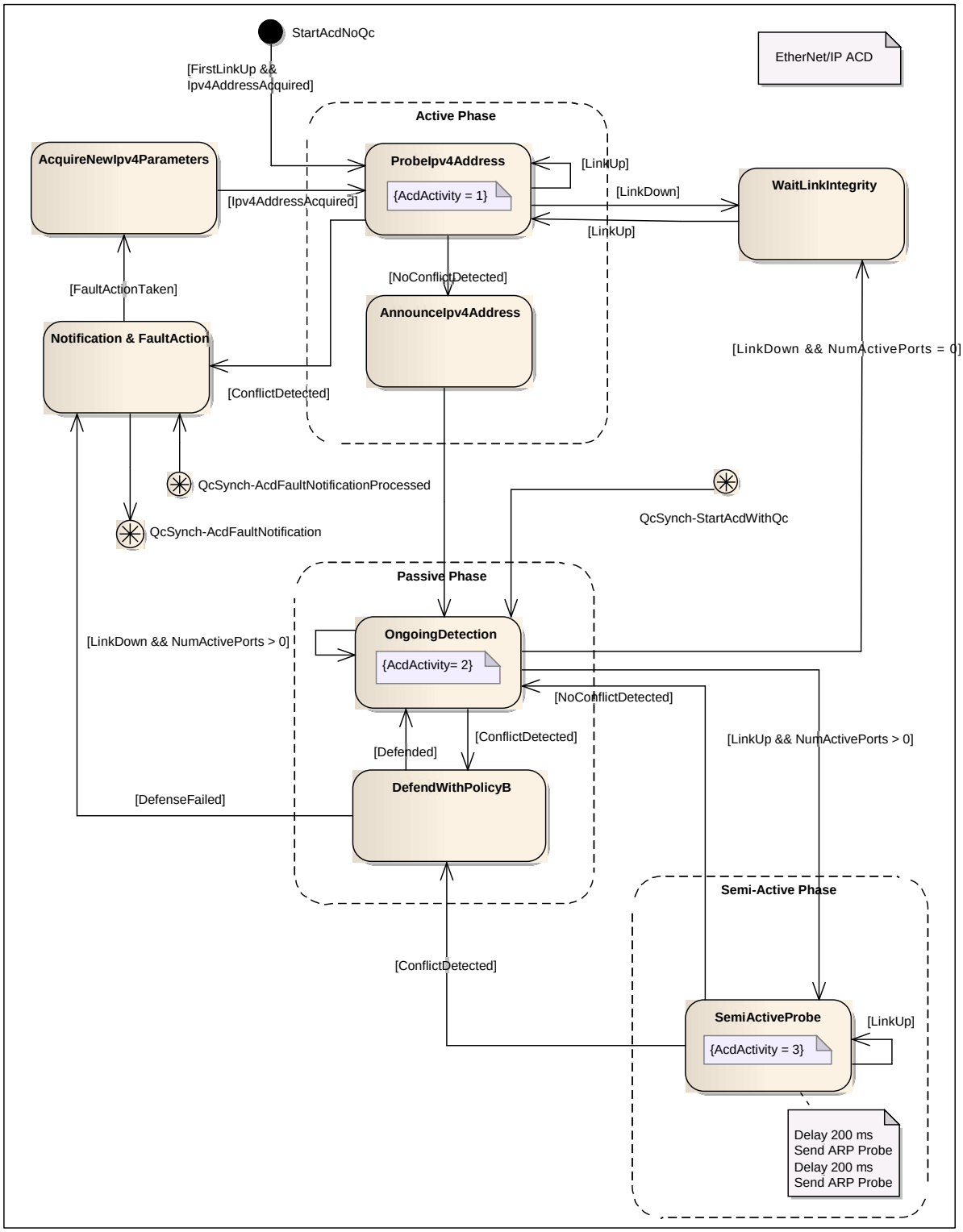



**Figure 23 – ACD Behavior**

#### 7.5.7.3 ACD details

##### 7.5.7.3.1 IPv4 ACD Timing Constants

IETF RFC 5227, 1.1, specifies a number of timing-related constants. Some of these constants are not optimal for Type 2 applications and networks. In particular, the start-up delays caused by the ARP Probe intervals would be unacceptable in the industrial setting. Adaptation of the specified alternate values is consistent with IETF RFC 5227, 1.3, on Applicability.

Devices that provide ACD shall implement the following alternate values for the parameters defined in IETF RFC 5227, 1.1.

- PROBE_WAIT: 200 ms (initial random delay)
- PROBE_NUM: 4 (number of probe packets)
- PROBE_MIN: 200 ms (minimum delay until repeated probe)
- PROBE_MAX: 200 ms (maximum delay until repeated probe)
- ANNOUNCE_WAIT: 200 ms (delay before announcing)
- ANNOUNCE_INTERVAL: 2 s (delay between announce packets)
- DEFEND_INTERVAL: 2 s (minimum interval between defensive ARPs)

Devices shall comply with the timing in IETF RFC 5227 as modified above within ±10 %. Notice that PROBE_MIN and PROBE_MAX being set to the same value represents the CPF 2 recommendation to use a fixed interval between probes rather than the recommendation in IETF RFC 5227, 2.1.1 (Probe Details) that subsequent probes after the initial probe be randomized between PROBE_MIN and PROBE_MAX.

##### 7.5.7.3.2 ProbeIpv4Address

See IETF RFC 5227, 2.1 and 2.1.1.

##### 7.5.7.3.3 AnnounceIpv4Address

See IETF RFC 5227, 2.3.

When the device has sent its first ARP Announcement it shall transition to OngoingDetection. If no conflict is detected within ANNOUNCE_INTERVAL seconds, then the device shall complete the send of its second ARP Announcement asynchronously with other network operations.

##### 7.5.7.3.4 WaitLinkIntegrity

The WaitLinkIntegrity activity waits for the occurrence of a LinkUp event from an Ethernet port.

This activity is initiated in two scenarios:

- one is when a single port device experiences a LinkDown event;
- the other is when a multi-port device experiences a LinkDown event and all of its other Ethernet ports are also down.

When a LinkUp event occurs the ProbeIpv4Address activity is then initiated.

When a device detects that Ethernet link integrity has been lost and then regained (e.g., the network cable has been removed and replaced), the device shall restart the initial ProbeIpv4Address activity.

#### 7.5.7.3.5 Notification & Fault Action

See IETF RFC 5227, Clause 1.

In addition to IETF RFC 5227 behavior, when an address conflict is detected, Type 2 devices shall take the following fault actions:

- set the device state to Recoverable Fault (Module Status LED flashing red);
- set the Network Status LED to solid red.

There are 2 synchronization transitions between the ACD Notification & FaultAction activity and the QuickConnect behavior diagram.

The QcSynch-AcdFaultNotification is a synchronization transition to the QuickConnect behavior diagram that occurs either when the ACD mechanism has a ConflictDetected fault from the ProbeIpAddress activity or a DefenseFailed fault from the DefendWithPolicyB activity.

The QcSynch-AcdFaultNotificationProcessed is a synchronization transition from the QuickConnect behavior diagram to the Notification & FaultAction activity that occurs after QuickConnect has processed the previous AcdFaultNotification.

#### 7.5.7.3.6 AcquireNewIpv4Parameters

See IETF RFC 5227, Clause 1.

After notification of the detection of an IPv4 Address conflict, the device may be designed to obtain or use an alternate IPv4 address. The design of this method and its resulting selection of IPv4 Parameters are vendor specific.

If a LinkDown occurs on the last active port while in the AcquireNewIpv4Parameter activity, then the ACD process in Figure 23 (ACD Behavior) shall terminate. The ACD process shall be restarted after at least one link has been reestablished and a valid IPv4 configuration has been acquired according to Figure 21 and Figure 22 showing the behavior of the TCP/IP Object.

Notice also that CP 2/2 devices shall limit the rate at which they probe for new candidate addresses using MAX_CONFLICTS and RATE_LIMIT_INTERVAL as defined in IETF RFC 5227, 2.1.1.

#### 7.5.7.3.7 OngoingDetection

See IETF RFC 5227, 2.4.

The QcSynch-StartAcdWithQc is a synchronization transition with the QuickConnect behavior diagram. If QuickConnect is enabled in the device, the activity for ACD begins at this point.

The ACD mechanism as specified in IETF RFC 5227, 2.1 states that “A host must not perform this check periodically as a matter of course”, indicating that periodic ARP Probes are not required to be sent as part of ongoing detection. However it has been observed that periodic ARP Probes allow the module to detect conflicts with devices that may not have been connected to the network at the time of initial probing, or when switches have dropped the initial ARP Probes due to initial forwarding delay. Therefore, Type 2 devices that provide the ACD algorithm should issue periodic ARP Probes during ongoing detection.

Type 2 devices that support periodic ARP Probes during ongoing detection shall use a transmission delay in the range of ONGOING_PROBE_MIN and ONGOING_PROBE_MAX using the values below:

- ONGOING_PROBE_MIN: 90 s (minimum time interval for ongoing probes);
- ONGOING_PROBE_MAX: 150 s (maximum time interval for ongoing probes).

These values are not defined in IETF RFC 5227. Devices shall not exceed this specified range by more than 10 % on either end.

It is recommended that the initial transmission delay for the first periodic ARP Probe has a pseudo-random value within the range of ONGOING_PROBE_MIN and ONGOING_PROBE_MAX (“randomized” using the Ethernet MAC address, as shown for example in 7.5.7.3.8.2). The transmission delay for subsequent periodic ARP Probes need not be randomized, and may, for example, select the center value (120 s).

#### 7.5.7.3.8 DefendWithPolicyB

##### 7.5.7.3.8.1 Usage

See IETF RFC 5227, 2.4.

IETF RFC 5227, 2.4 states that “Address Conflict Detection is not limited to only the time of initial interface configuration, when a host is sending ARP Probes. Address Conflict Detection is an ongoing process that is in effect for as long as a host is using an address.”

After an IP address has successfully been probed and put into use, a device shall perform ongoing conflict detection and defense according to IETF RFC 5227.

If an address conflict is detected, the device shall defend its address per alternative (b) in IETF RFC 5227, 2.4.

If a conflict persists (as defined by IETF RFC 5227) the device shall follow the behavior as when a conflict is detected during initial probing (device state, LEDs, DHCP behavior,...) and execute the specified notification & fault actions.

##### 7.5.7.3.8.2 Example pseudo-random delay algorithm (informative)

NOTE The following Xorshift Random Number Generator algorithm is based on the work of George Marsaglia from Florida State University.

This Random Number Generator (RNG) is used to produce an initial pseudo-random delay before sending out the first periodic ARP probe.

The goal is to produce a random number in the range of PROBE_WAIT (ie 180 to 200), by calculating a random offset from the beginning of the range.

PROBE_WAIT is calculated according to the following formula:

PROBE_WAIT = PROBE_WAIT_MIN + $\Delta$rn

$\Delta$rn is the random offset ranging from (0 .. to (PROBE_WAIT_MAX – PROBE_WAIT_MIN) and calculated according to the following formula:

$\Delta$rn = (PROBE_WAIT_MAX – PROBE_WAIT_MIN) * R256 / 256.

R256 is a random number in the range (0 .. 255) calculated using the algorithm below.

The RNG algorithm uses an IEEE 802.3 MAC address, eg 12-34-56-78-9A-BC, written here in canonical notation. Canonical notation represents the order in which the octets of the MAC address are transmitted, left to right, on the wire; i.e. 0x12 sent first, 0x34 sent next, and so on. The MAC address is mapped to an array, EnetAddr[], of 6 octets as follows, using the example MAC address.

EnetAddr[0] = 0x12;

EnetAddr[1] = 0x34;

EnetAddr[2] = 0x56;

EnetAddr[3] = 0x78;

EnetAddr[4] = 0x9A;

EnetAddr[5] = 0xBC;

R256 is then calculated using the following Xorshift RNG.

R256 = EnetAddr[0];

R256 = (R256 << 1 ) ^ EnetAddr[1];

R256 = (R256 << 1 ) ^ EnetAddr[2];

R256 = (R256 << 1 ) ^ EnetAddr[3];

R256 = (R256 << 1 ) ^ EnetAddr[4];

R256 = (R256 << 1 ) ^ EnetAddr[5];

R256 = R256 % 256;

Where << is the left shift operator, ^ is the exclusive OR operator, and % is the modulus operator.

#### 7.5.7.3.9 SemiActiveProbe

The SemiActiveProbe activity is initiated when a multi-port device receives a LinkUp event while other ports are still active.

The SemiActiveProbe activity is a modified probing activity in which only two ARP probes are sent using probe delays of 200 ms.

The SemiActiveProbe activity is defined to reduce the amount of ARP broadcast traffic that will be initiated from LinkUp activity on multi-port devices.

### 7.6 Ethernet link object

#### 7.6.1 Overview

The Ethernet Link object maintains link-specific counters and status information for a physical Ethernet ISO/IEC 8802-3 port. Each device shall support exactly one instance of the Ethernet Link object for each Ethernet port on the module. Devices may use an Ethernet Link object instance for an internally accessible interface, such as an internal port for an embedded switch.

#### 7.6.2 Revision history

The revision history of the Ethernet Link object is described in Table 62.

**Table 62 – Ethernet link object revision history**

| Class revision | Description of changes |
|---|---|
| **01** | Initial Release |
| **02** | Release for IEC 61158. |
| **03** | Add new instance attributes 7-10 providing support for multiple port Ethernet devices |

### 7.6.3 Class attributes

The Ethernet Link object shall support the class attributes as specified in Table 63.

**Table 63 – Ethernet link object class attributes**

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x01** | Required if the revision value is greater than 1 | Get | NV | Revision | UINT | Revision of this object | The minimum value shall be 1.<br>Shall be 2 or greater if instance attribute 6 is implemented.<br>Shall be 3 if any instance attributes 7-10 are implemented.<br>The maximum value shall be 3. |
| **0x02** | Required if the number of instances is greater than 1 | Get | NV | Max Instance | UINT | Maximum instance number of an object currently created in this class level of the device | The largest instance number of a created object at this class hierarchy level |
| **0x03** | Required if the number of instances is greater than 1 | Get | NV | Number of Instances | UINT | Number of object instances currently created at this class level of the device | The number of object instances at this class hierarchy level |
| **0x04 to 0x07** | These class attributes are optional and are described in IEC 61158-5-2. | | | | | | |

An error reading the class revision attribute implies this is a revision 1 only implementation.

### 7.6.4 Instance attributes

#### 7.6.4.1 General

The Ethernet Link object shall support the instance attributes as specified in Table 64.

**Table 64 – Ethernet link object instance attributes**

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x01** | Required | Get | V | Interface speed | UDINT | Interface speed currently in use | Speed in Mbit/s (e.g. 0, 10, 100, 1 000). See 7.6.4.2 |
| **0x02** | Required | Get | V | Interface flags | DWORD | Interface status flags | Bit map of interface flags. See 7.6.4.3 |
| **0x03** | Required | Get | NV | Physical address | USINT[6] | MAC layer address | See 7.6.4.4 |
| **0x04** | Conditional[a] | Get | V | Interface counters | STRUCT of 44 octets | | See 7.6.4.5 |
| | | | | In Octets | UDINT | Octets received on the interface | |

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| | | | | In Ucast Packets | UDINT | Unicast packets received on the interface | |
| | | | | In NUcast Packets | UDINT | Non-unicast packets received on the interface | |
| | | | | In Discards | UDINT | Inbound packets received on the interface but discarded | |
| | | | | In Errors | UDINT | Inbound packets that contain errors (does not include In Discards) | |
| | | | | In Unknown Protos | UDINT | Inbound packets with unknown protocol | |
| | | | | Out Octets | UDINT | Octets sent on the interface | |
| | | | | Out Ucast Packets | UDINT | Unicast packets sent on the interface | |
| | | | | Out NUcast Packets | UDINT | Non-unicast packets sent on the interface | |
| | | | | Out Discards | UDINT | Outbound packets discarded | |
| | | | | Out Errors | UDINT | Outbound packets that contain errors | |
| **0x05** | Optional | Get | V | Media counters | STRUCT of 48 octets | Media-specific counters | See 7.6.4.6 |
| | | | | Alignment errors | UDINT | frames received that are not an integral number of octets in length | |
| | | | | FCS errors | UDINT | DLPDUs received that do not pass the FCS check | |
| | | | | Single collisions | UDINT | Successfully transmitted DLPDUs which experienced exactly one collision | |
| | | | | Multiple collisions | UDINT | Successfully transmitted DLPDUs which experienced more than one collision | |
| | | | | SQE test errors | UDINT | Number of times SQE test error message is generated | |
| | | | | Deferred transmissions | UDINT | DLPDUs for which first transmission attempt is delayed because the medium is busy | |

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| | | | | Late collisions | UDINT | Number of times a collision is detected later than 512 bit-times into the transmission of a packet | |
| | | | | Excessive collisions | UDINT | DLPDUs for which transmission fails due to excessive collisions | |
| | | | | MAC transmit errors | UDINT | DLPDUs for which transmission fails due to an internal MAC sublayer transmit error | |
| | | | | Carrier sense errors | UDINT | Times that the carrier sense condition was lost or never asserted when attempting to transmit a frame | |
| | | | | Frame too long | UDINT | DLPDUs received that exceed the maximum permitted DLPDU size | |
| | | | | MAC receive errors | UDINT | DLPDUs for which reception on an interface fails due to an internal MAC sublayer receive error | |
| **0x06** | Optional | Get/Set | NV | Interface control | STRUCT of | Configuration for physical interface | See 7.6.4.7 |
| | | | | Control bits | WORD | Interface control bits | |
| | | | | Forced interface speed | UINT | Speed at which the interface shall be forced to operate | Speed in Mbit/s (e.g. 10, 100, 1 000) |
| **0x07** | Optional | Get | NV | Interface Type | USINT | Type of interface, e.g. twisted pair, fiber, internal | See 7.6.4.8 |
| **0x08** | Optional | Get | V | Interface State | USINT | Current state of the interface, e.g. operational, disabled | See 7.6.4.9 |
| **0x09** | Optional | Set | NV | Admin State | USINT | Administrative state: enable, disable | See 7.6.4.10 |
| **0x0A** | Conditional[b] | Get | NV | Interface Label | SHORT_STRING_ | Human readable identification | See 7.6.4.11 |

[a] The interface counters attribute is required if the media counters attribute is implemented.

[b] Required if number of instances is greater than 1.

#### 7.6.4.2 Interface speed

The interface speed attribute shall indicate the speed at which the interface is currently running (e.g. 10 Mbit/s, 100 Mbit/s, 1 Gbit/s, or other speeds). A value of 0 shall be used to indicate that the speed of the interface is indeterminate.

The scale of the attribute is in Mbit/s, so if the interface is running at 100 Mbit/s then the value of the interface speed attribute shall be 100. The interface speed is intended to represent the available link transmit time; the attribute shall not be doubled if the interface is running in full-duplex mode.

#### 7.6.4.3 Interface flags

The interface flags attribute contains status and configuration information about the physical interface as specified in Table 65.

**Table 65 – Interface flags bits**

| Bit | Name | Definition |
|---|---|---|
| **0** | Link status | Indicates whether or not the port is connected to an active network:<br>- 0 indicates an inactive link<br>- 1 indicates an active link<br>The determination of link status is implementation specific. In some cases devices can tell whether the link is active via hardware/driver support. In other cases, the device may only be able to tell whether the link is active by the presence of incoming packets |
| **1** | Half/Full duplex | Indicates the duplex mode currently in use:<br>- 0 indicates the port is running half duplex<br>- 1 indicates full duplex<br>If the Link status flag is 0, then the value of the Half/Full duplex flag is indeterminate. |
| **2-4** | Negotiation status | Indicates the status of link auto-negotiation:<br>0 = Auto-negotiation in progress<br>1 = Auto-negotiation and speed detection failed. Using default values for speed and duplex. Default values are product-dependent; recommended defaults are 10Mbit/s and half duplex<br>2 = Auto negotiation failed but detected speed. Duplex was defaulted. Default value is product-dependent; recommended default is half duplex<br>3 = Successfully negotiated speed and duplex<br>4 = Auto-negotiation not attempted. Forced speed and duplex |
| **5** | Manual setting requires reset | 0 indicates the interface can activate changes to link parameters (auto-negotiate, duplex mode, interface speed) automatically<br>1 indicates the device requires a Reset service be issued to its Identity object in order for the changes to take effect |
| **6** | Local hardware fault | 0 indicates the interface detects no local hardware fault<br>1 indicates a local hardware fault is detected<br>The meaning of this is product-specific. Examples are an AUI/MII interface detects no transceiver attached or a radio modem detects no antennae attached. In contrast to the soft, possible self-correcting nature of the Link Status being inactive, this is assumed a hard-fault requiring user intervention |
| **7-31** | Reserved | Reserved and shall be set to zero |

#### 7.6.4.4 Physical address

The physical address attribute contains the interface's MAC layer address. The physical address is an array of octets. The recommended display format is "XX-XX-XX-XX-XX-XX", starting with the first octet. The physical address is not a settable attribute: the Ethernet address shall be assigned by the manufacturer, and shall be unique per ISO/IEC 8802-3 requirements.

Devices with multiple ports but a single MAC interface (e.g., a device with a embedded switch technology) may use the same value for this attribute in each instance of the Ethernet Link object. The general requirement is that the value of this attribute shall be the MAC address used for packets to and from the device's own MAC interface over this physical port.

#### 7.6.4.5 Interface counters

The interface counters attribute contains counters relevant to the receipt of packets on the interface. These counters shall be as defined in IETF RFC 1213. The interface counters is a conditional attribute; it shall be implemented if the media counters attribute is implemented.

Multi-port devices with a single MAC interface (e.g., device with an embedded switch) shall maintain counter values in one of three ways.

- In each instance, count the MAC frames sent/received by the device itself over the port represented by that instance (i.e., each physical interface counts the MAC frames sent/received over that interface). This is the preferred solution.
- Use counter values of 0 for those instances that correspond to the external switch ports; count MAC frames in the instance that corresponds to the internal device interface.
- Use the same counter values for all instances, counting MAC frames sent/received by the device itself.

#### 7.6.4.6 Media counters

The media counters attribute contains counters specific to Ethernet media. These counters shall be as defined by IETF RFC 1643. If this attribute is implemented the interface counters attribute shall also be implemented. Instances that refer to internal interfaces may set the values of the Interface Counters to 0.

Some underlying hardware or system implementations may not provide all of the Media Counters. In the case of fiber media, some of the counters do not apply (e.g., collision counters). Devices shall use values of 0 for counters that are not implemented.

#### 7.6.4.7 Interface control

##### 7.6.4.7.1 Overview

The interface control attribute is a structure consisting of control bits and forced interface speed and shall be as specified in 7.6.4.7.2 to 7.6.4.7.3.

##### 7.6.4.7.2 Control bits

The control bits attribute is specified in Table 66.

**Table 66 – Control bits**

| Bit | Name | Definition |
|---|---|---|
| **0** | Auto-negotiate | 0 indicates ISO/IEC 8802-3 link auto-negotiation is disabled<br>1 indicates auto-negotiation is enabled<br><br>If auto-negotiation is disabled, then the device shall use the settings indicated by the forced duplex mode and forced interface speed bits |
| **1** | Forced duplex mode | If the Auto-negotiate bit is 0, the forced duplex mode bit indicates whether the interface shall operate in full or half duplex mode.<br><br>0 indicates the interface duplex should be half duplex<br>1 indicates the interface duplex should be full duplex<br><br>Interfaces not supporting the requested duplex shall return an error code 0x09 (Invalid Attribute Value). If auto-negotiation is enabled, attempting to set the forced duplex mode bits shall result in an error code 0x0C (Object State Conflict) |
| **2-15** | Reserved | Shall be set to zero |

#### 7.6.4.7.3 Forced interface speed

If the auto-negotiate bit is 0, the forced interface speed bits indicate the speed at which the interface shall operate. Speed is specified in Mbit/s (e.g. for 10 Mbit/s Ethernet, the interface speed shall be 10). Interfaces not supporting the requested speed should return an error code 0x09 (Invalid Attribute Value).

If auto-negotiation is enabled, attempting to set the Forced Interface Speed shall result in an error code 0x0C (Object State Conflict).

### 7.6.4.8 Interface type

The Interface Type attribute indicates the type of the physical interface. Table 67 specifies the Interface Type values. This attribute shall be stored in non-volatile memory.

**Table 67 – Interface type**

| Value | Type of interface |
|---|---|
| 0 | Unknown interface type |
| 1 | The interface is internal to the device, for example, in the case of an embedded switch |
| 2 | Twisted-pair (e.g., 10Base-T, 100Base-TX, 1000Base-T) |
| 3 | Optical fiber (e.g., 100Base-FX) |
| 4-255 | Reserved |

### 7.6.4.9 Interface state

The Interface State attribute indicates the current operational state of the interface. Table 68 specifies the Interface State values. This attribute shall be stored in volatile memory.

**Table 68 – Interface state**

| Value | Interface state |
|---|---|
| 0 | Unknown interface state |
| 1 | The interface is enabled and is ready to send and receive data |
| 2 | The interface is disabled |
| 3 | The interface is testing |
| 4-255 | Reserved |

#### 7.6.4.10 Admin state

The Admin State attribute allows administrative setting of the interface state. Table 69 specifies the Admin State values. This attribute shall be stored in non-volatile memory.

**Table 69 – Admin state**

| Value | Admin state |
|---|---|
| 0 | Reserved |
| 1 | Enable the interface |
| 2 | Disable the interface. If this is the only CPF2 communications interface, a request to disable the interface shall result in an error (status code 0x09) |
| 3-255 | Reserved |

Devices whose only communications port is a CP 2/2 port with a single Ethernet Link instance shall return general status code 0x09 (Invalid Attribute Value) if a request to disable its interface is received. Devices with multiple ports (any combination of multiple Ethernet Link instances and/or other communications ports) shall return general status code 0x10 (Device State Conflict) if performing a disable request for an interface would result in all of the device's communication ports becoming disabled.

#### 7.6.4.11 Interface label

The Interface Label attribute is a text string that describes the interface. The content of the string is vendor specific. For internal interfaces the text string should include "internal" somewhere in the string. The maximum number of characters in this string is 64. This attribute shall be stored in non-volatile memory.

### 7.6.5 Common services

#### 7.6.5.1 General

The Ethernet Link object shall support the common services as specified in Table 70.

**Table 70 – Ethernet Link object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| **0x01** | Optional | Optional | Get_Attribute_All | Returns a predefined listing of this objects attributes (see the Get_Attribute_All response definition in 7.6.5.2) |
| **0x0E** | Conditional | Required | Get_Attribute_Single | Returns the contents of the specified attribute |
| **0x10** | N/A | Conditional | Set_Attribute_Single | Modifies a single attribute |

The Get_Attribute_Single service shall be implemented for the class if any class attribute is implemented.

The Set_Attribute_Single service shall be implemented if the interface control or admin state attributes are implemented.

#### 7.6.5.2 Get_Attribute_All response

At the class level, the Get_Attribute_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

For instance attributes, attributes shall be returned in numerical order, up to the last implemented attribute. The Get_Attribute_All reply shall be as specified in Table 71.

**Table 71 – Get_Attribute_All reply format**

| Attribute ID | Size (octets) | Contents |
|---|---|---|
| **1** | 4 | Interface Speed |
| **2** | 4 | Interface Flags |
| **3** | 6 | Physical Address |
| **4** | 44 | Interface Counters<br>Default value of 0 if not implemented |
| **5** | 48 | Media Counters<br>Default value of 0 if not implemented |
| **6** | 4 | Interface Control<br>Default value of 0 if not implemented<br>A value of 0 (which would literally indicate Auto-negotiate disabled, half-duplex, but speed of 0) shall indicate that the attribute is not implemented. |
| **7** | 1 | Interface Type<br>Default value of 0 if not implemented. |
| **8** | 1 | Interface State<br>Default value of 0 if not implemented. |
| **9** | 1 | Admin State<br>Default value of 0 if not implemented. |
| **10** | 1 | Interface Label Length |
| | Variable, equal to Interface Label Length | Interface Label |

The length of the Interface Label is not known before issuing the Get_Attribute_All service request. Implementers shall be prepared to accept a response containing the maximum size of the Interface Label (65 USINT's).

### 7.6.6 Class specific services

#### 7.6.6.1 General

The Ethernet Link object shall support the class specific services as specified in Table 72.

**Table 72 – Ethernet Link object class specific services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| **0x4C** | N/A | Conditional | Get_and_Clear | Gets then clears the specified attribute (interface counters or media counters) |

The Get_and_Clear service shall only be implemented if the interface counters and media counters attributes are implemented.

#### 7.6.6.2 Get_and_Clear service

The Get_and_Clear service is a class specific service. It is only supported for the interface counters and media counters attributes. The Get_and_Clear response shall be the same as the Get_Attribute_Single response for the specified attribute. After the response is built, the value of the attribute shall be set to zero.

### 7.6.7 Behavior

The behavior of the TCP/IP Interface object shall be as illustrated in the State Transition Diagram shown in Figure 24.

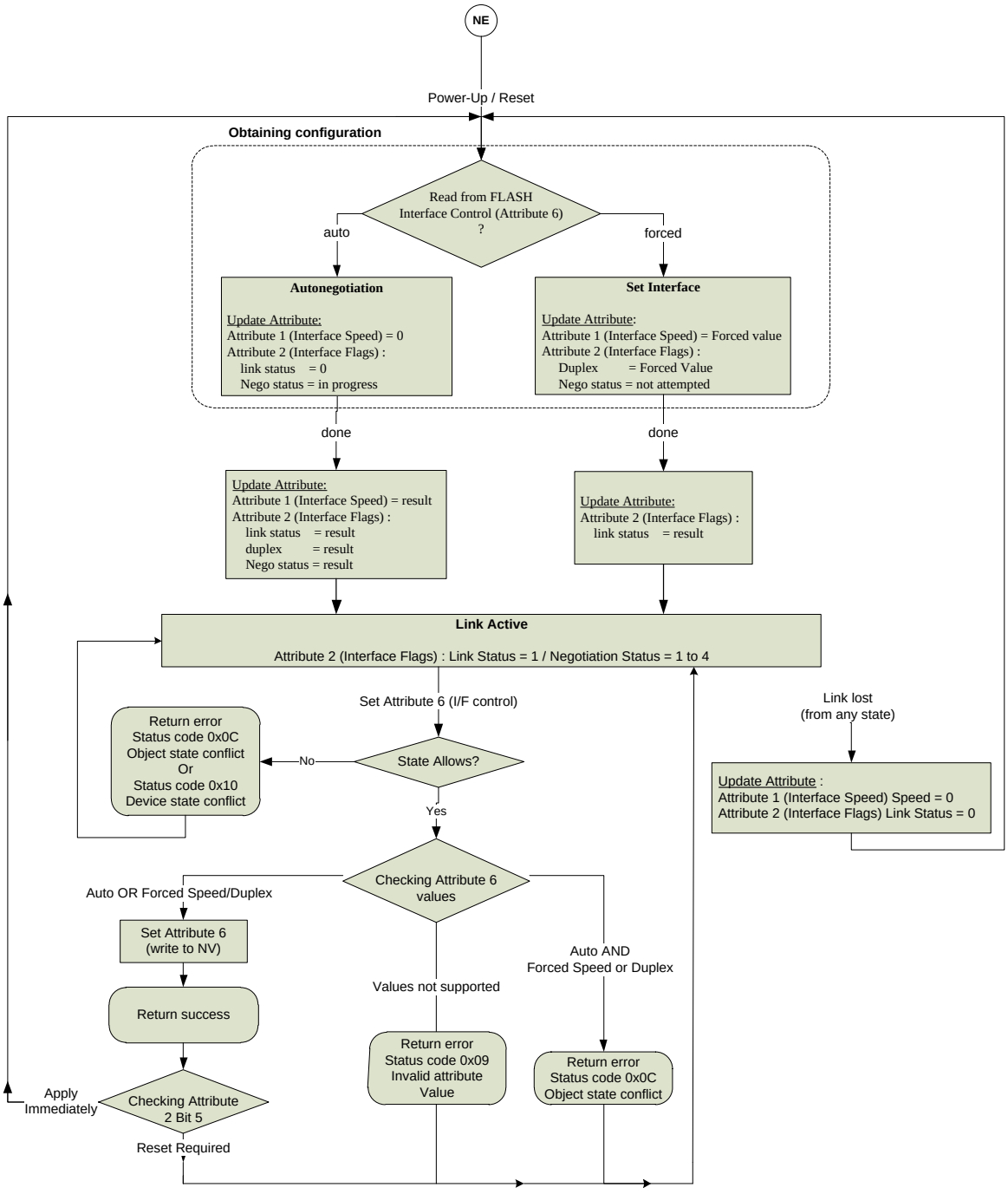



**Figure 24 – State transition diagram for Ethernet Link object**

## 7.7 DeviceNet object

### 7.7.1 Overview

The DeviceNet object shall provide a consistent Station Management interface to the Physical and Data Link Layers. This object shall make diagnostic information from these layers available to client applications. Each node shall support one DeviceNet object per link.

A device may contain more than one node.

### 7.7.2 Revision history

The revision history of the DeviceNet object is described in Table 73.

**Table 73 – DeviceNet object revision history**

| Class revision | Description of changes |
|---|---|
| **01** | Initial Release |
| **02** | Release for IEC 61158 series |

### 7.7.3 Class attributes

The DeviceNet object shall support the class attributes as specified in Table 74.

**Table 74 – DeviceNet object class attributes**

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x01** | Required | Get | NV | Revision | UINT | revision of this object | 2 (revision 2) |

### 7.7.4 Instance attributes

#### 7.7.4.1 General

The DeviceNet object shall support the instance attributes as specified in Table 75.

**Table 75 – DeviceNet object instance attributes**

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x01** | Optional | Get/Set | NV | MAC ID | USINT | node address | see 7.7.4.2 |
| **0x02** | Optional | Get/Set | NV | Bit Rate | USINT | bit rate | see 7.7.4.3 |
| **0x03** | Optional | Get/Set | NV | BOI | BOOL | bus-off interrupt | see 7.7.4.4 |
| **0x04** | Optional | Get/Set | V | Bus-Off Counter | USINT | number of times CAN went to the bus-off state | see 7.7.4.5 |
| **0x05** | Optional | Get | V | Allocation Information | STRUCT of 2 octets | | see 7.7.4.6 |
| | | | | Allocation choice | SWORD | | see 7.7.4.6.2 |
| | | | | Master's MAC ID | USINT | MAC ID of master (from allocate) | see 7.7.4.6.3 |
| **0x06** | Conditional[a] | Get | V | MAC ID switch changed | BOOL | The node address switch(es) have changed since last power–up/reset | 0 = no change<br>1 = change since last reset or power–up |
| **0x07** | Conditional[b] | Get | V | Bit rate switch changed | BOOL | The bit rate switch(es) have changed since last power–up/reset | 0 = no change<br>1 = change since last reset or power–up |
| **0x08** | Conditional[a] | Get | V | MAC ID switch value | USINT | Actual value of node address switch(es) | 0 to 99 |
| **0x09** | Conditional[b] | Get | V | Bit rate switch value | USINT | Actual value of bit rate switch(es) | 0 to 9 |

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x0A** | Optional[c] | Get/Set | NV | QuickConnect | BOOL | Enable/Disable of QuickConnect feature | 0 = disable (default)<br>1 = enable |
| **0x0B** | Conditional[d] | | | Safety Network Number | SWORD[6] | | See IEC 61784-3-2 |
| **0x0C** | Optional | Get | V | Diagnostic counters | STRUCT of 34 octets | List of ISO 11898 diagnostic counters | See 7.7.4.7 |
| | | | | Diagnostic counters descriptor | WORD | Indicates which diagnostic counters are supported | |
| | | | | Arbitration loss count | UINT | See ISO 11898 | Range = 0 to 65 535<br>Default = 0 |
| | | | | Overload count | UINT | See ISO 11898 | |
| | | | | Bit error count | UINT | See ISO 11898 | |
| | | | | Stuff error count | UINT | See ISO 11898 | |
| | | | | Ack error count | UINT | See ISO 11898 | |
| | | | | Form error count | UINT | See ISO 11898 | |
| | | | | CRC error count | UINT | See ISO 11898 | |
| | | | | Rx message loss count | UINT | ISO 11898 subsystem has detected a lost receive message | |
| | | | | Warning error count | UINT | See ISO 11898 | |
| | | | | Rx error counter | UINT | Receive error counter (see ISO 11898) | Range = 0 to 256 |
| | | | | Tx error counter | UINT | Transmit error counter (see ISO 11898) | Range = 0 to 256 |
| | | | | | UINT[5] | Reserved | Default = 0 |
| **0x0D** | Conditional[e] | Get | V | Active Node Table | BOOL[64] | Identifies which nodes are online on the local network, based on Node Address | One bit for each node number. The node number is the index into the bit array.<br>0 = Inactive<br>1 = Active |

a These attributes are required if the MAC ID switch can be set to a different value than the device is currently on-line at.

b These attributes are required if the bit rate switch can be set to a different value than the device is currently on-line at.

c This attribute shall be stored in non-volatile memory.

d This attribute is required for CPF 2 safety devices. Non-safety devices shall not implement this attribute.

e This attribute is required if the device supports routing between two or more CPF2 ports.

#### 7.7.4.2 MAC ID

This attribute contains the MAC ID of this device. The range of values is 0 to 63 decimal.

A device that uses a switch to set the MAC ID shall return an Error Response, with General Error Code 0x0E (Attribute not settable), in response to a Set_Attribute_Single request when the switch setting is 0 to 63 and the MAC ID switch is enabled by other settings. When the MAC ID switch setting is greater than 63 (disabled) or the MAC ID switch is disabled by other

settings, the device may support the setting of the MAC ID attribute by the Set_Attribute_Single service request. The device shall provide visual indication (e.g. via physical switch, LED) that the MAC ID switch has been overridden. A minor recoverable fault shall be declared when the MAC ID switch is enabled and indicates a valid MAC ID, but does not match the current on-line address of the device. During power up or reset, a device should go to the Communication Fault state if it has a MAC ID switch set to an invalid setting and does not support the setting of the MAC ID attribute.

The MAC ID attribute is considered non-volatile in that once configured the attribute shall be remembered after a power cycle or device reset. The “out-of-box” configuration shall default to a MAC ID value of 63. The following MAC ID determination scenarii apply only to devices that use non-volatile memory to store the MAC ID of the device.

1. When the MAC ID switch is valid and enabled, the switch value is loaded into non-volatile memory when the device powers up or is reset, and prior to attempt to go on-line. The MAC ID switch value is not loaded into non-volatile memory at any other time.
2. Devices shall attempt to go on-line with the MAC ID value stored in non-volatile memory or, if in the “out-of-box” configuration, with a MAC ID value of 63.
3. When a Set_Attribute_Single request with a valid MAC ID is received by a device with the MAC ID switch disabled, the non-volatile MAC ID shall be loaded with the new MAC ID.
4. When a Set_Attribute_Single request with a valid MAC ID is received by a device with the MAC ID switch enabled, the non-volatile MAC ID shall not change and an Error Response with General Error Code 0x0E (Attribute not settable) shall be returned.

The modification of the MAC ID requires a device to delete all Connection objects and re-execute the Link Access State Machine defined in IEC 62026-3:2008, 5.4.

#### 7.7.4.3 Bit Rate

The Bit Rate attribute indicates the selected bit rate. Values are specified in Table 76.

**Table 76 – Bit rate attribute values**

| Value | Meaning |
|---|---|
| 00 | 125 kbit/s |
| 01 | 250 kbit/s |
| 02 | 500 kbit/s |

A device that uses a switch to set the bit rate shall return an Error Response, with General Error Code 0x0E (Attribute not settable), in response to a Set_Attribute_Single request when the bit rate switch is set to a valid value and not disabled by other settings. When the bit rate switch is not set to a valid value (disabled) and/or the bit rate switch is disabled by other settings, the device may support the setting of the Bit Rate attribute by the Set_Attribute_Single service request. The device shall provide visual indication (via physical switch, LED, etc.) that the bit rate switch has been overridden. A minor recoverable fault shall be declared when the bit rate switch is enabled and indicates a valid bit rate, but does not match the current on-line bit rate of the device. During power up or reset, a device should go to the Communication Fault state if it has a bit rate switch set to an invalid setting and does not support the setting of the Bit Rate attribute.

The Bit Rate attribute is considered non-volatile in that once configured the attribute shall be remembered after a power cycle or device reset. The “out-of-box” configuration shall default such that the device is able to go online at 125 kbit/s. The following bit rate determination scenarii apply only to devices that use non-volatile memory to store the bit rate of the device.

1. When the bit rate switch is valid and enabled, the switch value is loaded into non-volatile memory when the device powers up or is reset, and prior to attempting to go online. The bit rate switch value is not loaded into non-volatile memory at any other time.
2. Devices shall attempt to go on-line with the bit rate value stored in non-volatile memory or, if in the “out-of-box” configuration, able to go online at 125 kbit/s.
3. When a Set_Attribute_Single request with a valid bit rate is received by a device with the bit rate switch disabled, the non-volatile Bit Rate shall be loaded with the new bit rate.
4. When a Set_Attribute_Single request with a valid bit rate is received by a device with the bit rate switch enabled, the non-volatile Bit Rate shall not change, and an Error Response with General Error Code 0x0E (Attribute not settable) shall be returned.

The modification of the Bit Rate will not take effect until the device is either physically reset (such as cycle of power or a reset switch) or reset by sending the Reset Service to the Identity object. During this time the Bit Rate attribute value will not match the actual network bit rate.

#### 7.7.4.4 BOI (Bus–off interrupt)

The BOI attribute consists of one bit that defines how a CAN device processes the bus–off interrupt. The BOI attribute is bit position 0 within an octet for the Get_Attribute_Single/ Set_Attribute_Single services. The rest of the bits in the octet shall be zeros.

**Table 77 – BOI attribute values**

| Value | Meaning |
|---|---|
| 00 | Hold the CAN chip in its bus-off (reset) state upon detection of a bus-off indication |
| 01 | If possible, fully reset the CAN chip and continue communicating upon detection of a bus-off indication |

When the BOI attribute is FALSE (set to zero) and an ISO 11898 CAN chip bus-off event is detected, the following steps are taken:

- the CAN chip is held in its reset/bus-off state;
- the device enters the Communication Fault state (see IEC 62026-3:2008, 5.4).

When the BOI attribute is TRUE (set to one) and a CAN chip bus-off event is detected, it may be possible to return the CAN chip to its normal operating mode and continue communicating based on the Link Access State Machine presented in IEC 62026-3:2008, 5.4.

If the BOI attribute is set to one the device shall insure that it does not perpetually reset and continue to produce corrupted packets on the bus. Failing to do so will allow the device to disrupt all communications on the bus.

Connections are not necessarily effected when a bus-off event is detected and the device is able to continue communicating. Previously established connections can remain existing or they can be deleted (soft reset). Either way, the Duplicate MAC ID detection algorithm shall be performed again per the Link Access State Machine.

As indicated in Table 75, support of the BOI attribute is optional. If it is not supported, a device shall implement the behavior indicated by attribute value zero (0) in Table 77.

#### 7.7.4.5 Bus–off counter

The Bus–off Counter counts the number of times the CAN chip went to the bus–off state (counts number of bus–off interrupts). The counter has values of 0 to 255 decimal.

The Bus–off Counter is initialized to zero at power–up or device initialization.

The Bus–off Counter stops counting when it reaches maximum count. The counter does not roll over. The Counter stays at maximum count until a Set_Attribute_Single is performed.

The DeviceNet object resets the Bus–off Counter to zero (0) whenever it receives a Set_Attribute_Single request specifying the Bus-Off Counter attribute. The Set_Attribute_Single data is not used and can be any value. The transmission of a Set_Attribute_Single request to the Bus–off Counter is all that's required to reset the counter.

#### 7.7.4.6 Allocation information

##### 7.7.4.6.1 Overview

The Allocation Information attribute is pertinent to the Predefined Master/Slave Connection Set. Its support is required if the Predefined Master/Slave Connection Set is supported. It indicates whether or not the Predefined Master/Slave Connection Set defined in IEC 62026-3:2008, 5.5 has been allocated. If it has been allocated, then, this attribute indicates the device that has performed the allocation and the Connection(s) that are currently allocated.

This attribute is modified when a successful response associated with an Allocate_Master/Slave_Connection_Set service (defined later in 7.7.6) is generated. This attribute is not modifiable by the Set_Attribute_Single Service. An Error Response whose General Error Code Field is set to 0x0E (Attribute not settable) is returned if a Set_Attribute_Single request specifies this attribute.

The Allocation Information attribute consists of the Allocation choice and the Master's MAC ID.

##### 7.7.4.6.2 Allocation choice

The Allocation choice indicates which of the Predefined Master/Slave Connections are active (in the Configuring, or Established state). Its format is specified in IEC 62026-3:2008, 5.5.

The Allocation choice is initialized to 00 at device power–up or reset.

##### 7.7.4.6.3 Master's MAC ID

The Master's MAC ID contains the MAC ID of the device that has allocated the Predefined Master/Slave Connection Set via the Allocate_Master/Slave_Connection_Set service. This contains the Allocator's MAC ID field copied from the Allocate_Master/Slave_Connection_Set request.

The range of values is 0 to 63 and 255 decimal. A value in the range 0 to 63 indicates that the Predefined Master/Slave Connection Set is currently allocated and denotes the MAC ID of the device that performed the allocation. The value 255 means the Predefined Master/Slave Connection set has not been allocated.

The Master's MAC ID attribute is initialized to 255 (0xFF) at device power–up/reset.

#### 7.7.4.7 Diagnostic counters

This attribute reports the counts of various types of network errors. The first field, diagnostic counters descriptor, identifies which counters are supported by the device. Each bit indicates if a designated counter is supported. If set, the counter is supported. Table 78 specifies the diagnostic counter bit assignment.

**Table 78 – Diagnostic counters bit description**

| Bit | Counter | Clearable |
|---|---|---|
| 0 | Arbitration loss | Yes |
| 1 | Overload error | Yes |
| 2 | Bit error | Yes |
| 3 | Stuff error | Yes |
| 4 | Ack error | Yes |
| 5 | Form error | Yes |
| 6 | CRC error | Yes |
| 7 | Rx message loss | Yes |
| 8 | Warning error | Yes |
| 9 | Rx error | No |
| 10 | Tx error | No |
| 11 – 15 | Reserved (shall be 0) | N/A |

The remaining fields provide counter values for each supported counter. The counters are advanced by one fore each occurrence of the error, except for the Rx and Tx error counters (fields 10 and 11) which are incremented and decremented by the CAN peripheral based on the ISO 11898 standard, and do not roll over.

All counters are cleared to zero when the device enters the Sending Duplicate MAC ID Check Request Message (see the Link Access State Transition Diagram in IEC 62026-3:2008, 5.4).

#### 7.7.4.8 Active Node Table

The Active Node Table attribute reports the activity state of each node on the network, based on node numbers. Each bit in the array represents a node on the local network, with the bit position matching the node address represented (i.e. Bit 0 = Node Address 0, Bit 1 = Node Address 1). When the bit is set (TRUE) the node is active on the link. When the bit is cleared (FALSE) the node is inactive on the link.

The bit is set (node is indicated as active) when the node represented by the bit is in either the On-Line, Sending Duplicate MAC ID Check Request Message, or Waiting for Duplicate MAC ID Check Message state. The setting of the bit shall occur within 1 s of a previously inactive node transmitting on the link. Also, the device containing the Active Node Table shall set all bits when transitioning to the On-Line state.

The bit is cleared (node is indicated as inactive) when the node represented by the bit is in either the Non-Existent or Communication Fault state. The clearing of the bit shall occur within 20 s of the previously active node leaving the link (or, if the node is not present on the network, within 20 s after the device containing the Active Node Table transitions to the On-Line state). Also, the device containing the Active Node Table shall clear all bits when transitioning out of the On-Line state.

A router shall not attempt to route an unconnected message request to a node on the local network when the Active Node Table bit for that node indicates inactive (bit is cleared). Instead, the router shall immediately return a General Error status of 0x01 and an Extended Error status of 0x0204.

### 7.7.5 Common services

#### 7.7.5.1 General

The DeviceNet object shall support the common services as specified in Table 79.

**Table 79 – DeviceNet object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| **0x05** | N/A | Conditional[a] | Reset | Used to reset this instance of the DeviceNet object |
| **0x10** | N/A | Optional | Set_Attribute_Single | Set network or node specific configuration information |
| **0x0E** | Optional | Optional | Get_Attribute_Single | Get network or node specific configuration information |

[a] This service is required when this DeviceNet object instance is addressable through some other CPF 2 access mechanism other than the subnet interface controlled by this DeviceNet object instance.

#### 7.7.5.2 Reset

When the DeviceNet object receives a Reset request, it

- determines if it can provide the type of reset requested;
- responds to the request;
- performs the type of reset requested.

The Reset common service has the object–specific parameter specified in Table 80.

**Table 80 – Reset service parameter**

| Name | Type | Description of request parameters | Semantics of values |
|---|---|---|---|
| Type | USINT | Type of reset | See Table 81 |

The parameter Type for the Reset common service has the enumeration specifications shown in Table 81.

**Table 81 – Reset service parameter values**

| Value | Type of reset |
|---|---|
| 0 | Emulate as closely as possible cycling power on the network link that the DeviceNet object instance represents. This value is the default if this parameter is omitted |
| 1 | Return as closely as possible to the out-of-box configuration, then emulate cycling power on the network link as closely as possible |
| 2 | Excepting the MAC ID and bit rate, return as closely as possible to the out-of-box configuration, then emulate cycling power on the network link as closely as possible. The MAC ID and bit rate is not changed by this reset |
| 3 – 99 | Reserved |
| 100 – 199 | Vendor specific reset behavior |
| 200 – 255 | Reserved |

### 7.7.6 Class specific services

#### 7.7.6.1 General

The DeviceNet object shall support the class specific services as specified in Table 82.

**Table 82 – DeviceNet object class specific services**

| Service code | Need in implementation | Service name | Description of service |
|---|---|---|---|
| **0x4B** | Optional | Allocate_Master/Slave_ Connection_Set | Requests the use of the Predefined Master/Slave Connection Set |
| **0x4C** | Conditional[a] | Release_Master/Slave_ Connection_Set | Indicates that the specified connections within the Predefined Master/Slave Connection Set are no longer desired. These connections shall be released (deleted) |
| **0x4D** | Conditional[b] | Clear_Diagnostics | Clears the diagnostic counters that are reported in attribute 12 |

[a] The Release_Master/Slave_Connection_Set service is required if the Allocate_Master/Slave_Connection_Set service is implemented in the device.

[b] This service shall be supported if any clearable counters are implemented in the Diagnostic counters attribute.

#### 7.7.6.2 Allocate_Master/Slave_Connection_Set, Release_Master/Slave_Connection_Set

These services are used to allocate and deallocate the Predefined Master/Slave Connection Set described in IEC 62026-3:2008, 5.5.

A device that behaves as the client across the Predefined Master/Slave Connection Set is referred to as a master. A device that behaves as the server across the Predefined Master/Slave Connection Set is referred to as a slave. Within the bounds of the service descriptions to follow, a master and/or slave is viewed as a functional unit within a communicating device.

A device that wants to function as another's master shall first allocate the Predefined Master/Slave Connection Set within the slave. Only one master can have the Predefined Master/Slave Connection Set allocated at any given time. The entire Connection Set is allocated and the master uses a select subset of the connections from the set (i.e. Bit Strobe only, or Poll only). When a master wants to "give–up" its slave it releases all connections, causing the slave to "deallocate" the Predefined Master/Slave Connection Set.

#### 7.7.6.3 Clear_Diagnostics

This service clears (to zero) the clearable counters within attribute 12 (diagnostic counters). The counters that are clearable are indicated in Table 78.

### 7.8 Connection configuration object (CCO)

#### 7.8.1 Overview

This object defines an interface used to create, configure and control connections in a device. This specification does not define or constrain the operation of the device's connection management state machine..

NOTE The CCO object is a device addressable object (not a port addressable object).

### 7.8.2 Revision history

The revision history of the Connection Configuration object is described in Table 83.

**Table 83 – Connection configuration object revision history**

| Class revision | Description of changes |
|---|---|
| **01** | Initial definition |
| **02** | Intermediate definition |
| **03** | Release for IEC 61158 |

### 7.8.3 Class attributes

#### 7.8.3.1 General

The Connection Configuration object shall support the class attributes as specified in Table 84.

**Table 84 – Connection configuration object class attributes**

| Attri-bute ID | Need in implemen tation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x01** | Required | Get | NV | Revision | UINT | Revision of this object | Third revision, value = 3 (see 7.8.2) |
| **0x02** | Required | Get | NV | Max instance | UDINT | Maximum instance number | Value determined by node specifics |
| **0x03** | Required | Get | NV | Num instance | UDINT | Number of connections currently instantiated | |
| **0x04** | This class attribute is optional and is described in IEC 61158-5-2 | | | | | | |
| **0x05** | Conditional[e] | Get | NV | Optional service list | STRUCT of variable size | List of optional services utilized in an object class implementation | A list of service codes specifying the optional services implemented in the device for this class |
| | | | | Number services | UINT | Number of services in the optional service list | The number of service codes in the list |
| | | | | Optional services | ARRAY of UINT | List of optional service codes | The optional service codes |
| **0x06 0x07** | These class attributes are optional and are described in IEC 61158-5-2 | | | | | | |
| **0x08** | Required | Get | NV | Format number | UDINT | See 7.8.3.2 | |
| **0x09** | Required | Get/Set | NV | Edit signature | UDINT | See 7.8.3.3 | |
| **0x0A – 0x20** | Reserved | | | | | | |
| **0x21** | Optional | Get/ Set[a,b] | NV | Scanner Mode | BOOL | The originator to target packets for connections originated by this device shall reflect the state of this attribute | 0 = idle mode<br>1 = run mode |
| **0x22** | Optional | Get | V | Scanner capabilities[c] | WORD | Bit 0, set to Scanner Mode (attribute 0x21) supported [d]<br>Bit 1, set to Scanner Mode (attribute 0x21) | Bits 0 – 10<br>= 0, No<br>= 1, Yes |

| Attri-bute ID | Need in implemen tation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| | | | | | | currently supported<br>Bit 2, Instances may currently be created in Run mode<br>Bit 3, Instances may currently be changed in Run mode<br>Bit 4, Instances may currently be deleted in Run mode<br>Bit 5, Instance may currently be created in Idle mode<br>Bit 6, Instances may currently be changed in Idle mode<br>Bit 7, Instances may currently be deleted in Idle mode<br>Bit 8, Open_Connection/Close_Connection services supported<br>Bit 9, Stop_Connection service supported<br>Bit 10, Get_Member service to read image tables supported<br>Bits 11-15 – reserved and shall be zero | |

a A set to attribute 0x21 shall return a device state conflict (0x10) error if changing the Scanner Mode is not permitted when the set attribute command is received.

b The value of attribute 0x21may be different than the value last set to attribute 0x21 if another mechanism (like a key switch) changed the scanner's mode.

c If the device in which this object is implemented has some sort of mechanism for changing connections, the state of these bits shall be changed to reflect the present state of the device.

d Bit 0 indicates if the Scanner Mode attribute has the ability to ever be set using a set attribute service.

e Required if object supports any optional services. Optional if object does not support any optional services.

#### 7.8.3.2 Format number

This number determines the format of instance attribute 0x09. This format value shall be specified in the format_number field of each instance attribute 0x09. Meaning of the format values is specified in Table 85.

**Table 85 – Format number values**

| Value | Meaning |
|---|---|
| 0 | Single O->T/T->O tables, 16-bit words, 0-based offsets |
| 1 | Multiple O->T/T->O tables, 16-bit words, 0-based offsets |
| 2 – 99 | Reserved |
| 100 – 199 | Vendor Specific |
| 200 – … | All other values are reserved |

#### 7.8.3.3 Edit signature

Created and used by configuration software to detect modification to the instance attribute values. For CP 2/1 this value is initially 0 and is incremented each time a change complete operation is performed. For all other CPF 2 networks this value is initially 0 and is set to a 32 bit CRC each time a change complete operation is performed (the polynomial is $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+x^0$), the CRC seed value shall be 0. The CRC is calculated on the set all attribute data stream for each connection instance (lowest to highest) plus class attribute 0x01, class attribute 0x02, class attribute 0x03 and class attribute 0x08.

### 7.8.4 Instance attributes

#### 7.8.4.1 General

The Connection Configuration object shall support the instance attributes as specified in Table 86.

**Table 86 – Connection configuration object instance attributes**

| Attribute ID | Need in implem | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x01** | Required | Get | V | Connection status | STRUCT of 4 octets | Connection status.<br>Only two octets of extended status are included for error codes | See 7.8.4.2.<br>NOTE For access to complete connection status, it is recommended that attribute 0x17 (Full Connection Status) be implemented as well. |
| | | | | Gen_status | USINT | General status | The General Status value returned in the Forward_Open/ Large_Forward_Open response |
| | | | | Reserved | USINT | Reserved | Shall be zero |
| | | | | Ext_status | UINT | Extended status | If no extended status is returned, the Ext_status shall be zero |
| **0x02** | Required | Get/Set | NV | Connection flags | WORD | Connection flags | See 7.8.4.3 |
| **0x03** | Required | Get/Set | NV | Target device ID | STRUCT of 8 octets | | See 7.8.4.4 |
| | | | | Vendor_id | UINT | Vendor ID | |
| | | | | Product_type | UINT | Device type | |
| | | | | Product_code | UINT | Product code | |
| | | | | Major_rev | USINT | Major revision | |
| | | | | Minor_rev | USINT | Minor revision | |
| **0x04** | Conditional | Get/Set | NV | CS data index number | UDINT | | See 7.8.4.5 |
| **0x05** | Required | Get/Set | NV | Net connection parameters | STRUCT of 14 octets | | See 7.8.4.6 |

| Attribute ID | Need in implem | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| | | | | Conn_timeout | USINT | Connection Timeout Multiplier | |
| | | | | Xport_class_and_trig | SWORD | Transport Class and Trigger | |
| | | | | Rpi_OT | UDINT | Originator to Target Requested Packet Interval | |
| | | | | Net_OT | UINT | Originator to Target network connection parameters | |
| | | | | Rpi_TO | UDINT | Target to Originator Requested Packet Interval | |
| | | | | Net_TO | UINT | Target to Originator network connection parameters | |
| **0x06** | Required | Get/Set | NV | Connection path | STRUCT of variable size | | See 7.8.4.7 |
| | | | | Open_path_size | USINT | Open connection path size | |
| | | | | Reserved | USINT | Reserved | |
| | | | | Open_connection_ path | Padded EPATH | Connection path | |
| **0x07** | Required | Get | NV | Proxy Config data | STRUCT of variable size | | See 7.8.4.8 |
| | | | | Config_data_size | UINT | Length of config_data in octets | |
| | | | | Config_data | ARRAY of octets | Proxy Config data | |
| **0x08** | Required | Get/Set | NV | Connection Name | STRUCT of variable size | | See 7.8.4.9 |
| | | | | name_size | USINT | Number of characters in the connection name. | |
| | | | | Reserved | USINT | Reserved | Shall be zero |
| | | | | connection_name | STRING2 | User-assigned connection name encoded in UNICODE | |

| Attribute ID | Need in implem | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x09** | Required | Get/Set | NV | I/O mapping | STRUCT of variable size | | See 7.8.4.10 |
| | | | | format_number | UINT | This number determines the format of this attribute | The format value shall match the value of class attribute 0x08 |
| | | | | mapping_data_ size | UINT | Size, in octets, of the mapping_data field that follows | |
| | | | | mapping_data | ARRAY of octets | I/O mapping information associated with this instance | |
| **0x0A** | Required | Get/Set | NV | Target Config data | STRUCT of variable size | | See 7.8.4.11 |
| | | | | Config_data_size | UINT | Length of config_data in octets | |
| | | | | Config_data | ARRAY of octets | Target Config data | |
| **0x0B** | Required | Get/Set | NV | Proxy device ID | STRUCT of 8 octets | | See 7.8.4.12 |
| | | | | Vendor_id | UINT | Vendor ID | |
| | | | | Product_type | UINT | Device type | |
| | | | | Product_code | UINT | Product code | |
| | | | | Major_rev | USINT | Major revision | |
| | | | | Minor_rev | USINT | Minor revision | |
| **0x0C** | Conditional[a] | Get/Set | NV | Connection disable | BOOL | Indicates if this instance of the Connection Configuration object is disabled | 0 – This instance of the Connection Configuration object is enabled.<br>1 – This instance of the Connection Configuration object is disabled.<br>When an Open service is received this value shall be set to 0. When a Close or Stop service is received this value shall be set to 1.<br>The default value of this parameter is 0 |
| **0x0D** | Conditional[b] | | | Safety Parameters | See IEC 61784-3-2 | | |
| **0x0E** | Conditional[b] | | | Safety Connection Parameter CRC | | | |

| Attribute ID | Need in implem | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| **0x0F** | Conditional[b] | | | Safety Configuration Instance | | | |
| **0x10** | Conditional[b] | | | Safety ID Allocation | | | |
| **0x11** | Conditional[b] | | | Safety Target Connection Serial Number | | | |
| **0x12** | Optional | Get/Set | | Net Connection Parameters Attribute Selection | USINT | Selects between attribute 0x05 and attribute 0x13 | See 7.8.4.13 |
| **0x13** | Conditional[c] | Get/Set | | Large Net Connection Parameters | STRUCT of 18 octets | | See 7.8.4.14 |
| | | | | Conn_timeout | USINT | Connection Timeout Multiplier | |
| | | | | Xport_class_and_trigger | SWORD | Transport Class and Trigger | |
| | | | | Rpi_OT | UDINT | Originator to Target Requested Packet Interval | |
| | | | | Net_OT | UDINT | Originator to Target large network connection parameters | |
| | | | | Rpi_TO | UDINT | Target to Originator Requested Packet Interval | |
| | | | | Net_TO | UDINT | Target to Originator large network connection parameters | |
| **0x14** | Conditional[b] | | | Format Type | See IEC 61784-3-2 | | |
| **0x15** | Conditional[b] | | | Format Status | | | |
| **0x16** | Conditional[b] | | | Max Fault Number | | | |
| **0x17** | Optional | Get | V | Full Connection Status | STRUCT of variable size | This attribute is the full connection status (handles extended status longer than a single WORD) | |

| Attribute ID | Need in implem | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| | | | | Gen_status | USINT | General status | General Status value returned in the Forward_Open/ Large_Forward_Open response |
| | | | | Size of Ext_status | USINT | Number of WORD in Ext_status array | |
| | | | | Ext_status | ARRAY of WORD | Extended status[d] | Empty, or Extended Status value returned in the Forward_Open/ Large_Forward_Open response |

[a] Attribute 0x0C shall be supported if conditional services Open_Connection (0x4C) and Close_Connection (0x4D) are supported. Attribute 0x0C shall not be supported if conditional services Open_Connection (0x4C) and Close_Connection (0x4D) are not supported.

[b] These attributes are not allowed if the device is not a safety device. If the device is a safety device, see IEC 61784-3-2.

[c] Conditional attribute 0x13 is required if attribute 0x12 is supported, otherwise conditional attribute 0x13 is not allowed.

[d] If size_of_Ext_status is non-zero, then the first WORD of Ext_status shall be the same value as the Ext_status element in the Connection Status structure of attribute 0x01.

#### 7.8.4.2 Connection status

The values for the originator connection status are defined in Table 87.

**Table 87 – Originator connection status values**

| General Status | Extended status | Error Description |
|---|---|---|
| 0x00 | N/A | Connection is open |
| 0x01 | 0x0000 or greater | See Connection Manager service request error codes, IEC 61158-6-2 |
| 0x02 – 0x26 | 0x0000 or greater | See General Status codes, IEC 61158-6-2 |
| 0xD0 | 0x0001 | Connection is closed or stopped (via the Close or Stop services) |
| | 0x0002 | Connection open is pending |
| | 0x0003 | Connection close is pending |

The values of the target connection status are defined in Table 88.

**Table 88 – Target connection status values**

| General Status | Extended status | Error Description |
|---|---|---|
| 0x00 | 0x0000 | Connection is open |
| 0x01 | 0x0001 or greater | The number of connection to this target |
| 0xD0 | 0X0001 | Connection is closed or stopped (via the Close or Stop services) |

#### 7.8.4.3 Connection flags

The bit patterns for the connection flags are defined Table 89.

**Table 89 – Connection flags**

| Bit | Meaning |
|---|---|
| 0 | Connection<br>0 = Originator<br>1 = Target |
| 1 - 3 | O->T Real time transfer format<br>000 – 32-bit header format includes run/idle information<br>001 – Zero length data format indicates idle<br>010 – Modeless format – no run/idle notification<br>011 – Heartbeat format – no run/idle notification<br>100 – Reserved<br>101 – Safety format (see IEC 61784-3-2)<br>100 thru 111 – reserved for future use |
| 4 - 6 | T->O Real time transfer format<br>000 – 32-bit header format includes run/idle information<br>001 – Zero length data format indicates idle<br>010 – Modeless format – no run/idle notification<br>011 – Heartbeat format – no run/idle notification<br>100 – Reserved<br>101 – Safety format (see IEC 61784-3-2)<br>100 thru 111 – reserved for future use |
| 7 - 15 | Reserved |

#### 7.8.4.4 Target device ID

This attribute is the identity of the target device for this connection instance. This identity information is not used for verifying (keying) the online target device, it is used by a configuration tool to locate the correct electronic data sheet for the connection configuration described in this CCO instance. For Target instances this attribute shall be the identity of the device containing this CCO object.

#### 7.8.4.5 CS data index number

The CS Data Index Number attribute is required for a device on CP 2/1; otherwise this attribute shall not be implemented.

The CS Data Index Number is a value set by the configuration software. This is the connection_index value returned from the Scheduling object read service (see 7.4.4 for a the definition of the Scheduling object services). This attribute is ignored for target instances.

#### 7.8.4.6 Net connection parameters

##### 7.8.4.6.1 conn_timeout

The conn_timeout is the value used for the connection timeout multiplier field defined in IEC 61158-6-2 (Forward_Open request). conn_timeout is ignored for target instances.

#### 7.8.4.6.2 xport_class_and_trig

The xport_class_and_trig is the value used for the Transport Type/Trigger field defined in IEC 61158-6-2 (Forward_Open request).

The xport_class_and_trig field shall be ignored for target instances.The device containing this CCO object shall determine if the similar Transport Type/Trigger parameter of the Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.14.

#### 7.8.4.6.3 rpi_OT

The rpi_OT is the value used for the O_to_T RPI field defined in IEC 61158-6-2 (Forward_Open request). rpi_OT is ignored for target instances.

#### 7.8.4.6.4 net_OT

The net_OT is the value used for the O_to_T Network Connection Parameters field in IEC 61158-6-2 (Forward_Open request).

The net_OT field shall be ignored for target instances, with the exception of the connection size. The device containing this CCO object shall determine if the similar O->T Network Connection Parameters parameter of the Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.1.

#### 7.8.4.6.5 rpi_TO

The rpi_TO is the value used for the T_to_O RPI field defined in IEC 61158-6-2 (Forward_Open request). rpi_TO is ignored for target instances.

#### 7.8.4.6.6 net_TO

The net_TO is the value used for the T_to_O Network Connection Parameters field in IEC 61158-6-2 (Forward_Open request).

The net_TO field shall be ignored for target instances, with the exception of the connection size. The device containing this CCO object shall determine if the similar T->O Network Connection Parameters parameter of the Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.1.

### 7.8.4.7 Connection path

#### 7.8.4.7.1 open_path_size

The open_path_size is the value used for the Connection_Path_Size field in IEC 61158-6-2 (Forward_Open/Large_Forward_Open requests). For target instances the open_path_size is used for matching the Connection_Path_Size received in a Forward_Open/Large_Forward_Open request.

#### 7.8.4.7.2 open connection path

The open_connection_path is the value used for the Connection_Path field in IEC 61158-6-2 (Forward_Open/Large_Forward_Open requests). For target instances the open_connection_path is used for matching the Connection_Path parameter received in a Forward_Open/Large_Forward_Open request.

### 7.8.4.8 Proxy Config data

This does not apply to target instances. The data specified in attributes 0x07 and 0x0A are concatenated (in that order) into a single data segment, then appended to the connection path

in attribute 0x06 to form the complete path sent in the Forward_Open service of the Connection Manager object.

The configuration data may be split between attributes 0x07 and 0x0A. When a connection is a non-proxied connection the convention that shall be followed is that all configuration data is placed in attribute 0x0A (Target Config). When a connection is a proxied connection the convention that shall be followed is that any configuration data intended for the proxying device is placed in attribute 0x07 (Proxy Config) and any configuration data intended for the proxied device is placed in attribute 0x0A (Target Config). Proxy Configdata is ignored for target instances.

#### 7.8.4.9 Connection name

This Connection Name field allows a user to name each connection instance. The connection name has no meaning to the Connection Configuration object.

#### 7.8.4.10 I/O mapping

The I/O mapping attribute contains I/O mapping data. The I/O mapping data specifies image table locations where target to originator data is placed and where originator to target data is obtained.

Table 90 describes the structure of the mapping_data field of the I/O mapping attribute for each of the format numbers.

**Table 90 – I/O mapping formats**

| Format Number | Mapping_data_size (in octets) | Name | Data Type | Description |
|---|---|---|---|---|
| 0 | 4 | Single I/O tables | STRUCT of 4 octets | 16 bit words, 0 based 16 bit word offsets |
| | | O->T offset | UINT | Offset into O->T image table |
| | | T->O offset | UINT | Offset into T->O image table |
| 1 | 8 | Multiple I/O tables | STRUCT of 8 octets | Table selection, 16 bit words, 0 based 16 bit word offsets |
| | | O->T table | UINT | O->T image table selection |
| | | O->T offset | UINT | Offset into O->T image table |
| | | T->O table | UINT | T->O image table selection |
| | | T->O offset | UINT | Offset into T->O image table |

#### 7.8.4.11 Target Config data

This note does not apply to target instances. The data specified in attributes 0x07 and 0x0A are concatenated (in that order) into a single data segment, then appended to the connection path in attribute 0x06 to form the complete path sent in the Forward_Open service of the Connection Manager object. All the configuration data may be placed in attribute 0x07 or all the configuration data may be placed in attribute 0x0A or the configuration data may be split between attributes 0x07 and 0x0A. When a connection is a proxied connection the convention that shall be followed is that any configuration data intended for the proxying device is placed in attribute 0x07 and any configuration data intended for the proxied device is placed in attribute 0x0A. Target Config data is ignored for target instances.

#### 7.8.4.12 Proxy Device ID

This attribute is the identity of the device proxying for the target device for this connection instance. This identity information is not used for verifying (keying) the online target device, it is used by a configuration tool to locate the correct electronic data sheet for the connection configuration described in this CCO instance. For Target instances and Connections that are not proxied this attribute shall be ignored.

#### 7.8.4.13 Net Connection Parameters Attribute Selection

The Net Connection Parameters Attribute Selection attribute selects between the use of instance attribute 0x05 (Net Connection Parameters) and attribute 0x13 (Large Net Connection Parameters).

Valid values are:

0 = indicates instance attribute 5 shall be used (default);

1 = indicates instance attribute 19 shall be used;

2-255 = reserved.

If this optional attribute is not supported attribute 0x05 shall be used.

#### 7.8.4.14 Large Net Connection Parameters

##### 7.8.4.14.1 conn_timeout

The conn_timeout is the value used for the connection timeout multiplier field defined in IEC 61158-6-2 (Large_Forward_Open request). conn_timeout is ignored for target instances.

##### 7.8.4.14.2 xport_class_and_trig

The xport_class_and_trig is the value used for the Transport Type/Trigger field defined in IEC 61158-6-2 (Large_Forward_Open request).

The xport_class and_trig field shall be ignored for target instances. The device containing this CCO object shall determine if the similar Transport Type/Trigger parameter of the Large_Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.14.

##### 7.8.4.14.3 rpi_OT

The rpi_OT is the value used for the O->T RPI field defined in IEC 61158-6-2 (Large_Forward_Open request). rpi_OT is ignored for target instances.

##### 7.8.4.14.4 net_OT

The net_OT is the value used for the O->T Network Connection Parameters field in IEC 61158-6-2 (Large_Forward_Open request).

The net_OT field shall be ignored for target instances, with the exception of the connection size. The device containing this CCO object shall determine if the similar O->T Network Connection Parameters parameter of the Large_Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.1.

##### 7.8.4.14.5 rpi_TO

The rpi_TO is the value used for the T->O RPI field defined in IEC 61158-6-2 (Large_Forward_Open request). rpi_TO is ignored for target instances.

#### 7.8.4.14.6 net_TO

The net_TO is the value used for the T->O Network Connection Parameters field in IEC 61158-6-2 (Large_Forward_Open request).

The net_TO field shall be ignored for target instances, with the exception of the connection size. The device containing this CCO object shall determine if the T->O Network Connection Parameters parameter of the Large_Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.1.

### 7.8.5 Connection Configuration Object change control

Changes to Connection Configuration object attributes can only be made after a Change_Start service has been successfully issued. Changes to Connection Configuration object attributes are applied after a Change_Complete service has been successfully issued.

The services which are valid during a change operation are listed in Table 91. A change operation is started by issuing a Change_Start service and completed by issuing a Change_Complete service.

**Table 91 – Services valid during a change operation**

| Service code | Service name |
|---|---|
| 0x02 | Set_Attribute_All |
| 0x08 | Create |
| 0x09 | Delete |
| 0x10 | Set_Attribute_Single |
| 0x15 | Restore |
| 0x4B | Kick_Timer |
| 0x51 | Change_Complete |
| 0x52 | Audit_Changes |

### 7.8.6 Common services

#### 7.8.6.1 General

The Connection Configuration object shall support the common services as specified in Table 92.

**Table 92 – Connection configuration object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x01 | Required | Required | Get_Attribute_All | Gets all attributes of the specified instance |
| 0x02 | N/A | Required | Set_Attribute_All | Sets all attributes of the specified instance |
| 0x08 | Required | N/A | Create | Creates a new connection instance |
| 0x09 | Required | Required | Delete | Deletes an existing connection instance |
| 0x0E | Optional | Required | Get_Attribute_Single | Returns the contents of the specified attribute |

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x10 | Required | Optional | Set_Attribute_Single | Modifies an attribute value |
| 0x15 | Required | Required | Restore | Restore current connection attributes |

#### 7.8.6.2 Get_Attribute_All

The structure of the Get_Attribute_All response at the class level is shown in Table 93.

**Table 93 – Get_Attribute_All Response – class level**

| Name | Data Type | Description |
|---|---|---|
| Revision | UINT | Object revision |
| Max instance | UDINT | Maximum instance number |
| Num instance | UDINT | Number of connections currently instantiated |
| Format number | UINT | The format value that shall be specified in the format field of each instance of attribute 9 |
| Edit signature | UDINT | Value created and used by configuration software to detect modification to the instance attribute values |

The structure of the Get_Attribute_All response at the instance level is shown in Table 94.

**Table 94 – Get_Attribute_All response – instance level**

| Name | Data Type | Description |
|---|---|---|
| Connection status | STRUCT of 4 octets | Connection status information. When the connection is not open, this attribute contains an error code indicating the reason |
| | USINT | General status |
| | USINT | Pad |
| | UINT | Extended status |
| Connection flags | WORD | Connection flags |
| Target device ID | STRUCT of 8 octets | Target device identification |
| | UINT | Vendor ID |
| | UINT | Product type |
| | UINT | Product code |
| | USINT | Major revision |
| | USINT | Minor revision |
| CS data index number | UDINT | Scheduling object read data connection_index number. The default value used when this attribute is not supported shall be 0xFFFFFFFF |
| Net connection parameters | STRUCT of 14 octets | Network connection parameters for both originator-to-target and target-to-originator directions |
| | USINT | Connection time-out multiplier |
| | SWORD | Transport class and trigger |
| | UDINT | Originator-to-target RPI |
| | UINT | Originator-to-target network connection parameters |
| | UDINT | Target-to-originator RPI |
| | UINT | Target-to-originator network connection parameters |

| Name | Data Type | Description |
|---|---|---|
| Connection path | STRUCT of variable size | Connection path |
| | USINT | Size of connection path, in 16-bit words, used in the Connection Manager Forward_Open service request |
| | USINT | Reserved. Shall be zero |
| | Padded EPÄTH | Connection path. The open connection path size is the length of this array |
| Proxy Config data | STRUCT of variable size | Proxy Config data. This data is sent to devices via the Connection Manager Forward_Open/Large_Forward_Open requests |
| | UINT | Configuration data length in octets |
| | ARRAY of USINT | Configuration data padded to an even number of octets |
| Target Config data | STRUCT of variable size | Target Config data. This data is sent to devices via the Connection Manager Forward_Open/Large_Forward_Open requests |
| | UINT | Configuration data length in octets |
| | ARRAY of USINT | Module configuration data padded to even number of octets |
| Connection name | STRUCT of variable size | Connection name |
| | USINT | Number of characters in connection name |
| | USINT | Pad |
| | STRING2 | Connection name encoded in UNICODE |
| I/O mapping | STRUCT of variable size | I/O mapping |
| | UINT | Format number |
| | UINT | Attribute size in octets |
| | ARRAY of USINT | Attribute data padded to an even number of octets |
| Proxy device ID | STRUCT of 8 octets | Proxy device identification |
| | UINT | Vendor ID |
| | UINT | Product type |
| | UINT | Product code |
| | USINT | Major revision |
| | USINT | Minor revision |
| Safety parameters | STRUCT of 55 octets | See IEC 61784-3-2.<br>These 55 octets are included in the Get_Attribute_All response only if either the O=>T or T=>O real time transfer format in the Connection Flags attribute (attribute 0x02) indicates the format is Safety |
| Connection disable | BOOL | Bit 0, value of connection disable attribute (attribute 0x0C) if supported. 0, if connection disable attribute is not supported |
| Net Connection Parameters Attribute Selection | USINT | Selection of Net Connection Parameters attribute or Large Net Connection Parameters attribute if supported. 0 if Net Connection Parameters Attribute Selection is not supported |
| Large Net Connection Parameters | STRUCT of 18 octets | Network connection parameters for both O=>T and T=>O |
| | USINT | Connection time-out multiplier |
| | SWORD | Transport class and trigger |
| | UDINT | O->T RPI |

| Name | Data Type | Description |
|---|---|---|
| | UDINT | O->T network connection parameters |
| | UDINT | T->O RPI |
| | UDINT | T->O network connection parameters |
| Additional safety parameters | STRUCT of variable length | See IEC 61784-3-2.<br>This group of parameters may be included in the Get_Attribute_All response only if either the O=>T or T=>O real time transfer format in the Connection Flags attribute (attribute 0x02) indicates the format is Safety |

#### 7.8.6.3 Set_Attribute_All

This service shall set all attributes associated with an existing instance and shall support the error codes shown in Table 95, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

**Table 95 – Set_Attribute_All error codes**

| General Status | Extended status | Error name | Description |
|---|---|---|---|
| 0x0C | None | Object state conflict | The Connection Configuration Object cannot accept the Set_Attribute_Single service when an edit session is not active |

The structure of the Set_Attribute_All request is shown in Table 96.

**Table 96 – Set_Attribute_All request**

| Name | Data Type | Description |
|---|---|---|
| Connection flags | WORD | Connection flags |
| Target device id | STRUCT of 8 octets | Target device identification |
| | UINT | Vendor ID |
| | UINT | Product type |
| | UINT | Product code |
| | USINT | Major revision |
| | USINT | Minor revision |
| Cs data index number | UDINT | Scheduling object read data connection_index number. The default value used when this attribute is not supported shall be 0xFFFFFFFF |
| Net connection parameters | STRUCT of 14 octets | Network connection parameters for both O⇒T and T⇒O directions |
| | USINT | Connection time-out multiplier |
| | SWORD | Transport class and trigger |
| | UDINT | O⇒T RPI |
| | UINT | O⇒T network connection parameters |
| | UDINT | T⇒O RPI |
| | UINT | T⇒O network connection parameters |
| Connection path | STRUCT of variable size | Connection path |
| | USINT | Size of connection path, in 16-bit words, used in the Connection Manager Forward_Open service request |

| Name | Data Type | Description |
|---|---|---|
| | USINT | Reserved. Shall be zero |
| | Padded EPATH | Connection path. The open connection path size is the length of this array |
| Proxy Config data | STRUCT of variable size | Proxy Config data. This data is sent to devices via the Connection Manager Forward_Open request |
| | UINT | Module configuration data length in octets |
| | ARRAY of USINT | Module configuration data padded to even number of octets |
| Target Config data | STRUCT of variable size | Target Config data. This data is sent to devices via the Connection Manager Forward_Open request |
| | UINT | Module configuration data length in octets |
| | ARRAY of USINT | Module configuration data padded to even number of octets |
| Connection name | STRUCT of variable size | Connection name |
| | USINT | Number of characters in connection name |
| | USINT | Pad |
| | STRING2 | Connection name encoded in UNICODE |
| I/O mapping | STRUCT of variable size | I/O mapping |
| | UINT | Format number |
| | UINT | Attribute size in octets |
| | ARRAY of USINT | Attribute data padded to an even number of octets |
| Proxy device id | STRUCT of 8 octets | Proxy Device identification |
| | UINT | Vendor Id |
| | UINT | Product type |
| | UINT | Product code |
| | USINT | Major revision |
| | USINT | Minor revision |
| Safety parameters | STRUCT of 49 octets | See IEC 61784-3-2.<br>These 49 octets are included in the Set_Attribute_All service data only if either the O=>T or T=>O real time transfer format in the Connection Flags attribute (attribute 0x02) indicates the format is Safety |
| Connection disable | BOOL | Bit 0, value of connection disable attribute (attribute 12) |
| Net Connection Parameters Attribute Selection | USINT | Selection of Net Connection Parameters attribute or Large Net Connection Parameters attribute if supported. |
| Large Net Connection Parameters | STRUCT of 18 octets | Network connection parameters for both O=>T and T=>O |
| | USINT | Connection time-out multiplier |
| | SWORD | Transport class and trigger |

| Name | Data Type | Description |
|---|---|---|
| | UDINT | O->T RPI |
| | UDINT | O->T network connection parameters |
| | UDINT | T->O RPI |
| | UDINT | T->O network connection parameters |
| Additional safety parameters | STRUCT of variable length | See IEC 61784-3-2.<br>This group of parameters may be included in the Set_Attribute_All response only if either the O=>T or T=>O real time transfer format in the Connection Flags attribute (attribute 0x02) indicates the format is Safety |

Data for unsupported attributes shall be accepted if the default values are sent. The service shall be rejected if non-default values are sent for unsupported attributes.

#### 7.8.6.4 Create

This service creates a new instance of a Connection Configuration object. Initial attribute values may also be specified with this service. The created instance number is assigned by the class and returned to the requestor. Table 97 defines the request parameters for this service.

**Table 97 – Create request parameters**

| Parameter | Data Type | Description |
|---|---|---|
| Connection flags | WORD | Connection flags |
| NumConnParams | UINT | Number of attribute/value pairs that follow |
| ConnParams | ARRAY of | List of Attribute number/value pairs |
| | STRUCT of variable size | |
| | UINT | Attribute number |
| | Object/class attribute specific STRUCT | Attribute value |

Data for unsupported attributes shall be accepted if the default values are sent. The service shall be rejected if non-default values are sent for unsupported attributes.

This service shall read all attributes associated with an existing instance and shall support error codes shown in Table 98, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

**Table 98 – Create error codes**

| General Status | Extended status | Error name | Description |
|---|---|---|---|
| 0x02 | 0x0001 | Resource unavailable | The maximum number of instances already exist |
| | 0x0002 | Resource unavailable | Not enough memory on device |
| 0x03 | None | Invalid parameter value | The attribute count is invalid |
| 0x08 | None | Service not supported | Unimplemented service |
| 0x0C | None | Object state conflict | The Connection Configuraton object cannot execute the Set_Attribute_Single service when an edit session is not active |
| 0x0E | None | Attribute not settable | Attempt to set a read-only attribute |
| 0x13 | None | Not enough data | The request was too short or truncated |
| 0x1C | None | Missing attribute list entry data | The required attribute was missing |

#### 7.8.6.5 Delete

This service deletes existing connection instances. If addressed to the class-level, all connection instances are deleted. If addressed to the instance-level, only the addressed instance is deleted. This service shall support the error codes shown in Table 99, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

**Table 99 – Delete error codes**

| General Status | Extended status | Error name | Description |
|---|---|---|---|
| 0x0C | None | Object state conflict | The Connection Configuraton object cannot execute the Delete service when an edit session is not active |

#### 7.8.6.6 Restore

This service shall discard modifications to instances that have not yet been committed by the Change_Complete service. If the Restore service is addressed to the class (instance 0), then pending modifications for all instances shall be discarded and the edit session shall be ended. If addressed to a specific instance, only modifications for that instance shall be discarded and the edit session shall not be ended.

This service shall support the error codes shown in Table 100, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

**Table 100 – Restore error codes**

| General Status | Extended status | Error name | Description |
|---|---|---|---|
| 0x0C | None | Object state conflict | The Connection Configuraton object cannot execute the Restore service when an edit session is not active |

### 7.8.7 Class specific services

#### 7.8.7.1 General

The Connection Configuration object shall support the class specific services as specified in Table 101.

**Table 101 – Connection configuration object class specific services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x4B | Required | N/A | Kick_Timer | Kicks Edit Watchdog Timer |
| 0x4C | Conditional[a] | Conditional[a] | Open_Connection | Opens connections |
| 0x4D | Conditional[a] | Conditional[a] | Close_Connection | Closes connections |
| 0x4E | Conditional[a] | Conditional[a] | Stop_Connection | Stops connections |
| 0x4F | Required | N/A | Change_Start | Manages session editing |
| 0x50 | Required | N/A | Get_Status | Get status for multiple connections |
| 0x51 | Required | N/A | Change_Complete | Completes session editing |

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x52 | Required | N/A | Audit_Changes | Audits pending changes |
| [a] The Open_Connection and Close_Connection services shall either both be supported or neither the Open_Connection or Close_Connection services shall be supported. The Stop_Connection service may only be supported if the Open_Connection service is supported | | | | |

#### 7.8.7.2 Kick_Timer

This service shall reinitialize the edit watchdog timer. Upon successful execution of the Change_Start service, the edit watchdog timer shall be started with a period of 60 s. This timer is used to recover from the loss of a configuration client between the Change_Start and Change_Complete/Restore operations. Receipt of any service request shall reset the edit watchdog timer. Clients may request this service to reset the timer without otherwise affecting the state of the Connection Configuration object. If the edit watchdog timer expires, all pending modifications shall be discarded and the edit session shall be ended.

#### 7.8.7.3 Open_Connection

The Open_Connection service shall cause the connection associated with an instance of the Connection Configuration object to open. If the Open_Connection service is addressed to the class (instance 0), then all connection instances shall be opened. If addressed to a specific instance, then only that connection instance shall be opened.

#### 7.8.7.4 Close_Connection

The Close_Connection service shall cause the connection associated with an instance of the Connection Configuration object to close. If the Close_Connection service is addressed to the class (instance 0), then all connection instances shall be closed. If addressed to the instance level, then only the specified instance shall be closed. The Close_Connection service shall initiate a “graceful” connection shutdown, that is, a Forward_Close request shall be sent to the connection target. Once a connection has been closed by this service it shall remain closed until an Open_Connection service is issued.

#### 7.8.7.5 Stop_Connection

The Stop_Connection service shall cause the connection associated with an instance of the Connection Configuration object to stop producing data immediately without sending an Forward_Close request to the connection target. If the Stop_Connection service is addressed to the class (instance 0), then all connection instances shall be stopped. If addressed to the instance level, then the specified instance shall be stopped. Once a connection has been stopped, it remains stopped until a subsequent Open_Connection service request is issued.

#### 7.8.7.6 Change_Start

This service shall

a) signal the beginning of an edit session;

b) synchronize the current and pending attributes;

c) place all connections in the "changeable" state;

d) start the edit watchdog timer.

Change_Start shall be requested prior to performing any services that modify the attributes of a connection. This service shall only be addressed to the class-level (instance 0). If a Change_Start service is received while an edit session is active, an Object State Conflict error (0x0C) shall be returned.

This service shall support the error codes shown in Table 102, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

**Table 102 – Change_Start error codes**

| General Status | Extended status | Error name | Description |
|---|---|---|---|
| 0x0C | None | Object state conflict | The Connection Configuraton object cannot execute the Restore service when an edit session is not active |
| 0x10 | None | Device state conflict | The device is in a state that prevents starting an edit session |

### 7.8.7.7 Get_Status

The Get_Status service shall retrieve the status attribute (attribute 1) for multiple connections via a single transaction. This service shall be supported at the class-level (instance 0) only. Given a starting instance number, the Get_Status service shall return instance/status pairs until either the response buffer is full or the status of all connections has been returned.

The request parameter are defined in Table 103.

**Table 103 – Get_Status service parameter**

| Parameter | Data Type | Description |
|---|---|---|
| Starting Instance | UDINT | Starting instance number |

The response parameters are defined for this service in Table 104.

**Table 104 – Get_Status service response**

| Parameter | Data Type | Description |
|---|---|---|
| Done Indicator | UINT | 0 = More status to be retrieved<br>1 = All connection status information has been retrieved.<br>All other values reserved. |
| NumStatusEntries | UINT | Number of instance/status pairs that follow |
| StatusEntries | ARRAY of | List of instance/status pairs |
| | STRUCT of 8 octets | |
| | UDINT | Connection Configuration instance number |
| | USINT | General status |
| | USINT | Reserved, shall be 0 |
| | UINT | Extended status |

This service shall support the error codes shown in Table 105, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

**Table 105 – Get_Status service error codes**

| General Status | Extended status | Error name | Description |
|---|---|---|---|
| 0x03 | None | Invalid Parameter Value | The attribute count is invalid |

### 7.8.7.8 Change_Complete

This service shall signal the completion of an edit session. Pending attributes for all modified connection instances shall be applied. This service shall take a parameter indicating the type of change being performed; either full or incremental. If an incremental edit is specified, then only the connections that have been modified shall be broken and re-established. If a full edit is specified, then all connections shall be broken and all supporting resources shall be freed and reallocated before attempting to re-establish the connections. The Change_Complete service shall be supported at the class-level (instance 0) only.

If optional instance attribute 0x0C is supported and the value of instance attribute 0x0C is FALSE or if optional instance attribute 0x0C is not supported an attempt to open the connection shall be made. If optional instance attribute 0x0C is supported and the value of instance attribute 0x0C is TRUE, no attempt shall be made to open the connection.

The request parameter is defined in Table 106.

**Table 106 – Change_Complete service parameter**

| Parameter | Data Type | Description |
|---|---|---|
| Change type | UINT | 0 = Full<br>1 = Incremental<br>All other values reserved. |

This service shall support the error codes shown in Table 107, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

**Table 107 – Change_Complete service error codes**

| General Status | Extended status | Error name | Description |
|---|---|---|---|
| 0x02 | None | Resource unavailable | Indicates that there is not enough memory on the host device to support the specified configuration. |
| 0x0C | None | Object state conflict | An edit session is not active |
| 0x10 | None | Device state conflict | The device is in a state that prevents completing an edition session |

### 7.8.7.9 Audit_Changes

This service shall verify whether or not there is enough memory on the host device to support a proposed configuration. Like the Change_Complete service, this service shall take a parameter indicating the type of change being performed; either full or incremental. The Audit_Changes service shall be supported at the class-level (instance 0) only. This service allows a configuration client to determine if all pending changes will be successful before actually committing the changes with the Change_Complete service.

The request parameter is defined in Table 108.

**Table 108 – Audit_Changes service parameter**

| Parameter | Data Type | Description |
|---|---|---|
| Change type | UINT | 0 = Full<br>1 = Incremental<br>All other values reserved |

This service shall support the error codes shown in Table 109, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

**Table 109 – Audit_Changes service error codes**

| General Status | Extended status | Error name | Description |
|---|---|---|---|
| 0x02 | None | Resource unavailable | Indicates that there is not enough memory on the host device to support the specified configuration. |
| 0x0C | None | Object state conflict | An edit session is not active |
| 0x10 | None | Device state conflict | The device is in a state that prevents completing an audit |

### 7.8.8 Behavior

Figure 25 summarizes the process of creating, editing, and deleting connections.

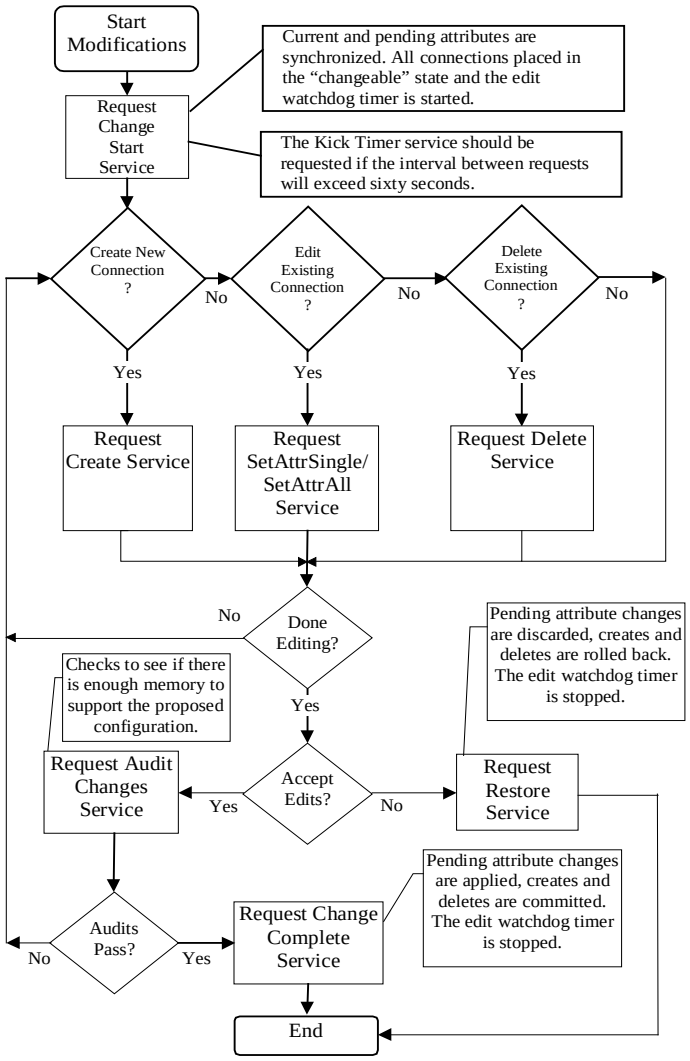



**Figure 25 – Connection configuration object edit flowchart**

## 7.9 DLR object

### 7.9.1 Overview

The Device Level Ring (DLR) object provides the CPF 2 application-level configuration and status information interface for the DLR protocol. The DLR protocol is fully specified in Clause 10.

The DLR object shall be implemented in all multi-port CP 2/2 devices that support the DLR protocol.

Devices shall implement no more than one instance of the DLR object.

NOTE Support for the DLR protocol on multiple pairs of ports is future enhancement that is at the present time undefined.

### 7.9.2 Revision history

Table 110 shows the revision history for the DLR object.

**Table 110 – Revision history**

| Revision | Description |
|---|---|
| 1 | Initial revision of this object definition |
| 2 | Instance attribute 10 changed to required, instance attribute 12 added, and corresponding changes in Get_Attribute_All response<br>Added Restart_Sign_On object specific service |
| 3 | Added conditional attribute 13 for redundant gateway configuration<br>Added conditional attribute 14 for redundant gateway status<br>Added conditional attribute 15 for active gateway address<br>Added conditional attribute 16 for active gateway Precedence<br>Added redundant gateway capability bit in attribute 12, capability flags<br>Added Flush_Tables frame support capability bit in attribute 12, capability flags<br>Added Get_Attribute_All responses for redundant gateway devices<br>Revision 2 is obsolete |

### 7.9.3 Class attributes

The DLR Object shall support the class attributes as specified in Table 111.

**Table 111 – DLR object class attributes**

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| 0x01 | Conditional[a] | Get | NV | Revision | UINT | Revision of this object | Second revision, value = 2 |
| 0x01 to 0x07 | These class attributes are optional and are described in IEC 61158-5-2 | | | | | | |

[a] Required if the Revision value is greater than 1.

### 7.9.4 Instance attributes

#### 7.9.4.1 General

The DLR object shall support the instance attributes as specified in Table 112.

**Table 112 – DLR object instance attributes**

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| 0x01 | Required | Get | V | Network Topology | USINT | Current network topology mode | 0 – Linear”<br>1 – Ring |
| 0x02 | Required | Get | V | Network Status | USINT | Current status of network | 0 – Normal<br>1 – Ring Fault<br>2 – Unexpected loop detected<br>3 – Partial network fault<br>4 – Rapid fault/ restore cycle |

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| 0x03 | Conditional [a] | Get | V | Ring Supervisor Status | USINT | Ring supervisor active status flag | See 7.9.4.4 |
| 0x04 | Conditional [a] | Set | NV | Ring Supervisor Config | STRUCT of | Ring Supervisor configuration parameters | |
| | | | | Ring Supervisor Enable | BOOL | Ring supervisor enable flag | TRUE – the device is configured as a ring supervisor.<br>FALSE – the device is configured as a normal ring node<br>Default=FALSE |
| | | | | Ring Supervisor Precedence | USINT | Precedence of a ring supervisor in network with multiple ring supervisors | Numerically higher value indicates higher precedence<br>Default=0 |
| | | | | Beacon Interval | UDINT | Duration of ring beacon interval | Beacon interval in microseconds.<br>Default=400 µs |
| | | | | Beacon Timeout | UDINT | Duration of ring beacon timeout | Beacon timeout in microseconds.<br>Default=1 960 µs |
| | | | | DLR VLAN ID | UINT | VLAN ID to use in ring protocol messages | Value range is 0 to 4 094<br>Default=0 |
| 0x05 | Conditional [a] | Set | V | Ring Faults Count | UINT | Number of ring faults since power up | |
| 0x06 | Conditional [a] | Get | V | Last Active Node on Port 1 | STRUCT of | Last active node at the end of chain through port 1 of active ring supervisor during ring fault | |
| | | | | | UDINT | Device IP Address | A value of 0 indicates no IP Address has been configured for the device. Initial value shall be 0 |
| | | | | | USINT[6] | Device MAC Address | Ethernet MAC address |
| 0x07 | Conditional [a] | Get | V | Last Active Node on Port 2 | STRUCT of | Last active node at the end of chain through port 2 of active ring supervisor during ring fault | |
| | | | | | UDINT | Device IP Address | A value of 0 indicates no IP Address has been configured for the device |
| | | | | | USINT[6] | Device MAC Address | Ethernet MAC address |
| 0x08 | Conditional [a] | Get | V | Ring Protocol Participants Count | UINT | Number of devices in ring protocol participants list | |
| 0x09 | Conditional [a] | Get | V | Ring Protocol Participants List | ARRAY of | List of devices participating in ring protocol | |

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| | | | | | STRUCT of | | |
| | | | | | UDINT | Device IP Address | A value of 0 indicates no IP Address has been configured for the device |
| | | | | | USINT[6] | Device MAC Address | Ethernet MAC address |
| 0x0A | Required | Get | V | Active Supervisor Address | STRUCT of | IP and/or MAC address of the active ring supervisor | |
| | | | | | UDINT | Supervisor IP Address | A value of 0 indicates no IP Address has been configured for the device |
| | | | | | USINT[6] | Supervisor MAC Address | Ethernet MAC address |
| 0x0B | Conditional [a] | Get | V | Active Supervisor Precedence | USINT | Precedence value of the active ring supervisor | |
| 0x0C | Required | Get | NV | Capability Flags | DWORD | Describes the DLR capabilities of the device | See 7.9.4.13 |
| 0x0D | Conditional [b] | Set | NV | Redundant Gateway Config | STRUCT of | Redundant Gateway configuration parameters | |
| | | | | Redundant Gateway Enable | BOOL | Redundant gateway enable flag | TRUE indicates the device is configured as a redundant gateway.<br>FALSE indicates device is not configured as a redundant gateway.<br>Default=FALSE |
| | | | | Gateway Precedence | USINT | Precedence of a gateway in network with multiple gateways | Numerically higher value indicates higher Precedence.<br>Default=0 |
| | | | | Advertise Interval | UDINT | Duration of active gateway Advertise Interval | Advertise Interval in microseconds.<br>Default=2 000 µs |
| | | | | Advertise Timeout | UDINT | Duration of active gateway Advertise Timeout | Advertise Timeout in microseconds.<br>Default=5 000 µs |

| Attribute ID | Need in implementation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| | | | | Learning Update Enable | BOOL | Learning Update Enable flag | TRUE indicates all DLR nodes will send Learning_Update frame after gateway switchover.<br>FALSE indicates DLR nodes will not send Learning_Update frame after gateway switchover.<br>Default=TRUE |
| 0x0E | Conditional [b] | Get | V | Redundant Gateway Status | | | See 7.9.4.15 |
| 0x0F | Conditional [b] | Get | V | Active Gateway Address | STRUCT of | IP and/or MAC address of the active gateway device | |
| | | | | | UDINT | Active gateway IP Address | A value of 0 indicates no IP Address has been configured for this device |
| | | | | | USINT[6] | Active gateway MAC Address | Ethernet MAC address |
| 0x10 | Conditional [b] | Get | V | Active Gateway Precedence | USINT | Precedence value of the active gateway | |

[a] Shall be implemented for devices capable of functioning as a ring supervisor. Shall not be implemented by non-supervisor devices.

[b] Shall be implemented for devices capable of functioning as a redundant gateway. Shall not be implemented by non-redundant gateway devices.

#### 7.9.4.2 Network topology

The Network Topology attribute indicates the current network topology mode. A value of 0 indicates “Linear” topology. A value of 1 indicates “Ring” topology. The value of the attribute shall correspond to the DLR State.

When a supervisor-capable device is enabled as a ring supervisor, the Network Topology attribute shall always indicate “Ring”. When a supervisor-capable device is not enabled as a ring supervisor, or is not a supervisor-capable device, the device shall initially indicate “Linear”, then shall transition between “Ring” and “Linear” modes.

#### 7.9.4.3 Network status

The Network Status attribute provides current status of the network based the device’s view of the network, as specified in the DLR behavior in 10.5. Table 113 specifies the possible values for the Network Status.

**Table 113 – Network Status values**

| Network Status value | Description |
|---|---|
| 0 | Normal operation in both, Ring and Linear Network Topology modes |
| 1 | Ring Fault. A ring fault has been detected. Valid only when Network Topology is Ring |
| 2 | Unexpected Loop Detected. A loop has been detected in the network. Valid only when the Network Topology is Linear |
| 3 | Partial Network Fault. A network fault has been detected in one direction only. Valid only when Network Topology is Ring and the node is the active ring supervisor |
| 4 | Rapid Fault/Restore Cycle. A series of rapid ring fault/restore cycles has been detected, according to the criteria given in 10.5.3.5. Similar to the Partial Network Fault status, the supervisor remains in a state with forwarding blocked on its ring ports. The condition shall be cleared explicitly via the “Clear Rapid Faults” service |

#### 7.9.4.4 Ring supervisor status

The Ring Supervisor Status attribute indicates the device’s status as a ring supervisor. Table 114 specifies the possible values for the Ring Supervisor Status.

**Table 114 – Ring Supervisor Status values**

| Ring Supervisor Status value | Description |
|---|---|
| 0 | The device is functioning as a backup supervisor |
| 1 | The device is functioning as the active ring supervisor |
| 2 | The device is functioning as a normal ring node (supervisor not enabled) |
| 3 | The device is functioning in a non-DLR topology (supervisor not enabled, and no other supervisor is present) |
| 4 | The device cannot support the currently operating ring parameters (Beacon Interval and/or Beacon Timeout) |

#### 7.9.4.5 Ring supervisor config

##### 7.9.4.5.1 General

The Ring Supervisor Config attribute contains the configuration parameters needed for ring operation: Supervisor Precedence, Beacon Interval, Beacon Timeout, VLAN ID, Supervisor Enable/Disable.

If the device is the active supervisor, changes to the attribute shall be applied immediately. When the Supervisor Precedence, Beacon Interval, Beacon Timeout or VLAN ID are changed on the active ring supervisor, the ring supervisor shall cease sending Beacon frames for two beacon timeout periods then shall resume sending Beacon frames using the new parameters.

Backup ring supervisors shall obtain values for the Beacon Interval, Beacon Timeout and VLAN ID from the Beacon frame sent by the active ring supervisor, and shall store those values in non-volatile storage. If the obtained values cannot be supported by the device (e.g., Beacon Interval too small), the device shall set the Ring Supervisor Status attribute as noted in the attribute description, shall report a minor recoverable fault via the Identity object Status attribute (5), and shall not take over as active supervisor after a ring reconfiguration.

When the Ring Supervisor Config attribute is modified on a backup supervisor, the behavior depends on the backup supervisor’s new precedence value compared to the active supervisor’s precedence value:

- new backup precedence value is greater than the current active ring supervisor’s precedence or of equal precedence with numerically higher MAC address than active

supervisor MAC address – backup shall immediately begin sending Beacon frames with the new parameters;

- new backup precedence value is less than the active supervisor's precedence or of equal precedence with numerically lower MAC address than active supervisor MAC address – modification to the Beacon Interval, Beacon Timeout, and VLAN ID shall be ignored.

Attempts to set invalid Ring Supervisor Config values shall result in error code 0x09 (Invalid attribute value) returned from the Set service, regardless of whether the device is an active or backup supervisor.

#### 7.9.4.5.2 Ring supervisor enable

The Ring Supervisor Enable item enables or disables the ring supervisor function in a supervisor-capable device. A value of TRUE enables the supervisor function. A value of FALSE disables the supervisor function. The default value is FALSE.

#### 7.9.4.5.3 Ring supervisor precedence

The Ring Supervisor Precedence item contains the user-assigned precedence value given to the ring supervisor. When multiple ring supervisors are enabled, the precedence value allows the user to configure the order in which the configured supervisors select the active supervisor.

The precedence value for the ring supervisor shall be chosen from the range 0 to 255, with numerically higher values indicating higher precedence. The default value shall be 0.

When more than one supervisor is enabled, the supervisor with highest precedence becomes active ring supervisor, in accordance with 10.5.2. If multiple supervisors have the same precedence, the supervisor with the numerically higher MAC address becomes the active supervisor.

#### 7.9.4.5.4 Beacon interval

The Beacon Interval item contains the interval in microseconds that the ring supervisor shall use in generating beacon frames. Per the DLR protocol specification in Clause 10, the default value shall be 400 µs. Supervisors shall support a range from 400 µs to 100 ms. Supervisors may support a Beacon Interval smaller than 400 µs, but this is not required. The absolute minimum Beacon Interval is 100 µs.

#### 7.9.4.5.5 Beacon timeout

The Beacon Timeout item contains the number of microseconds the ring supervisor shall wait for a beacon frame before declaring a beacon timeout.

The default value shall be 1 960 µs, which is based on a nominal network size of 50 nodes and 100 Mbit/s, full-duplex operation (see the performance calculations in 10.11). The user may wish to change the Beacon Timeout for other exceptional network circumstances (e.g., very large networks or very small high-performance motion networks).

The Beacon Timeout shall be at least 2 times the Beacon Interval value. If the Beacon Interval is changed and the Beacon Timeout becomes less than 2 times the Beacon Interval, the supervisor shall adjust the Beacon Timeout to be 2 times the Beacon Interval.

Supervisors shall support a range from 800 µs to 500 ms. Supervisors may support a Beacon Timeout of smaller than 800 µs but this is not required. The absolute minimum Beacon Timeout is 200 µs.

#### 7.9.4.5.6 DLR VLAN ID

The DLR VLAN ID contains the VLAN ID to be used in the DLR protocol frames. The DLR VLAN ID shall be used for all DLR protocol frames originated by the device, when the device is operating as the active ring supervisor. Devices that are not the active ring supervisor shall use the VLAN ID obtained from the active supervisor's frames (see Clause 10 for additional details).

The VLAN ID value shall be in the range 0 to 4 094. The default value shall be 0 (indicating no VLAN).

### 7.9.4.6 Ring Faults Count

The Ring Faults Count attribute contains the number of times since power up that the device has detected a ring fault, as either active or backup supervisor. If the Ring Supervisor Enable is set to FALSE, the Ring Faults Count shall be set to 0. The Ring Faults Count rolls over to 0 after it reaches its maximum value.

The attribute may also be reset to 0 via the Set_Attribute_Single service. Values other than 0 shall result in an error response.

### 7.9.4.7 Last active node on port 1

The Last Active Node on Port 1 attribute contains the IP address and/or Ethernet MAC address of the last node reachable through port 1 of an active ring supervisor. The value of the attribute is obtained via the Link_Status/Neighbor Status frames, as specified in 10.9.6.

On transition to FAULT_STATE, this attribute shall remain clear until the supervisor receives Link/Neighbor Status information.

On transition from FAULT_STATE to NORMAL_STATE, the value of the attribute shall be retained, to aid in diagnosing the previous ring fault.

The initial values of IP address and Ethernet MAC address shall be 0. When the device is not enabled as a ring supervisor, or is operating as the backup supervisor the IP address and Ethernet MAC address shall be 0.

### 7.9.4.8 Last active node on port 2

The Last Active Node on Port 2 attribute contains the IP address and/or Ethernet MAC address of the last node reachable through port 2 of an active ring supervisor. The value of the attribute is obtained via the Link_Status/Neighbor Status frames, as specified in 10.9.6.

On transition to FAULT_STATE, this attribute shall remain clear until the supervisor receives Link/Neighbor Status information.

On transition from FAULT_STATE to NORMAL_STATE, the value of the attribute shall be retained, to aid in diagnosing the previous ring fault.

The initial values of IP address and Ethernet MAC address shall be 0. When the device is not enabled as a ring supervisor, or is operating as the backup supervisor the IP address and Ethernet MAC address shall be 0.

### 7.9.4.9 Ring protocol participants count

This attribute contains the number of members in the Ring Protocol Participants List attribute. The count and the list are gathered by the active ring supervisor through Sign_On frame as specified in 10.9.9.

If the device is not the active supervisor, the attribute shall be set to 0.

#### 7.9.4.10 Ring protocol participants list

The Ring Protocol Participants List attribute contains the list of ring nodes participating in ring protocol. The participants list is gathered by the active ring supervisor via the Sign_On frame (see 10.9.9).

Since the size of the Ring Protocol Participants List could be large, depending on the number of nodes participating in the ring, this attribute shall be accessible with the Get_Member service.

Clients may elect to use the Get_Attribute_Single service to read the Ring Protocol Participants List. If the participants list is too large, the DLR object shall return error code 0x11 (Reply Data Too Large).

If the device is not active supervisor, status code 0x0C (Object State Conflict) shall be returned.

If the number of participants received as a result of the Sign_On process exceeds the capacity of the supervisor's Ring Protocol Participants List, the last entry in the list shall be all 0xFF's (0xFFFFFFFFFFFFFFFFFFFF).

#### 7.9.4.11 Active supervisor address

This attribute contains the IP address and/or Ethernet MAC address of the active ring supervisor. The initial values of IP address and Ethernet MAC address shall be 0, until the active ring supervisor is determined.

#### 7.9.4.12 Active supervisor precedence

This attribute contains the precedence value of the active ring supervisor. The initial value shall be 0, until the active ring supervisor is determined.

#### 7.9.4.13 Capability flags

The Capability Flags describe the DLR capabilities of the device, as specified in Table 115.

**Table 115 – Capability flags**

| Bit(s): | Name | Definition |
|---|---|---|
| 0 | Announce-based Ring Node[a] | Set if device's ring node implementation is based on processing of Announce frames (see 10.4) |
| 1 | Beacon-based Ring Node[a] | Set if device's ring node implementation is based on processing of Beacon frames (see 10.4) |
| 2-4 | Reserved | Shall be set to zero. |
| 5 | Supervisor Capable | Set if device is capable of providing the supervisor function. |
| 6 | Redundant Gateway Capable | Set if device is capable of providing the redundant gateway function. |
| 7 | Flush_Table frame Capable | Set if device is capable of supporting the Flush_Tables frame. |
| 8-31 | Reserved | Shall be set to zero. |

[a] Bits 0 and 1 are mutually exclusive. Exactly only one of these bits shall be set in the attribute value that a device reports.

#### 7.9.4.14 Redundant gateway config

##### 7.9.4.14.1 General

The Redundant Gateway Config attribute contains the configuration parameters needed for redundant gateway operation: Gateway enable/disable, Gateway Precedence, Advertise Interval, Advertise Timeout and Learning Update enable/disable.

If the device is the active gateway, changes to the attribute shall be applied immediately. When the Gateway Precedence, Advertise Interval, Advertise Timeout or Learning Update enable/disable are changed on the active gateway, the active gateway shall cease sending Advertise frames for 1,5 times old Advertise Timeout period and then shall resume sending Advertise frames using the new parameters.

Backup gateway devices shall obtain values for the Advertise Interval, Advertise Timeout and Learning Update enable/disable from the Advertise frame sent by the active gateway, and shall store those values in non-volatile storage. If the obtained values cannot be supported by the device (e.g., Advertise Interval too small), the device shall set the Redundant Gateway Status attribute as noted in the attribute description, shall report a minor recoverable fault via the Identity object Status attribute (5), and shall not take over as active gateway after a gateway reconfiguration.

When the Redundant Gateway Config attribute is modified on a backup gateway, the behavior depends on the backup gateway's new Precedence value compared to the active gateway's Precedence value.

- New backup Precedence value is greater than the current active gateway's Precedence or of equal Precedence with numerically higher MAC address than active gateway MAC address: backup shall immediately begin sending Advertise frames with the new parameters (see 10.10.4).
- New backup Precedence value is less than the active gateway's Precedence or of equal Precedence with numerically lower MAC address than active gateway MAC address: Advertise Interval, Advertise Timeout and Learning Update enable/disable shall be ignored.

Attempts to set invalid Redundant Gateway Config values shall result in error code 0x09 (Invalid attribute value) returned from the set service, regardless of whether the device is an active or backup gateway.

##### 7.9.4.14.2 Redundant gateway enable

The Redundant Gateway Enable item enables or disables the gateway function in a redundant gateway-capable device. A value of TRUE enables the gateway function. A value of FALSE disables the gateway function. The default value is FALSE.

##### 7.9.4.14.3 Gateway precedence

The Gateway Precedence item contains the user-assigned Precedence value given to a gateway. When multiple gateways are enabled, the Precedence value allows the user to configure the order in which the configured gateways select the active gateway.

The gateway Precedence shall be chosen from the range 0 to 255, with numerically higher values indicating higher Precedence. The default value shall be 0.

When more than one gateway is enabled the gateway with highest Precedence becomes active gateway, in accordance with Clause 10. If multiple gateways have the same Precedence, the gateway with the numerically higher MAC address becomes the active gateway.

#### 7.9.4.14.4 Advertise interval

The Advertise Interval item contains the interval in microseconds that the active gateway shall use in generating Advertise frames. Per the DLR protocol specification in Clause 10, the default value shall be 2 000 µs. Gateways shall support a range from 1 000 µs to 100 ms. Gateways may support an Advertise Interval smaller than 1 000 µs, but this is not required. The absolute minimum Advertise Interval is 200 µs.

#### 7.9.4.14.5 Advertise timeout

The Advertise Timeout item contains the number of microseconds a backup gateway shall wait for an Advertise frame before declaring an Advertise Timeout.

The default value shall be 5 000 µs, which is based on a nominal network size of 50 nodes and 100 Mbit/s, full-duplex operation (refer to the Performance Calculations in Clause 10). The user may wish to change the Advertise Timeout for exceptional network circumstances (e.g., very large networks or very small high-performance motion networks).

The Advertise Timeout shall be at least 2,5 times the Advertise Interval value. If the Advertise Interval is changed and the Advertise Timeout becomes less than 2,5 times the Advertise Interval, the gateway shall adjust the Advertise Timeout to be 2,5 times the Advertise Interval.

Gateways shall support a range from 2 500 µs to 500 ms. Gateways may support a Advertise Timeout of smaller than 2 500 µs but this is not required. The absolute minimum Advertise Timeout is 500 µs.

#### 7.9.4.14.6 Learning update enable

The Learning Update Enable item enables/disables transmission of Learning_Update frames by DLR nodes when they receive Flush_Tables frame from active gateway. This parameter is encoded by active gateway in Flush_Tables frame and sent to DLR nodes. The Learning_Update frames from DLR devices accelerate the new network topology learning by all non-DLR switches outside DLR network after an active gateway switchover. A value of TRUE enables the learning update function. A value of FALSE disables the learning update function. The default value is TRUE.

### 7.9.4.15 Redundant gateway status

The Redundant Gateway Status attribute indicates the device's status as a gateway. Table 116 shows the possible values.

**Table 116 – Redundant Gateway Status values**

| Redundant Gateway Status value | Description |
|---|---|
| 0 | Indicates the device is functioning as a non-gateway DLR node (gateway not enabled) |
| 1 | Indicates the device is functioning as a backup gateway |
| 2 | Indicates the device is functioning as the active gateway |
| 3 | Indicates gateway fault state due to loss of communication on uplink port |
| 4 | Indicates the device cannot support the currently operating gateway parameters (Advertise Interval and/or Advertise Timeout) |
| 5 | Indicates gateway fault state due to partial network fault (see 10.8.4.8) |

#### 7.9.4.16 Active gateway address

This attribute contains the IP address and/or Ethernet MAC address of the active gateway device. The initial values of IP address and Ethernet MAC address shall be 0, until the active gateway is determined.

#### 7.9.4.17 Active gateway precedence

This attribute contains the Precedence value of the active gateway device. The initial value shall be 0, until the active gateway is determined.

### 7.9.5 Common services

#### 7.9.5.1 General

The common services implemented by the DLR object are specified in Table 117.

**Table 117 – DLR object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x01 | Optional | Required | Get_Attribute_All | Returns multiple attributes in numerical order |
| 0x0E | Optional | Required | Get_Attribute_Single | Returns a single attribute |
| 0x10 | n/a | Conditional [a] | Set_Attribute_Single | Modifies a single attribute |
| 0x18 | n/a | Conditional [b] | Get_Member | Returns members of attribute |

[a] This service shall be implemented for devices capable of functioning as a ring supervisor and/or redundant gateway.

[b] Required for access to the Ring Protocol Participants List. The member services extended protocol for multiple sequential members shall be supported.

#### 7.9.5.2 Get_Attribute_All response

##### 7.9.5.2.1 Class level

At the class level, the Get_Attribute_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

##### 7.9.5.2.2 Instance level

At the instance level the Get_Attribute_All response varies depending on the revision of the object and the device's capabilities. The response format can be differentiated by the response length.

The response length for an Object Revision 1, non-supervisor device is 2 octets. The response format is specified in Table 118.

**Table 118 – Get_Attribute_All Response – Object Revision 1, non supervisor device**

| Attribute ID | Data type | Name |
|---|---|---|
| 0x01 | USINT | Network Topology |
| 0x02 | USINT | Network Status |

The response length for an Object Revision 1, supervisor-capable device is 50 octets. The response format is specified in Table 119.

**Table 119 – Get_Attribute_All Response – Object Revision 1, supervisor-capable device**

| Attribute ID | Data type | Name |
|---|---|---|
| 0x01 | USINT | Network Topology |
| 0x02 | USINT | Network Status |
| 0x03 | USINT | Ring Supervisor Status |
| 0x04 | STRUCT of | Ring Supervisor Config: |
| | BOOL | Ring Supervisor Enable |
| | USINT | Ring Supervisor Precedence |
| | UDINT | Beacon Interval |
| | UDINT | Beacon Timeout |
| | UINT | DLR VLAN ID |
| 0x05 | UINT | Ring Faults Count |
| 0x06 | STRUCT of | Last Active Node on Port 1 |
| | UDINT | Device IP Address |
| | USINT[6] | Device MAC Address |
| 0x07 | STRUCT of | Last Active Node on Port 2 |
| | UDINT | Device IP Address |
| | USINT[6] | Device MAC Address |
| 0x08 | UINT | Ring Protocol Participants Count |
| 0x0A | STRUCT of | Active Supervisor Address |
| | UDINT | Device IP Address |
| | USINT[6] | Device MAC Address |
| 0x0B | USINT | Active Supervisor Precedence |

The response length for an Object Revision 2, non-supervisor–capable device is 16 octets. The response format is specified in Table 120.

**Table 120 – Get_Attribute_All Response – Object Revision 2, non supervisor device**

| Attribute ID | Data type | Name |
|---|---|---|
| 0x01 | USINT | Network Topology |
| 0x02 | USINT | Network Status |
| 0x0A | STRUCT of | Active Supervisor Address |
| | UDINT | Device IP Address |
| | USINT[6] | Device MAC Address |
| 0x0C | DWORD | Capability Flags |

In all other cases, the Get_Attribute_All response (see Table 121) shall contain the instance attributes 0x01 through 0x08 and 0x0A through 0x10, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default value specified.

The response length for an Object Revision 2, supervisor–capable device is 54 octets.

The response length for an Object Revision 3, non-supervisor, non-gateway device is 54 octets.

The response length for an Object Revision 3, supervisor, non-gateway device is 54 octets.

The response length for an Object Revision 3, gateway-capable device is 77 octets.

**Table 121 – Get_Attribute_All Response – All other cases**

| Attribute ID | Data type | Name | Default value (if not implemented) |
|---|---|---|---|
| 0x01 | USINT | Network Topology | |
| 0x02 | USINT | Network Status | |
| 0x03 | USINT | Ring Supervisor Status | 0xFF – Not applicable |
| 0x04 | STRUCT of | Ring Supervisor Config | |
| | BOOL | Ring Supervisor Enable | 0 |
| | USINT | Ring Supervisor Precedence | 0 |
| | UDINT | Beacon Interval | 0 |
| | UDINT | Beacon Timeout | 0 |
| | UINT | DLR VLAN ID | 0 |
| 0x05 | UINT | Ring Faults Count | 0 |
| 0x06 | STRUCT of | Last Active Node on Port 1 | |
| | UDINT | Device IP Address | 0 |
| | USINT[6] | Device MAC Address | All zero |
| 0x07 | STRUCT of | Last Active Node on Port 2 | |
| | UDINT | Device IP Address | 0 |
| | USINT[6] | Device MAC Address | All zeroes |
| 0x08 | UINT | Ring Protocol Participants Count | 0xFFFF – Not applicable |
| 0x0A | STRUCT of | Active Supervisor Address | |
| | UDINT | Device IP Address | |
| | USINT[6] | Device MAC Address | |
| 0x0B | USINT | Active Supervisor Precedence | 0 |
| 0x0C | DWORD | Capability Flags | |
| 0x0D | STRUCT of | Redundant Gateway Configuration | |
| | BOOL | Redundant Gateway Enable | |
| | USINT | Gateway Precedence | |
| | UDINT | Advertise Interval | |
| | UDINT | Advertise Timeout | |
| | BOOL | Learning Update Enable | |
| 0x0E | USINT | Redundant Gateway Status | |
| 0x0F | STRUCT of | Active Gateway Address | |
| | UDINT | Device IP Address | |
| | USINT[6] | Device MAC Address | |
| 0x10 | USINT | Active Gateway Precedence | |

### 7.9.6 Class specific services

#### 7.9.6.1 General

The DLR object shall support the class specific services specified in Table 122.

**Table 122 – DLR object class specific services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x4B | n/a | Conditional [a] | Verify_Fault_Location | Causes ring supervisor to verify fault location by issuing Locate_Fault ring protocol message to ring nodes and update Last Active Node 1 and Last Active Node 2 attributes |
| 0x4C | n/a | Conditional [a] | Clear_Rapid_Faults | Clears the Rapid Fault/Restore Cycle Detected condition in the ring supervisor, allowing the supervisor to return to normal operation |
| 0x4D | n/a | Conditional [a] | Restart_Sign_On | Restart Sign On process and refresh DLR participants list |
| 0x04E | n/a | Conditional [b] | Clear_Gateway_Partial_Fault | Clears the partial network fault condition in the gateway device, allowing the gateway to return to normal operation |

[a] This service shall be implemented for devices capable of functioning as a ring supervisor.

[b] This service shall be implemented for devices capable of functioning as a redundant gateway.

#### 7.9.6.2 Verify_Fault_Location

The Verify_Fault_Location service shall cause an active ring supervisor to verify a ring fault location by retransmitting the Locate_Fault frame to ring nodes (see 10.9.7). The Last Active Node 1 and Last Active Node 2 attributes shall be updated based on the response to the Locate_Fault frame.

There are no parameters for either the Verify_Fault_Location request or response.

If the Verify _Fault_Location service is received when the supervisor is not enabled, or is currently the backup supervisor, or is the active supervisor but not in fault state, status code 0x0C (Object State Conflict) shall be returned, and the Last Active Node 1 and Last Active Node 2 attributes shall be set to 0.

#### 7.9.6.3 Clear_Rapid_Faults

The Clear_Rapid_Faults service shall clear the condition where the ring supervisor has detected a cycle of rapid ring fault/restore (as defined in 10.5.3.5). Upon clearing the condition, the ring supervisor shall return to normal operation.

If the Clear_Rapid_Faults service is received when the supervisor is not enabled, or is currently the backup supervisor, or is the active supervisor but not in the rapid fault/restore condition, status code 0x0C (Object State Conflict) shall be returned.

#### 7.9.6.4 Restart_Sign_On

The Restart_Sign_On service shall restart Sign On process (see 10.5.2.3 Sign On for more information) by active ring supervisor, if it is not currently in progress.

If the Restart_Sign_On service is received when the supervisor is not enabled, or is currently the backup supervisor, or is the active supervisor but not in the NORMAL_STATE, status code 0x0C (Object State Conflict) shall be returned. If the Restart_Sign_On service is received when a prior Sign On process is in progress, success shall be returned and the request shall be ignored.

#### 7.9.6.5 Clear_Gateway_Partial_Fault

The Clear_Gateway_Partial_Fault service shall clear the condition where the gateway has detected a partial network fault (as defined in Clause 10). Upon clearing the condition, the gateway shall execute state machine as specified in Clause 10.

If the Clear_Gateway_Partial_Fault service is received when the gateway is not enabled, or is not in partial network fault condition, status code 0x0C (Object State Conflict) shall be returned.

### 7.10 QoS object

#### 7.10.1 Overview

Quality of Service (QoS) is a general term that is applied to mechanisms used to treat traffic streams with different relative priorities or other delivery characteristics. Standard QoS mechanisms include IEEE 802.1D/IEEE 802.1Q (Ethernet frame priority) and Differentiated Services (DiffServ) in the TCP/IP protocol suite.

The QoS object provides a means to configure certain QoS-related behaviors in CP 2/2 devices.

The QoS Object is required for devices that support sending CP 2/2 messages with non-zero DiffServ code points (DSCP), or sending CP 2/2 messages in IEEE 802.1Q tagged frames.

#### 7.10.2 Revision History

The revision history of the QoS object is described in Table 123.

**Table 123 – QoS object revision history**

| Revision | History |
|---|---|
| 01 | Initial Definition |

#### 7.10.3 Class attributes

The QoS object shall support the class attributes as specified in Table 124.

**Table 124 – QoS object class attributes**

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| 0x01 | Required | Get | Revision | UINT | Revision of this object | Current value for this attribute = 1 |
| 0x02 to 0x07 | These class attributes are optional and are described in IEC 61158-5-2 | | | | | |

### 7.10.4 Instance Attributes

#### 7.10.4.1 General

The QoS object shall support the instance attributes as specified in Table 125.

**Table 125 – QoS object instance attributes**

| Attr ID | Need in implem | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| 0x01 | Conditional[a] | Set | NV | 802.1Q Tag Enable | USINT | Enables or disables sending IEEE 802.1Q frames on CP 2/2 and IEC 61588 messages | A value of 0 indicates tagged frames disabled. A value of 1 indicates tagged frames enabled.<br>The default value shall be 0 |
| 0x02 | Conditional[b] | Set | NV | DSCP PTP Event | USINT | DSCP value for PTP (IEC 61588) event messages | See 7.10.4.3 |
| 0x03 | Conditional[b] | Set | NV | DSCP PTP General | USINT | DSCP value for PTP (IEC 61588) general messages | See 7.10.4.3 |
| 0x04 | Required | Set | NV | DSCP Urgent | USINT | DSCP value for CP 2/2 transport class 0/1 Urgent priority messages | See 7.10.4.3 |
| 0x05 | Required | Set | NV | DSCP Scheduled | USINT | DSCP value for CP 2/2 transport class 0/1 Scheduled priority messages | See 7.10.4.3 |
| 0x06 | Required | Set | NV | DSCP High | USINT | DSCP value for CP 2/2 transport class 0/1 High priority messages | See 7.10.4.3 |
| 0x07 | Required | Set | NV | DSCP Low | USINT | DSCP value for CP 2/2 transport class 0/1 low priority messages | See 7.10.4.3 |
| 0x08 | Required | Set | NV | DSCP Explicit | USINT | DSCP value for CP 2/2 explicit messages (transport class 2/3 and UCMM) and all other CP 2/2 encapsulation messages | See 7.10.4.3 |

[a] Required if the device supports sending IEEE 802.1Q frames.

[b] Required if the device supports the Time Sync object.

#### 7.10.4.2 802.1Q Tag Enable

The 802.1Q Tag Enable attribute enables or disables sending IEEE 802.1Q frames on CP 2/2 and IEC 61588 messages. When the attribute is enabled, the device shall send IEEE 802.1Q frames for all CP 2/2 and IEC 61588 messages.

A value of 1 indicates enabled. A value of 0 indicates disabled. The default value for the attribute shall be 0. A change to the value of the attribute shall take effect the next time the device restarts.

Devices shall always use the corresponding DSCP values regardless of whether IEEE 802.1Q frames are enabled or disabled.

#### 7.10.4.3 DSCP value attributes

Attributes 0x02 to 0x08 contain the DSCP values that shall be used for the different types of CP 2/2 traffic.

The format of the DSCP value within the IP header is specified in IEC 61158-6-2, 11.7.4. Since the DSCP field has a size of 6 bits, the valid range of values for these attributes is 0 to 63. The DSCP value, if placed directly in the ToS field in the IP header, shall be shifted left 2 bits.

Table 126 specifies the default DSCP values and traffic usages.

**Table 126 – Default DCSP values and usages**

| Attribute | Traffic type usage | Default DSCP |
|---|---|---|
| DSCP PTP Event | PTP (IEC 61588) event messages | 59 (‘111011’) |
| DSCP PTP General | PTP (IEC 61588) general messages | 47 (‘101111’) |
| DSCP Urgent | Transport class 0/1 messages with Urgent priority | 55 (‘110111’) |
| DSCP Scheduled | Transport class 0/1 messages with Scheduled priority | 47 (‘101111’) |
| DSCP High | Transport class 0/1 messages with High priority | 43 (‘101011’) |
| DSCP Low | Transport class 0/1 messages with Low priority | 31 (‘011111’) |
| DSCP Explicit | UCMM messages, transport class 2/3 messages, and all other CP 2/2 encapsulation messages | 27 (‘011011’) |

A change to the value of the above attributes shall take effect the next time the device restarts.

### 7.10.5 Common services

The QoS object shall support the common services as specified in Table 127.

**Table 127 – QoS object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x01 | Optional | N/A | Get_Attribute_All | Returns multiple attributes in numerical order |
| 0x0E | Conditional [a] | Required | Get_Attribute_Single | Returns the contents of the specified attribute |
| 0x10 | N/A | Required | Set_Attribute_Single | Modifies a single attribute |

[a] Required if any class attributes are implemented.

### 7.10.6 Get_Attribute_All response (class level)

The Get_Attribute_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

## 7.11 Port object

### 7.11.1 Overview

The Port object describes the CPF 2 ports present on the device. One instance shall exist for each CPF 2 routable port (see IEC 61158-5-2, 6.2.2.3 for the definition of routable port). Instances may also exist for non-routable ports. Devices with a single CPF 2 port are not required to support the Port object.

### 7.11.2 Class attributes

The Port Object shall support the class attributes as specified in Table 128.

**Table 128 – Port object class attributes**

| Attribute ID | Need in implementation | Access rule | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|
| 0x01 | This class attribute is optional and is described in IEC 61158-5-2 | | | | | |
| 0x02 | Required | Get | Max Instance | UINT | Maximum instance number | |
| 0x03 | Required | Get | Num Instances | UINT | Number of ports currently instantiated | |
| 0x04 to 0x07 | These class attributes are optional and are described in IEC 61158-5-2 | | | | | |
| 0x08 | Required | Get | Entry Port | UINT | Returns the instance of the Port object that describes the port through which this request entered the device | |
| 0x09 | Required | Get | Port Instance Info | ARRAY of STRUCT of | Array of structures containing instance attributes 1 and 2 from each instance | The array is indexed by instance number, up to the maximum number of instances.<br>The values at index 1 (offset 0) and any non-instantiated instances shall be zero. |
| | | | Port Type | UINT | Enumerates the type of port | See instance attribute #1 |
| | | | Port Number | UINT | CPF 2 port number associated with this port | See instance attribute #2 |

### 7.11.3 Instance attributes

#### 7.11.3.1 General

The Port object shall support the instance attributes as specified in Table 129.

**Table 129 – Port object instance attributes**

| Attribute ID | Need in implement ation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| 0x01 | Required | Get | NV | Port Type | UINT | Enumerates the type of port | See 7.11.3.2 |
| 0x02 | Required | Get | NV | Port Number | UINT | CPF 2 port number associated with this port | See 7.11.3.3 |
| 0x03 | Required | Get | NV | Link object | STRUCT of | | |
| | | | | Path Length | UINT | Number of 16 bit words in the following path | Range = 2 to 6 |
| | | | | Link Path | Padded EPATH | Logical path segments that identify the object for this port. For example, this could be the TCP/IP Interface object. | The path shall consist of one logical class segment and one logical instance segment.<br>The maximum size is 12 octets. See definition of logical segments in IEC 61158-6-2, 4.1.9.4. |
| 0x04 | Required | Get | NV | Port Name | SHORT_ STRING | String which names the physical network port. The maximum number of characters in the string is 64.<br>For example, this may be “Port A”. | This value is unique for each physical network port. If multiple CPF 2 ports use the same physical network port, they shall each have the same Port Name value.<br>Likewise, a CPF 2 port shall not have the same Port Name value as any other CPF 2port if they do not share the same physical network port. |
| 0x05 | Optional | Get | NV | Port Type Name | SHORT_ STRING | String which names the port type. The maximum number of characters in the string is 64.<br>For example, this may be "EtherNet/IP". | |
| 0x06 | Optional | Set | NV | Port Description | SHORT_ STRING | String which describes the port. The maximum number of characters in the string is 64.<br>For example, this may be "Product Line 22". | |
| 0x07 | Required | Get | NV | Node Address [a] | Padded EPATH | Node number of this device on port. The range within this data type is restricted to a Port Segment. | The encoded port number shall match the value presented in attribute 2. |
| 0x08 | Conditional | Get | NV | Port Node Range [b] | STRUCT of | | |
| | | | | Minimum Node Number | UINT | Minimum node number on port | |
| | | | | Maximum Node Number | UINT | Maximum node number on port | |

| Attribute ID | Need in implement ation | Access rule | NV | Name | Data type | Description of attribute | Semantics of values |
|---|---|---|---|---|---|---|---|
| 0x09 | Optional | Get | NV | Port Key | Packed EPATH | Electronic key of network/chassis this port is attached to.<br>This attribute shall be limited to format 4 of the Logical Electronic Key segment. | |

a A device which does not have a node number on the port may indicate a zero length node address within the Port Segment (0x10 0x00).

b If a device can report its port characteristics within the range allowed (e.g. Type 2 MAC ID) then it shall support this attribute. Otherwise (e.g. Internet address) it shall not support this attribute.

#### 7.11.3.2 Port Type

All Port Type values, with the exception of 0, indicate routable ports of the defined type.

The Port Type values are:

0 = used when the port does not support CIP routing. Attribute 2 (Port Number) is ignored;

1 = reserved for compatibility with existing protocols;

2 = ControlNet;

3 = ControlNet redundant;

4 = EtherNet/IP (this port type was formerly known as TCP/IP);

5 = DeviceNet;

6 to 99 = reserved for compatibility with existing protocols;

100 to 199 = Vendor Specific;

200 = CompoNet;

201 = Modbus/TCP;

202 = Modbus/SL;

203 = SERCOS_III

204 to 65 534 = Reserved for future use;

65 535 = unconfigured port

#### 7.11.3.3 Port Number

Manufacturer assigns a unique value to identify each communication port. Value 1 is defined for internal product use (i.e. backplane). Value 0 is reserved and can not be used.

The Port Number value is the Port Identifier defined for port segments in IEC 61158-6-2, 4.1.9.3, and used for routing.

If Port Type is 0, the Port Number attribute is ignored.

### 7.11.4 Common services

#### 7.11.4.1 General

The common services implemented by the Port object are specified in Table 130.

**Table 130 – Port object common services**

| Service code | Need in implementation | | Service name | Description of service |
|---|---|---|---|---|
| | Class | Instance | | |
| 0x01 | Optional | Optional | Get_Attribute_All | Returns the contents of all attributes of the class/instance |
| 0x05 | n/a | Optional | Reset | |
| 0x0E | Conditional [a] | Conditional [a] | Get_Attribute_Single | Used to read a Port object class/instance attribute |
| 0x10 | n/a | Optional | Set_Attribute_Single | |

[a] This service shall be implemented if any of the class/instance attribute are supported.

#### 7.11.4.2 Get_Attribute_All response

##### 7.11.4.2.1 Class level

At the class level, the Get_Attribute_All response shall concatenate attributes 1, 2, 3, 8 and 9 in that order. If class attribute 1 (Revision) is not supported, then a default value of one (1) shall be returned.

##### 7.11.4.2.2 Instance level

At the instance level the Get_Attribute_All response shall concatenate attributes 1, 2, 3, 4 and 7 in that order.

## 8 Other DLE elements of procedure

### 8.1 Network attachment monitor (NAM)

#### 8.1.1 General

The network attachment monitor (NAM) shall control the DLL to allow new nodes to non-disruptively join a working link.

The NAM controls attaching the DLL to an existing link without causing disruption to transmissions on that link. It accomplishes this by monitoring and controlling the DLL via the Station Management and the normal send/receive interfaces to the DLL as summarized in Table 131 and Figure 26.

**Table 131 – NAM states**

| State | Actions |
|---|---|
| **Check for Cable** | 1) auto-address nodes shall find an address before entering (see 8.1.3)<br>2) use default parameters (see 8.1.2)<br>3) transmit DLPDUs, but no moderators<br>4) if receive rogue moderator, go to “Check for Moderator”<br>5) if receive any good DLPDU, transmit one more DLPDU, then go to “Wait to Rogue” |
| **Wait to Rogue** | 1) use default parameters (see 8.1.2)<br>2) transmit no DLPDUs<br>3) if hear rogue moderator, go to “Check for Moderator”<br>4) if hear no rogue moderator for 400 – 600 ms, go to “I’m Alive” |

| State | Actions |
|---|---|
| **Check for Moderator** | 1) transmit no DLPDUs<br>2) if hear 9 identical valid moderators in successive NUTs, go to “I’m Alive” using the parameters in those moderators<br>3) if have not heard a moderator for 1,0 s – 3,0 s, go to “Check for Cable”<br>4) if receive moderator with invalid link parameters, stay in this state, but change network status indicator to invalid link configuration (flashing red / green) |
| **I’m Alive** | 1) auto-address nodes shall find an address before entering (see 8.1.3)<br>2) transmit DLPDUs, including moderators if this node has the lowest MAC ID<br>3) transmit three fixed tag 0x80 Lpackets<br>4) after sending them go to “Attached”<br>5) if receive rogue moderator, go to “Check for Moderator” |
| **Attached** | 1) normal network operation<br>2) transmit DLPDUs, including moderators if this node has the lowest MAC ID<br>3) if receive rogue moderator, go to “Check for Moderator”<br>4) if no other nodes on link for 8 NUTs, go to “Check for Cable” |

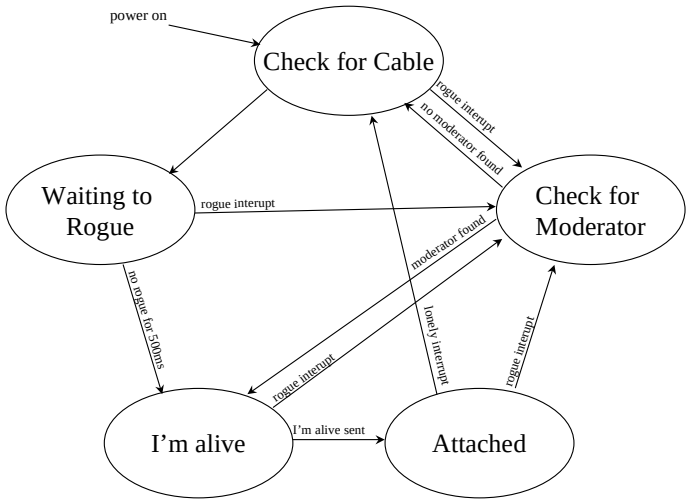



**Figure 26 – NAM state machine**

### 8.1.2 Default parameters

The default link parameters for a node shall be as shown in Table 132:

**Table 132 – Default link parameters**

| Parameter | Data type | Value |
|---|---|---|
| NUT_length | UINT | 100 ms (10 000 intervals of 10 µs each) |
| smax | USINT | 0 |
| umax | USINT | 99 |
| slotTime | USINT | 254 (255 µs) |
| blanking | USINT | 6 (octet times) |
| gb_start | USINT | 610 µs (61 intervals of 10 µs each) |
| gb_center | USINT | 450 µs (45 intervals of 10 µs each) |
| modulus | USINT | 127 |
| gb_prestart | USINT | 920 µs (92 intervals of 10 µs each) |

### 8.1.3 Auto-addressing

Some nodes may have no pre-assigned MAC ID. These nodes, known as auto-address nodes, shall search for an unused MAC ID by observing the link. The auto-address node shall not transmit until a free MAC ID is found.

NOTE Typically, auto-address nodes are transient nodes such as hand-held terminals.

### 8.1.4 Valid MAC IDs

On a link with default configuration parameters, an auto-address node shall search in the range 92 through 99. On a link with any other configuration parameters, an auto-address node shall search in the range SMAX+1 through UMAX. A free MAC ID shall be declared found if at least three sequential unscheduled transmit opportunities for that MAC ID have passed without receiving any DLPDUs from that MAC ID.

NOTE The ControlNet object provides an interface to determine the MAC ID which has been chosen.

### 8.1.5 State machine description

The following state machine description shall define the behavior of the network attachment monitor:

```
// Network Attachment Monitor state machine description
//
// This state machine monitors and controls the DLL to make sure that it goes on-line
// in a fashion that does not disrupt a running network.
//
//
// Lpacket constants: masks for ctl octet
//
#define FIXEDSCREEN 1
#define TAGPAD 2

#define DATAPAD 4
#define OCTETWORD 8
//
// moderator Lpacket constants
//
#define MODERATOR_SIZE 9      // moderator Lpacket size octet
#define MODERATOR_CTL 1       // moderator Lpacket control octet
#define MODERATOR_TAG 0       // moderator Lpacket tag octet

//
// Lpacket class
//
class Lpacket
   {
   public:

   USINT size;         // return the size octet
   USINT ctl;          // return the control octet

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }
   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }
   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }
   // get the next octet to be transmitted
   USINT get_next_octet(void);
```

```
    // is this Lpacket aborted?
    BOOL abort(void);
    };


class fixedLpacket: public Lpacket
    {
    public:
    USINT     tag;
    USINT     dest;
    };


class moderatorLpacket: public fixedLpacket
    {
    public:
    UINT      NUT_length;
    USINT     smax;
    USINT     umax;
    USINT     slotTime;
    USINT     blanking;
    USINT     gb_start;
    USINT     gb_center;
    USINT     usr;
    USINT     interval_count;
    USINT     modulus;
    USINT     tMinus;
    USINT     gb_prestart;
    USINT     spare;


    BOOL isValid(void);
                    // this checks for umax<myaddr, and similar parameter
                    // error that would prevent a node from successfully
                    // joining a network


    BOOL operator==(moderatorLpacket &);
                    // the equality test for moderator Lpackets excludes
                    // the interval count, and usr, but includes the
                    // tMinus counter
    };


//
// interface to station management
//
BOOL SM_powerup;          // input indication: powerup has occurred
void SM_LEDS(LED_STATE);         // set the LEDS to a specified pattern
#define LED_bad_network_parameters     // flashes the network status indicator

class DLL_config_data
    {
    public:
    BOOL   listen_only;
    USINT  myaddr;
    UINT   NUT_length;
    USINT  smax;
    USINT  umax;
    USINT  slotTime;
    USINT  blanking;
    USINT  gb_start;
    USINT  gb_center;
    USINT  interval_count;
    USINT  modulus;
    USINT  gb_prestart;
    };
```

```
//
// interface to DLL
//
void DLL_xmit_fixed_request(char *comment);
void DLL_xmit_fixed_confirm(void);
void DLL_recv_fixed_indication();



//
// These timer objects are as used internally to the ACM of the DLL.
//
// various behaviors for timers
enum {ONCE_UP, ONCE_DOWN, REPEAT_UP, REPEAT_DOWN} timer_mode;



class timer
   {
   public:

   float count;           // number of ticks left
   float preset;          // initialize ticks, or cycle length
   BOOL expire;           // latched flag when timer expires
   timer_mode mode;

   // constructor: defines mode
   timer(timer_mode m)
      {
      mode = m;
      }

   void restart(void)
      {
      if(mode == REPEAT_DOWN
         || mode == ONCE_DOWN)  // if counting down,
         {
         count = preset;     // start with preset
         }
      else
         {
         count = 0;      // else start from zero
         }
      // start counting
      }

   void start(float interval)   // start counting down from interval towards zero
      {
      mode = ONCE_DOWN;
      preset = interval;     // set interval
      restart();          // start counting
      }

   void begin_counting(void)    // start counting up from zero
      {
      mode = ONCE_UP;
      restart();
      }

   bool expired(void)        // returns true if the timer has expired; clears flag
      {
      BOOL tmp;
      tmp = expire;
      expire = 0;
      return tmp;
      }
   };



//
// time conversion factors
//
#define usec
```

```
#define msec *1000
#define sec *1000000


class timer timer(ONCE_DOWN);    // general purpose timer
int counter;           // general purpose counter
moderatorLpacket new_mod;    // buffer for getting new moderator Lpackets
moderatorLpacket old_mod;    // buffer for saving moderator Lpackets


DLL_config_data default_config;  // DLL config parameters for default mode
DLL_config_data DLL_config;      // scratch DLL config buffer


// Wait for powerup.
// Bring the DLL on-line.
// It is assumed that the node's MAC ID is known before this machine is allowed to
power-up.
// If this node is auto-addressed, a free MAC ID shall be determined
//  before enabling this machine.


state: powerup0

   event:    SM_powerup
   destination: checkForCable0
   action:
      DLL_set_current_request(default_config);  // set default parameters


state: checkForCable0

   event:    DLL_set_current_confirm()    // default parameters were set
   destination: checkForCable1
   action:
      DLL_enable_moderator_request(FALSE);      // disable moderators
      DLL_online_request(TRUE);       // send DLL on-line


state: checkForCable1

   event:    DLL_online_confirm()         // DLL is now on-line, but can't moderate
   destination: checkForCable2


state: checkForCable2

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   event:       DLL_recv_fixed_indication
             || DLL_recv_gen_indication   // any Lpacket received
   destination: waitForRogue
   action:
      wait until transmit once more;
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_moderator_request(FALSE);      // re-enable moderators
      timer.start(500 msec);

   event:    MAC has not transmitted any frames for 1300 to 1500 msec
   destination: checkForCable0
   action:
      DLL_set_current_request(default_config);     // set default parameters
```

```
state: waitForRogue

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   event:    timer.expired()
   destination: alive0
   action:
      DLL_listen_only_request(FALSE);     // send DLL to listen only
      DLL_xmit_fixed_request("send an I'm alive message");



//
// in the "alive" states, the DLL is on-line
// three "I'm alive" message are sent
// if lonely is detected, do nothing
//
state: alive0

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   // first message sent, send the second
   event:    DLL_xmit_fixed_confirm();
   destination: alive1
   action:
      DLL_xmit_fixed_request("send an I'm alive message");



state: alive1

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   // second message sent, send the third
   event:    DLL_xmit_fixed_confirm();
   destination: alive2
   action:
      DLL_xmit_fixed_request("send an I'm alive message");

state: alive2

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   // third message send, consider attachment  to be complete
   event:    DLL_xmit_fixed_confirm();
   destination: attached

//
// the DLL is on-line and fully functional
//
state: attached

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
```

```
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   event:    DLL_SM_report(lonely)
   destination: checkForCable
   action:
      DLL_enable_moderator_request(FALSE);      // disable moderators

//
// a rogue event was detected
// the DLL instantly disables itself (it goes to its rogue state)
//
state: checkForModerator0

   event:    DLL_enable_fixed_confirm();      // receive moderators enabled
   destination: checkForModerator1
   action:
      counter = 0;              // init moderator counter
      old_mod = NULL;
      timer.start(3 sec);

state: checkForModerator1

   // timeout
   event:    timer.expired
   destination: checkForCable0
   action:
      DLL_set_current_request(default_params);  // set default parameters


   // ignore irrelevant received DLPDUs
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is not mod
         || received Lpacket is not TUI
         || TUI.net_change == false
   destination: checkForModerator1

   // received a TUI indicating a net change in progress -- reset counter
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is TUI
         && TUI.net_change == TRUE    // the new moderator is different
   destination: checkForModerator1
   action:
      counter = 0;

   // received a moderator that would not permit this node
   // to operate normally (for example,  - myaddr>umax)
   // reset counter and flash bad_net_parameter indicator
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   !isValid(new_mod)         // the new moderator is no good for us
   destination: checkForModerator1
   action:
      SM_LEDS(LED_bad_network_parameters);      // flash the indicator
      old_mod = new_mod;
      counter = 0;

   // received moderator that doesn't match the last one -- reset counter
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is mod
         && new_mod != old_mod        // the new moderator is different
   destination: checkForModerator1
   action:
      old_mod = new_mod;
      counter = 0;

   // received moderator that matches, but not the ninth match -- just increment the
counter
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is mod
         && isValid(new_mod)
         && new_mod==old_mod
         && counter<8
```

```
   destination: checkForModerator1
   action:
      counter++;

   // have received 9 successive identical moderators
   // adopt these parameters and proceed to go back on-line
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is mod
          && isValid(new_mod)
          && new_mod==old_mod
          && counter==8
   destination: checkForModerator2
   action:
      DLL_online_request(FALSE);       // send DLL off-line

//
// second step to going back on-line
//
state: checkForModerator2

   event:    DLL_online_confirm()          // now off-line
   destination: check ForModerator3
   action:
      copy parameters from new_mod to DLL_config;   // get net configuration from
moderator
      DLL_set_current_request(DLL_config);       // adopt those parameters

//
// third step to going back on-line
//
state: checkForModerator3

   event:    DLL_set_current_confirm()     // have now adopted new parameters
   destination: alive0
   action:
      DLL_listen_only_request(FALSE);     // send DLL to listen only
      DLL_xmit_fixed_request("send an I'm alive message");
```

## 8.2 Calculating link parameters

### 8.2.1 Link parameters

Subclause 8.2 describes the requirements governing the selection of values for the following configuration variables:

– NUT_length;
– gb_prestart;
– gb_start;
– gb_center;
– slotTime;
– blanking.

NOTE The definitions of the link parameters include `NUT_length`, `gb_center`, `gb_prestart`, `gb_start`, `blanking`, and `slotTime`.

### 8.2.2 Conditions affecting link parameters

Parameter value selection shall allow for worst case conditions of these factors:

– signal propagation time between nodes;
– the timing reference accuracy at various nodes;
– moderator changes resulting in a change in the cable distance or signal propagation time between moderator and non-moderator nodes.

NOTE 1 A typical moderator change occurs as follows: At some random time the previous moderator is disconnected from the cable, loses power, or stops transmitting moderators for some other reason. Two NUTs

(sometimes three) pass during which no moderator DLPDUs are transmitted. The new moderator transmits its first moderator DLPDU in the guardband of the third NUT since it last received a valid moderator DLPDU.

NOTE 2 In the event that the previous moderator was lost near the time that the moderator DLPDU was being transmitted it could be possible that the new moderator received the last moderator DLPDU from the previous moderator, while other nodes would not have received the last moderator DLPDU.

It is therefore required that nodes tolerate an interval of 4 NUTs between reception of valid moderator DLPDUs.

### 8.2.3 Moderator change

In the requirements which follow, the term “moderator change” shall be interpreted by a non-moderator node as:

– reception of a valid moderator DLPDU from the original moderator node,
– a time interval of up to 4 NUTs during which no valid moderator DLPDUs are received,
– reception of a valid moderator DLPDU from a second moderator node.

### 8.2.4 NUT timing

#### 8.2.4.1 Tone

A network update time (NUT) for a node shall begin at a point in time called the tone, and shall end at the next tone.

NOTE The times specified in 8.2.4 assume a perfectly accurate clock. Any permitted inaccuracy is explicitly included in the requirements. Sources of inaccuracy can include firmware delay, hardware delay, and local crystal inaccuracy.

Since some of the time range requirements do not explicitly specify the clock accuracy tolerance, implementations should provide that internal timers are set such that the requirements are met.

#### 8.2.4.2 Clock accuracy

The timing specifications for PhL Signaling to support the tone and NUT precision shall be as defined in Table 133.

**Table 133 – PhL timing characteristics**

| Specification | Limits / characteristics | Comments |
|---|---|---|
| Data Rate | 5 Mbit/s ± CA | also called M_Symbol rate, *data “zero” or “one”* |
| Bit Time | 200 ns ± CA | also called M_Symbol time, *data “zero” or “one”* |
| PhL symbol time | 100 ns ± CA | also called Phy_Symbol time, see data encoding rules |
| Clock Accuracy (CA) | ± 150 parts/million maximum | including temperature, long term, and short term stability |

#### 8.2.4.3 Restriction on `NUT_length`

The maximum value of `NUT_length` shall be 10 000 (100 ms). The minimum value `NUT_length` shall ensure that each NUT allows a maximum length unscheduled Lpacket to be sent after the scheduled Lpackets have been sent.

NOTE For example, with no scheduled connections, the minimum value of `NUT_length` is about 1 ms.

#### 8.2.4.4 Moderator node timing

The moderator node shall begin a new NUT at (NUT_length × 10 μs) ± CA after the previous tone. The reception of a valid or invalid DLPDU shall not affect the timing of the transmission of the moderator DLPDU by the moderator node.

The moderator node shall begin transmission of the moderator DLPDU at (NUT_length – gb_center) × 10 μs ± CA after the tone.

#### 8.2.4.5 Non-moderator node timing

A non-moderator node shall begin a new NUT at (gb_center × 10 - 40 ± 2) μs after the last M_Symbol of the end delimiter of any valid received moderator DLPDU. A non-moderator node shall begin a new NUT at (NUT_length × 10) μs ± CA after the previous tone if a valid moderator DLPDU is not received in the current NUT.

Every non-moderator node shall receive and process any valid moderator DLPDU which begins between (tone + 20) μs and (tone + (NUT_length × 10 - 30) μs ± CA).

#### 8.2.4.6 Post-tone silence

No DLPDUs shall be transmitted between the tone and 20 μs after the tone.

#### 8.2.4.7 Pre-tone silence

No DLPDU shall be transmitted, except the moderator DLPDU, that does not terminate before ((NUT_length - gb_prestart) × 10 + 11,2) μs ± CA after the tone.

The value of gb_prestart shall provide that the blanking time after the last non-moderator DLPDU transmitted by any node and received by any other node expires before the start of the guardband in the receiving node. This condition shall also be met during a moderator change.

#### 8.2.4.8 Guardband start

The guardband shall start at (NUT_length - gb_start) × 10 μs ± CA after the tone.

The value of gb_start shall provide that any node shall observe the start of the moderator DLPDU no earlier than ((NUT_length - gb_start) × 10 + 20) μs ± CA after the previous tone in that node. This condition shall be met during a moderator change.

#### 8.2.4.9 Guardband center

The value of gb_center shall provide that any node shall observe the end of the moderator DLPDU no later than (NUT_length × 10 - 40) μs ± CA after the tone. This condition shall also be met during a moderator change.

#### 8.2.4.10 Useable time

Every node shall receive and process any valid DLPDU that begins between (tone + 20 μs) and (tone + (NUT_length - gb_start) × 10 μs ± CA), except that reception of a valid or invalid DLPDU shall not affect the timing of the transmission of the moderator DLPDU by the moderator node.

### 8.2.5 Slot timing

#### 8.2.5.1 Slot time

In the following description, the slots shall be numbered sequentially. The value of `slotTime` shall provide that a DLPDU transmitted in slot N is received in slot N at all other nodes assuming worst case conditions of

– crystal drift;
– signal propagation delay between nodes;
– MAC ID assignment;

but excluding noise events.

#### 8.2.5.2 First slot

Scheduled slot 0 shall begin at 20 μs after the tone.

#### 8.2.5.3 Missing node

If no DLPDU is sent or received during slot N-1, slot N shall begin at `slotTime` × 1 μs ± CA after the beginning of slot N-1.

#### 8.2.5.4 Restarting the slot timer

If a DLPDU is sent or received during slot N-1, slot N shall begin no later than 6,0 μs after the last M_Symbol of the end delimiter of the DLPDU. If a damaged DLPDU, with `ph_status_indication` not equal to Normal, is received during slot N-1, slot N shall begin no later than 6,0 μs after `ph_lock_indication` goes FALSE.

#### 8.2.5.5 Node turn around time

If both node N-1 and node N transmit in their respective slots, node N shall begin transmitting between 11,2 μs and 12,8 μs after the last M_Symbol of the end delimiter of the DLPDU transmitted by node N-1.

#### 8.2.5.6 Beginning transmission after missed node

If node N-1 does not transmit in its slot, and node N is enabled to transmit in its slot, node N shall begin transmitting no earlier than the start of slot N, and no later than 6,8 μs after the start of slot N.

The major factor to consider here is crystal drift. In the worst case, where there are two nodes attached to the network at MAC IDs 0 and 99, SMAX=98, UMAX=99, and USR=1, there can be a silence of 196 slot times between the scheduled DLPDU transmitted by node 0 and the unscheduled DLPDU transmitted by node 99. If node 99 has a fast crystal and node 0 has a slow crystal, or vice versa, the value of `slotTime` should be adequate to guarantee that, when node 99 transmits, node 0 agrees that the current slot is 99.

### 8.2.6 Blanking

The value of `blanking` shall be 6.

### 8.2.7 Example implementation

The requirements in 8.2.4, 8.2.5 and 8.2.6 are met (not optimally) in the following example program which calculates the link parameters:

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>

#define roundup(x,y) (((x) - 1L) / (y) + 1L) /* Round up integer division */

#define F_TURN      16060L    /* ns for firmware turnaround and detect      */
#define B_TURN       5720L    /* ns extra turnaround with large blanking value */
#define NULLFRAME       7L    /* octets for length of a null Lpacket transmission */
#define DELAY    4110L    /* ns per 1000 meters cable delay          */
#define RPT_DLY   815L    /* ns per repeater delay            */
#define PPMT     1500L    /* Max crystal error in tenths of PPM = +/- 150,0 ppm */

#define MODBEFORE     2    /* min moderator overhead before Lpacket (housekeeping)*/
#define MODRECOV  3    /* worst case min moderator overhead after any Lpacket */
#define MODTIME   5    /* ticks for the moderator plus error recovery if bad */
#define MINUNSCHED  850    /* minimum unscheduled time */

int main(int argc, char **argv)
   {

   /* Default values for command line input parameters.  These are
    * arbitrarily chosen for our typical convenient network sizes.
    */
   int dest  = 0xff;
   int board = -1;   /* If -1 use driver, else use XT version TX   */
   int redund   = 3;   /* Unspecified.  Later code defaults to 3 */
   long NUT_time= 10000L;
   int length   = 1000;
   int smax  = 7;
   int umax  = -1;
   int repeater = 0;
   int blank = 6;
   int maxfrm   = 255;
   int intmod   = 256;
   int terse = 0;
   int listenonly  = 0;
   int noleds   = 0;

   /* Other derived variables.  */

   int gap;
   long drift;
   long prop;
   long turn;
   long bslot;
   long slot;
   long picodev;
   int center;
   int start;
   int prestart;
   long unscheduled;
   long nuttenth;
   int nuthigh;
   int nutlow;
   char txhead[7];
   char *a;

   /* Parse arglist */

   argc--;
   argv++;
   if(argc==0)
      {
      argc=1;
      argv[0]="-h";
      }

   while(argc && (a=&argv[0][0],*a++=='-'))
      {
      while(*a)switch(*a++)
         {
      case 'x':
```

```
            board = atoi(a);
            if ((board < 0) || (board > 3))
               {
               printf("%%ERROR -- 0 <= board number <= 3.\n");
               return(1);
               }
            goto nextarg;

         case 'l':
            listenonly = 128;
            break;
         case 'n':
            noleds = 64;
            break;
         case 'b':
            redund = 1;
            break;
         case 'a':
            redund = 2;
            break;
         case 'r':
            redund = 3;
            break;
         case 't':
            terse = 1;
            break;

         case 'h':
         default:
            printf("Compute and print a net config Lpacket.\n\n");
            printf("NETCONF [-abdhlnrt] [-x<board>] [0x<dest>]"
               " [<NUT time> [<length> [<smax>\n");
            printf("        [<umax> [<repeaters> [<blanking> [<maxframe> "
               "[<int modulus>]]]]]]]\n");
            return(1);
            }
nextarg:  argc--;
      argv++;
      }

   /* Check for network address option */
   if (argc && argv[0][0] == '0' && argv[0][1] == 'x')
      {
      sscanf (argv[0],"%i",&dest);
      if ((dest < 0) || (dest > 255))
         {
         printf("%%ERROR --  0x00 <= destination address <= 0xff.\n");
         return(1);
         }
      argc--;
      argv++;
      }


   /* Parse remaining arguments.
    */
   switch (argc)
      {
   case 8:   intmod = atoi(argv[7]);
   case 7:   maxfrm = atoi(argv[6]);
   case 6:   blank = atoi(argv[5]);
   case 5:   repeater = atoi(argv[4]);
   case 4:   umax = atoi(argv[3]);
   case 3:   smax = atoi(argv[2]);
   case 2:   length = atoi(argv[1]);
   case 1:   NUT_time = atol(argv[0]);
      }


   /* check the validity of all the arguments */

   if ((intmod < 1) || (intmod > 256))
      {
      printf("%%ERROR --  1 <= interval count modulus <= 256.\n");
```

```
      return(1);
      }
   if ((maxfrm < 0) || (maxfrm > 255))
      {
      printf("%%ERROR --  0 <= max scheduled frame <= 255.\n");
      return(1);
      }
   if ((blank < 0) || (blank > 255))
      {
      printf("%%ERROR --  0 <= blanking <= 255.\n");
      return(1);
      }
   if (repeater < 0)
      {
      printf("%%ERROR --  number repeaters >= 0.\n");
      return(1);
      }
   if ((smax < 0) || (smax > 254))
      {
      printf("%%ERROR --  0 <= max scheduled addr <= 254.\n");
      return(1);
      }
   if (length < 0)
      {
      printf("%%ERROR --  cable length >= 0.\n");
      return(1);
      }
   if ((NUT_time < 350L) || (NUT_time > 100000L))
      {
      printf("%%ERROR --  350 <= nut time <= 100,000.\n");
      return(1);
      }

   if (umax == -1) umax = smax;
   if ((umax < smax) || (umax > 254))
      {
      printf("%%ERROR --  smax <= umax <= 254.\n");
      return(1);
      }
   if (NUT_time%10!=0)
      {
      printf("%%ERROR --  NUT length is specified in 10 µs intervals.\n");
      return(1);
      }

   /* Gap is the number of slot times that can pass in the normal
      protocol between transmissions.

      When smax<umax,then the maximum silence equals smax + umax – 1.
      The maximum silence occurs on the network with smax<umax when
      the first node is at 0, the only other is at umax with the USR==1.

      To illustrate a transmit sequence example when smax<umax,
      set smax=4 and umax=5, USR==1, with two nodes at 0 and 5.
      Slot times -> ssssuuuu   -> (smax + umax – 1)
                    01234123450

      When smax=umax, then the maximum silence equals smax + umax – 2.

      The maximum silence occurs on the network with smax = umax when
      the first node is at 0, the only other is at 1 with the USR===2.

      To illustrate a transmit sequence example when smax=umax,
      set smax=umax=5, USR==2, with two nodes at 0 and 1.
      Slot times -> ssssuuuu   -> (smax + umax – 2)
                    012345234501

      Units: slot times
   */

   gap = (smax < umax): smax + umax - 1: smax + umax - 2;
   if (gap < 0) gap = 0;
```

```
/* the drift is a correction factor needed to compensate for
   the amount of time that the fastest and slowest nodes drift
   apart during the longest silence due to crystal inaccuracy.

   Actual drift factor = (1-(2*gap+1)(ppm)) / (1-ppm^2)
   however, ppm^2 is so small it may be ignored.

   Units: unitless factor scaled by micro/pico
*/

drift = roundup(10000000L - (long)(2*gap+1) * PPMT , 10L);

/* Prop is the time the signal takes to travel down the cable and
   work its way through all the repeaters and modems.  It assumed
   on even a cable system that has no repeater boxes, that two
   two repeater exist to allow nodes connected via the NAP.

   Units: ps
*/
prop = (long)(length)*DELAY + (long)(repeater+2)*RPT_DLY*1000L;


/* turn is:

   - the amount of time it takes for the DLL to respond to a
   transmission, that is,  - the time from the end of the end delimiter
   to the start of the preamble

   plus

   - the time is takes another DLL to detect that the first *has*
   responded, that is,  - the time from the start of the preamble to the
   end of the start delimiter.

   Units: ns
*/
                /* turnaround could be */
turn = (long)(blank)*1600L + B_TURN;   /* <-- blanking constrained */
if (turn < F_TURN) turn = F_TURN;   /* <-- or firmware constrained */


/* bslot is the ideal slot time - which is usually defined as
   2 props + turnaround + detect (note that in this case, the
   detect time and the turn time are combined)
   Units: ps
*/

bslot = 2*prop + (turn*1000L);

/* slot is the ideal slot time adjusted to account for variances in
   crystal frequency.
   Units: µs
*/

slot = roundup(bslot , drift);
if ((slot < 1L) || (slot > 256L))
   {
   printf("%%ERROR --  Slot time is large, reduce network complexity.  \n");
   return(1);
   }

/* Calculate guardband parameters (round up to a multiple of a 10 µs tick clock).*/

/* calc nut time - units: 10 usec ticks */
nuttenth = roundup(NUT_time,10L);

/* picodev is the amount the clocks of the slowest and fastest
   node drifts apart during four NUTs
   Units: ps
*/
picodev = nuttenth * PPMT * 8L;


/* center is the time before the next tone that the moderator
```

```
      begins transmitting the moderator DLPDU.  This includes adjustments
      for clock drift during up to 4 NUTs, plus a moderator switchover
      from a near to a far node.
      Unit: 10 usec ticks
   */
   center = MODTIME + MODRECOV +
     (int)roundup(picodev + 2*prop , 10000000L);

   /* start is the time before the next tone that the guardband starts.
      Unit: 10 usec ticks
   */
   start = center + MODBEFORE +
     (int)roundup(picodev , 10000000L);


   /* prestart is the time before the next tone beyond which nodes may not
      transmit.  Due to clock skew and other factors a transmission which
      is sent such that it ends just at prestart may actually be received
      by another node such that the transmission ends just as the guardband
      starts.  The prestart margin is required in order to ensure that
      transmissions from slow nodes do not cross over into the guardbands
      of faster nodes.
      Unit: 10 usec ticks
   */
   prestart = (int)roundup((nuttenth*PPMT*2L) + ((NULLFRAME+blank)*1600000L)
     + bslot , 10000000L) + start;

   /* Determine if nut parameters is practical and print warnings.
    * Initial calculations are conservative due to roundup of several
    * parameters.  Usage of fixround helps correct it.
    */

   /* calc the max length of unscheduled time assuming that all
     scheduled nodes are silent.
     Units: µs
   */
   unscheduled = NUT_time - max(roundup(NULLFRAME*1600000L+prop+turn*1000L,
                  1000000000L),slot)*(long)(smax+1)
                              - (long)(prestart)*10L;
   if (unscheduled < MINUNSCHED)
      printf("%%WARNING --  Not enough scheduled time available.  Increase nut
time.\n");

   /* Print statistics and the transmit command.
    */
   nutlow = nuttenth & 0xff;
   nuthigh = (nuttenth >> 8) & 0xff;
   if(!terse)
      {
      long deviation = roundup(picodev , 1000000L);

      printf("slot = %ld usec, max unscheduled time = %ld usec\n",
         slot,unscheduled);
      printf("dev = %ld usec, prestart = %d ticks, start = %d ticks, center = %d
ticks\n",
         deviation,prestart,start,center);
      }
   if (board == -1) sprintf(txhead,"tx");
   else sprintf(txhead,"txt /%d",board);
   printf(   "%s m a 17 %02x %02x %02x %02x %02x %02x %02x “
      ”%02x %02x %02x %02x %02x %02x 00 00 00\n",
      txhead,dest,nutlow,nuthigh,smax,umax,(int)slot-1,blank,
      start,center,(listenonly|noleds|redund),maxfrm,intmod-1,prestart);
   return(0);
   }
```

# 9 Detailed specification of DL components

## 9.1 General

Clause 9 specifies each main functional component of the DL internal architecture as shown in Figure 2 using the modeling language of 6.1.

## 9.2 Access control machine (ACM)

The Access Control Machine (ACM) shall control the sequencing of transmissions by this node.

```
// ACM State Machine Description

////////////////////////////////////////////////////////////////////////
// constant and type definitions

//
// generic type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;

//
// protocol constants
//
#define LOWCOUNTINIT      2  // NUTs as lowman before switchover to moderator
#define MODERATOR_LENGTH 40  // length of the moderator DLPDU in usec
#define TXDUPCHECKS       3  // moderators that have to be heard before transmit at
powerup
#define TXNOMOD           5  // number of missed moderators before disabling transmit
#define DEAFNESS          5  // number of missed moderators before adjusting phase
#define DEAFADJUST        5  // 10 usec ticks to slip NUT if deafness is detected
#define LONELYINIT        8  // NUTs that are tolerated without hearing anything
             // before becoming lonely (should be longer than DEAFNESS
             // to ensure time for deaf recovery)
////////////////////////////////////////////////////////////////////////
// Lpacket definition

//
// Lpacket constants: masks for control octet
//
#define FIXEDSCREEN       1
#define TAGPAD            2
#define DATAPAD           4

//
// moderator Lpacket constants
//
#define MODERATOR_SIZE    9  // moderator Lpacket size octet (in 16 bit words)
#define MODERATOR_CTL     1  // moderator Lpacket ctl octet (fixed tag)
#define MODERATOR_TAG     0  // moderator Lpacket service octet

//
// Lpacket class
//
class Lpacket
   {
   public:

   USINT size;  // the size octet
   USINT ctl;   // the ctl octet

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }

   // return the value of the data pad bit
   int data_pad(void)
      {
```

```
      return (ctl&DATAPAD) >>2;
      }

   // return the size of the Lpacket (in octets)
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }

   // get the next octet to be transmitted
   USINT get_next_octet(void);

   // is this Lpacket aborted?
   BOOL abort(void);
   };

//
// fixed tag Lpacket (two octet tag)
//
class fixedLpacket: public Lpacket
   {
   public:

   USINT  service;
   USINT  dest;
   };

//
// moderator fixed tag Lpacket
//
class moderatorLpacket: public fixedLpacket
   {
   public:

   UINT   NUT_length;
   USINT  smax;
   USINT  umax;
   USINT  slotTime;
   USINT  blanking;
   USINT  gb_start;
   USINT  gb_center;
   USINT  usr;
   USINT  interval_count;
   USINT  modulus;
   USINT  tMinus;
   USINT  gb_prestart;
   USINT  spare;
   };


////////////////////////////////////////////////////////////////////////////
//
// internal VARIABLES
//
// unless otherwise specified, all variables are initialized to zero at powerup

int tmp;
USINT  interval_count;  // counts how many NUTs have passed (0 – modulus)
USINT  itr;          // implicit token register - tracks which MAC should xmit
USINT  usr;          // unscheduled start register -- first unscheduled token
USINT  tMinus;          // countdown to synchronized change
USINT  lowcount;     // how many times have I been lowman?
USINT  modcount;     // counts moderators (or missed moderators) depends on netState
USINT  modcountinit;    // how many NUTs to check for dupnode
USINT  nqlowman;     // optimistic lowman detector
USINT  qlowman;      // pessimistic lowman detector
USINT  lonely;          // how many NUTs have I been lonely
USINT  deafcount;       // number of times moderator apparently occurred during deaf
period
BOOL   in_guardband;    // TRUE during the guardband
BOOL   dupflag;      // edge detector for dupnode
BOOL   scheduled;       // scheduled/unscheduled part of the NUT
UINT   octetsleft;      // octets / octets left before end of current transmitted DLPDU
```

```
USINT  macSrce;     // source MAC ID for the current DLPDU
USINT  currentMod;      // the current moderator
moderatorLpacket moderator;  // buffer for moderator Lpackets


enum { OFFLINE,
   WATCH,
   DUPCHECK,
   NOTMOD,
   MOD,
   ROGUE,
   DUPNODE
   } netState; // connection state



////////////////////////////////////////////////////////////////////////
// timer definition
//
// various behaviors for timers
typedef enum {ONCE_UP, ONCE_DOWN, REPEAT_UP, REPEAT_DOWN} timer_mode;

class timer
   {
   public:
   float  count;    // number of ticks left
   float  preset;       // init ticks, or cycle length
   BOOL   expire;       // latched flag when timer expires
   timer_mode mode;

   // constructor: defines mode
   timer(timer_mode m)
      {
      mode = m;
      }

   void restart(void)
      {
      // if counting down, start with preset
      if(mode == REPEAT_DOWN || mode == ONCE_DOWN)
         {
         count = preset;
         }
      else
         {
         count = 0;   // else start from zero
         }
      // start counting (implementation not specified)
      }

   void start(float interval)// start counting down from interval towards zero
      {
      mode = ONCE_DOWN;
      preset = interval;  // set interval
      restart();       // start counting
      }

   void begin_counting(void) // start counting up from zero
      {
      mode = ONCE_UP;
      restart();
      }


   bool expired(void)     // returns true if the timer has expired; clears flag
      {
      BOOL tmp;

      tmp = expire;
      expire = 0;
      return tmp;
      }

   };
```

```
//
// time conversion factors
//
#define usec
#define msec *1000
#define sec *1000000

//
// define the timers used by the ACM
//
class timer gen_timer(ONCE_DOWN);   // general purpose, in several places
class timer slot_timer(REPEAT_DOWN);   // used for response time-out between messages
class timer watch_timer(ONCE_DOWN); // used for watch time state
class timer NUT_timer(REPEAT_DOWN); // this generates the NUT timing
class timer holdoff_timer(ONCE_DOWN);  // holds off transmit for a timer after
transmit or receive


////////////////////////////////////////////////////////////////////////////
//
// interface to PhL
//
BOOL ph_lock_indication;  // optimistic DLPDU detect (clock recovery is tracking)
BOOL ph_frame_indication; // pessimistic DLPDU detect (Manchester valid since the SD)


////////////////////////////////////////////////////////////////////////////
//
// interface to Deserializer
//
BOOL RX_dataReady;     // an octet of a DLPDU is available
USINT RX_rxData;    // octet most recently received from DLPDU (source MAC ID)
BOOL RX_endMAC;     // the end delimiter has been detected
BOOL RX_FCSOK;         // FCS on last DLPDU was OK
BOOL RX_abort;         // abort received on last DLPDU


////////////////////////////////////////////////////////////////////////////
//
// interface to RxM
//
BOOL RX_receivedLpacket;  // an Lpacket has been received
moderatorLpacket RX_Lpacket; // the Lpacket most recently received


////////////////////////////////////////////////////////////////////////////
//
// interface to transmitter (TxM)
//
void TX_sendHeader(USINT myaddr);   // send preamble, SD, source MAC ID
void TX_sendLpacket(Lpacket &lp);   // send an Lpacket
void TX_sendTrailer(void);    // send FCS, ED
BOOL TX_headerComplete;       // header has been sent
BOOL TX_LpacketComplete;      // Lpacket has been sent
BOOL TX_trailerComplete;      // trailer has been sent
BOOL TX_abort;            // DLPDU was aborted


////////////////////////////////////////////////////////////////////////////
//
// interface to LLC
//
Lpacket LLC_Lpacket;          // shared variable between LLC and ACM containing the
Lpacket
BOOL LLC_pickLpacket (BOOL scheduled,
              UINT octetsLeft);// tell LLC to pick an Lpacket for transmission
LLC_LpacketSent(void);        // tell LLC that the Lpacket was sent
////////////////////////////////////////////////////////////////////////////
//
// interface to station management
```

```
//

//
// various events that can be reported to SM
//
typedef enum
   {
   DLL_EV_rxGoodFrame,
   DLL_EV_txGoodFrame,
   DLL_EV_badFrame,
   DLL_EV_errA,
   DLL_EV_errB,
   DLL_EV_txAbort,
   DLL_EV_NUT_overrun,
   DLL_EV_dribble,
   DLL_EV_nonconcurrence,
   DLL_EV_rxAbort,
   DLL_EV_lonely,
   DLL_EV_dupNode,
   DLL_EV_noisePulse,
   DLL_EV_collision,
   DLL_EV_invalidModAddress,
   DLL_EV_rogue,
   DLL_EV_deafness,
   DLL_EV_supernode,
   }event;

extern void  DLL_event(event ev, int opt=0);     // report an event
extern void  DLL_currentMod_indication(USINT mac_ID);
extern void  DLL_tminus_zero_indication(void);

extern void  DLL_online_confirm(BOOL en);
extern void  DLL_listen_only_confirm(BOOL en);
extern void  DLL_tone_indication(USINT NUT_count);

class DLL_config_data
   {
   public:

   USINT  myaddr;

   UINT   NUT_length;
   USINT  smax;
   USINT  umax;
   USINT  slotTime;
   USINT  blanking;
   USINT  gb_start;
   USINT  gb_center;
   USINT  interval_count;
   USINT  modulus;
   USINT  gb_prestart;
   };


// interface to DLL management interface

extern DLL_config_data DLL_SM_pending;
extern DLL_config_data DLL_SM_current;

extern BOOL SM_powerup;

extern BOOL  SM_mod_enable;      // state of mod_enable
extern BOOL  SM_online;      // state of on-line/off-line
extern BOOL  SM_online_req;      // pending state of on-line/off-line
extern BOOL  SM_listen_only; // listen only state
extern BOOL  SM_listen_only_req; // pending listen only state



////////////////////////////////////////////////////////////////////////////
//
// subroutines
//
//
```

```
// take care of common processing for network parameter changes
//
void handle_net_change(void)
   {
   if(SM_listen_only != SM_listen_only_req)  // handle completion of listen_only
change
      {
      SM_listen_only = SM_listen_only_req;
      DLL_listen_only_confirm(SM_listen_only);
      }

   if(tMinus==1)            // handle completion of synchronized change
      {
      tMinus = 0;
      DLL_tminus_zero_indication();
      }
   }



//
// maintenance work done whenever the moderator is received
//
void processModerator(moderatorLpacket &moderator)
   {
   //
   // copy data from the moderator
   // which cannot cause a rogue condition
   //
   interval_count = moderator.interval_count;
   usr = moderator.usr;
   tMinus = moderator.tMinus;

   lowcount = LOWCOUNTINIT;  // re-initialize lowman counter

   // resynchronize the NUT timer to the moderator
   NUT_timer.count = DLL_SM_current.gb_center - MODERATOR_LENGTH/10;

   if(netState == MOD)             // if this node is moderator
      {                  // and a moderator was received!
      netState = NOTMOD;           // drop moderatorship
      modcount = 0;             // init count of missing moderators
      DLL_event(DLL_EV_nonconcurrence);      // indicate non-concurrence
      }
   else
      {
      if(!in_guardband)            // unexpected moderator
         {
         DLL_event(DLL_EV_nonconcurrence);   // indicate non-concurrence
         }

      if(netState == WATCH)           // got moderator, there is a network
         {
         netState = DUPCHECK;         // go check for dupnodes
         modcount = modcountinit;     // this is how many NUTs to check
         }
      else if(netState == DUPCHECK)       // if checking for duplicate MAC IDs
         {
         if(SM_listen_only == FALSE)      // unless this node is required to
            {            //    remain in this state
            modcount--;         // decrement NUT counter
            if(modcount == 0)      // when done
               {
               netState = NOTMOD;  // it's OK to go transmit
               }
            }
         }
      else if(netState == ROGUE)       // a rogue that sees a good moderator
         {
         netState = DUPCHECK;         // can exit rogue state
         modcount = modcountinit;
         }
      else
         {
```

```
            modcount = 0;       // else just reset mod counter
            }
        }
    }


//
// recovery after a time-out while waiting for moderator
//
void missed_moderator(void)
    {
    if(netState == NOTMOD)        // if supposed to be on-line and hearing moderators
        {
        modcount++;               // count consecutive missing moderator
        if(modcount >= TXNOMOD)       // if exceeded limit
            {
            netState = DUPCHECK;      // change to a quietly waiting mode
            modcount = 1;       // init mods required before return on-line
            }

        if(SM_listen_only == FALSE    // if allowed to transmit
           && SM_mod_enable == TRUE      //   and allowed to be moderator
           && DLL_SM_current.myaddr <= nqlowman)  //   and apparently the lowman
            {
            lowcount--;         // count consecutive NUTs as lowman
            if(lowcount == 0)       // if this is the second time
                {
                netState = MOD; // then activate the moderator on third NUT
                }
            }
        else
            {
            lowcount = LOWCOUNTINIT;  // otherwise, re-initialize lowman counter
            }
        }
    }


//
// housekeeping actions at the end of each NUT
//
void housekeeping(void)
    {
    if(netState == WATCH)           // in watch state
        {
        lonely = LONELYINIT;          // hold the lonely counter reset
        if(watch_timer.count == 0)    // if timed out (no moderators)
            {           // change to on-line -- try to build a link
            if (SM_listen_only == FALSE  // if allowed to transmit
                && SM_mod_enable == TRUE //   and allowed to moderate
                && nqlowman >= DLL_SM_current.myaddr) //   and lowman
                {
                netState = MOD; // activate moderator state
                }
            else
                {
                netState = DUPCHECK;   // else wait for another node to moderate
                modcount = modcountinit;
                }
            }

        }
    else if (qlowman == 255         // no good FCS received in the last NUT
         && DLL_SM_current.myaddr != 0   // local node is not supernode
         && netState != WATCH)       // not already in the watch state
        {
        lonely--;
        if(lonely == 0 || netState == ROGUE || netState == DUPNODE)
            {
            DLL_event(DLL_EV_lonely);
            if(netState == ROGUE)
                {
```

```
            watch_timer.start(0,75 sec);
            }
         else
            {
            watch_timer.start(1,25 sec);
            }
         netState = WATCH;
         }
      }
   else
      {
      lonely = LONELYINIT;
      }

   if(deafcount >= DEAFNESS)
      {
      deafcount = 0;
      DLL_event(DLL_EV_deafness);
      NUT_timer.count += DEAFADJUST;
      }

   // this counter determines how many NUTS a node should
   // check the link before deciding that it is not
   // a dupnode.  if 0<myaddr<=SMAX a node is guaranteed to
   // an access each NUT, so the check time is low.  On the
   // other hand, if smax<myaddr<=umax, a node might have to wait
   // a long time for its access slot come up, so use a long
   // time.  If a DUPCHECKing unscheduled only node ever sees its slot
   // go by, it sets modcount to a lower number immediately.

   // nodes between SMAX and UMAX aren't guaranteed to get a slot every NUT
   // if not an unscheduled only node, use a low number else use a huge number

   if (DLL_SM_current.myaddr > DLL_SM_current.smax)
      modcountinit = 255;
   else
      modcountinit =TXDUPCHECKS;

   // update the interval count once per NUT
   interval_count = (interval_count + 1) % DLL_SM_current.modulus;

   // update the usr once per NUT
   usr = (usr + 1) % (DLL_SM_current.umax + 1);
   }

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//
// The ACM State Machine
//
state: powerup

   event:          SM_powerup
   destination:    offline

//
// be idle until told to go on-line
//
state: offline

   // normal case
   event:          SM_online == TRUE
   condition:      ph_frame_indication == FALSE
   destination:    rxMod0
   action:
      in_guardband = TRUE;           // start up at the beginning of a guardband
      if(DLL_SM_current.myaddr == 0) // either local node is supernode
         {
         netState = MOD;
         SM_listen_only = FALSE;
         }
      else                           // or not a supernode
         {
```

```
            netState = WATCH;
            watch_timer.start(1,25 sec);
            }
         scheduled = FALSE;
         in_guardband = TRUE;
         DLL_online_confirm(SM_online);


      // exception: there was an Lpacket on the wire at transition to on-line
      event:          SM_online == TRUE
      condition:      ph_frame_indication == TRUE
      destination:    badMod                // treat it as an error inside the guardband
      action:
         DLL_event(DLL_EV_badFrame,macSrce);
         in_guardband = TRUE;
         if(DLL_SM_current.myaddr == 0)
            {
            netState = MOD;
            SM_listen_only = FALSE;
            }
         else
            {
            netState = WATCH;
            watch_timer.start(1,25 sec);
            }
         DLL_online_confirm(SM_online);



//
// wait for either energy on the wire or a slot time-out
//
state: waitFrame

   event:    ph_lock_indication == TRUE
   destination: frEst1
   action:      gen_timer.begin_counting();

   event:    slot_timer.expired()
   destination: gap
   action:      gen_timer.start(0,6 usec);

//
// wait for frame start, or noise
//
state: frEst1
   event:    ph_frame_indication == TRUE
   destination: rxFrame

   event:    ph_lock_indication == FALSE      // short energy burst == noise blip
   destination: waitFrame
   action:      DLL_event(DLL_EV_noisePulse);    // ignore it

   event:    gen_timer.count >= 8,0 usec
   destination: frEst2



//
// continue waiting
//
state: frEst2

   event:    ph_frame_indication == TRUE
   destination: rxFrame

   event:    ph_lock_indication == FALSE      // longer burst treat as a damaged DLPDU
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);

   event:    gen_timer.count >= 11,2 usec     // energy this long with no data start
   destination: endFrame           // treat as bad DLPDU, no start delimiter
```

```
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);

//
// wait here for there to be no detectable energy on the wire (no clock lock)
//
state: endFrame

   event:    ph_lock_indication == FALSE
   destination: gap
   action:
      slot_timer.restart();

//
// simulate first half of node turnaround time with this timing gap
//
state: gap

   event:    ph_frame_indication == TRUE  // DLPDU start overrides all
   destination: rxFrame

   event:    gen_timer.expired()    // process the rest of
   destination: nextSlot
   action:
      gen_timer.start(6,1 usec);    // start second half of turnaround

      if(scheduled)          // do implicit token rotation, scheduled
         {
         itr++;
         if(itr>DLL_SM_current.smax)  // after SMAX,
            {         // change from scheduled to unscheduled
            itr=usr;
            scheduled = FALSE;
            }
         }
      else               // implicit token rotation, unscheduled
         {
         itr = (itr + 1) % (DLL_SM_current.umax + 1);
         }

//
// decide what comes next
//
state: nextSlot

   event:    ph_frame_indication == TRUE      // detect a DLPDU start at all times
   destination: rxFrame

   // time for the guardband
   event:    gen_timer.expired()
   condition:   NUT_timer.count < DLL_SM_current.gb_prestart
   destination: guardBand
   action:

      // scheduled time not completed -- log the error and proceed
      if(scheduled == TRUE)
         {
         DLL_event(DLL_EV_NUT_overrun);
         scheduled = FALSE;
         }
      in_guardband = TRUE;

   // not this node's slot, keep counting.
   // Note that the slot timer auto repeats to maintain regular slot intervals
   event:    gen_timer.expired()
   condition:   NUT_timer.count >= DLL_SM_current.gb_prestart
         && itr != DLL_SM_current.myaddr
   destination:    waitFrame

   // this node's slot, but not allowed to talk
   event:    gen_timer.expired()
   condition:   NUT_timer.count >= DLL_SM_current.gb_prestart
```

```
          && itr == DLL_SM_current.myaddr
          && !(netState == MOD || netState == NOTMOD)
   destination: waitFrame
   action:      modcount = min(modcount, TXDUPCHECKS); // optimize dupcheck

   // this node's slot, and allowed to talk
   event:    gen_timer.expired()
   condition:   NUT_timer.count >= DLL_SM_current.gb_prestart
          && itr == DLL_SM_current.myaddr
          && (netState==MOD || netState==NOTMOD)
   destination: waitForHold


//
// a DLPDU has started, this deals with the data
//
state: rxFrame

   // prematurely lost the DLPDU event
   event:    ph_frame_indication == FALSE
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);

      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // duplicate node condition
   event:    RX_dataReady
   condition:   RX_rxData == DLL_SM_current.myaddr
   destination: dupNode
   action:
      macSrce = RX_rxData;
      nqlowman = min(nqlowman,macSrce);

   // record the source MAC ID
   event:    RX_dataReady
   condition:   RX_rxData != DLL_SM_current.myaddr
   destination: nextLpacket
   action:
      macSrce = RX_rxData;
      nqlowman = min(nqlowman,macSrce);

//
// start here to parse each new Lpacket
// the RxM delivers one complete Lpacket at a time
//
state: nextLpacket

   // received an out-of-sequence moderator -- attempt resync
   event:    RX_receivedLpacket
   condition:   RX_Lpacket.service == MODERATOR_TAG
   destination: rxMod3              // unexpected moderator

   // ignore most other received Lpackets
   event:    RX_receivedLpacket
   condition:   RX_Lpacket.service != MODERATOR_TAG
   destination: nextLpacket

   // lost the DLPDU in the middle
   event:    ph_frame_indication == FALSE
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // the DLPDU was aborted
   event:          RX_abort
   destination:    endFrame
   action:
      DLL_event(DLL_EV_rxAbort);
```

```
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // if bad FCS, or DLPDU extended into the guardband, treat as bad DLPDU
   event:    RX_endMAC
   condition:   NUT_timer.count <= DLL_SM_current.gb_start || !RX_FCSOK
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // good DLPDU, and not guardband time
   // use the source ID
   event:    RX_endMAC
   condition:   NUT_timer.count > DLL_SM_current.gb_start && RX_FCSOK
   destination: endFrame
   action:
      qlowman = min(qlowman, macSrce);
      DLL_event(DLL_EV_rxGoodFrame);
      if(itr != macSrce) DLL_event(DLL_EV_nonconcurrence);
      itr = macSrce;
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// deal with a duplicate node indication, local MAC ID already in by another node
//
state: dupNode

   // bad DLPDU so ignore indication
   event:    ph_frame_indication == FALSE
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // bad DLPDU so ignore indication
   event:    RX_endMAC
   condition:   !RX_FCSOK
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // good DLPDU so accept the duplicate indication
   event:    RX_endMAC
   condition:   RX_FCSOK
   destination: endFrame
   action:
      netState = DUPNODE;
      if (!dupflag)
         {
         dupflag == TRUE;
         DLL_event(DLL_EV_dupNode);
         }
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// Hold off transmission until blanking + 1 octet has expired
// since the last transmit or receive.  It is possible that the
// hold timer expires before the gen_timer, dealing with turnaround time.
//
state: waitForHold
```

```
   // time to transmit, but line has activity, treat it as a collision, and back off.
   event:    holdoff_timer.expired()
   condition:   ph_lock_indication == TRUE
   destination: endFrame
   action:
      DLL_event(DLL_EV_collision);
      gen_timer.start(0,6 usec);

   // no problem, start transmission
   event:    holdoff_timer.expired()
   condition:   ph_lock_indication == FALSE
   destination: sendHeader
   action:
      // calculate time remaining in this NUT
      tmp = (unsigned)(NUT_timer.count-DLL_SM_current.gb_prestart);
      octetsleft = min(255, (tmp<<1) + tmp + (tmp>>3)) * 2;
      TX_sendHeader(DLL_SM_current.myaddr);

//
// when header has been sent, start first Lpacket
//
state: sendHeader

   // TxLLC has something, send a new Lpacket
   event:    TX_headerComplete
   condition:   LLC_pickLpacket(scheduled, octetsleft) == TRUE
   destination: sendNextLpacket
   action:
      TX_sendLpacket(LLC_Lpacket);
      octetsleft = octetsleft - LLC_Lpacket.wire_size();

   // nothing to send that fits in the time left, close the DLPDU
   event:    TX_headerComplete
   condition:   LLC_pickLpacket(scheduled,octetsleft) == FALSE
   destination: sendTrailer
   action:
      TX_sendTrailer();
      if(scheduled && fifo[SCHEDULED].has_data())
         {
         DLL_event(DLL_EV_dribble);
         }

//
// send subsequent Lpackets
//
state: sendNextLpacket

   // last Lpacket was aborted, end frame
   // note that the TXM has already sent the abort sequence on the wire
   event:       TX_LpacketComplete
   condition:   Tx_abort==TRUE
   destination: endFrame
   action:
      DLL_event(DLL_EV_txAbort);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // TxLLC has something, send a new Lpacket
   event:    TX_LpacketComplete
   condition:   (TX_abort == FALSE) && (LLC_pickLpacket(scheduled,octetsleft) == TRUE)
   destination: sendNextLpacket
   action:
      TX_sendLpacket(LLC_Lpacket);
      octetsleft = octetsleft - LLC_Lpacket.wire_size();

   // nothing to send that fits in the time left, close the DLPDU
   event:    TX_LpacketComplete
   condition:   (TX_abort == FALSE)&&(LLC_pickLpacket(scheduled,octetsleft) == FALSE)
   destination: sendTrailer
   action:
      TX_sendTrailer();
```

```
//
// complete DLPDU termination
//
state: sendTrailer

   event:    TX_trailerComplete
   destination: endFrame
   action:
      DLL_event(DLL_EV_txGoodFrame);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// almost guardband start time, wait for guardband start then check lots of things
//
state: guardBand

   event:    ph_frame_indication == TRUE      // received DLPDU takes precedence
   destination: rxFrame                // deal with it

   // net change not in progress, this node is not moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start // guardband start time NOW
   condition:   tMinus == 0
          && SM_listen_only == SM_listen_only_req
          && netState != MOD
   destination: rxMod0

   // net change not in progress, this node is moderator AND can remain moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus == 0
          && SM_listen_only == SM_listen_only_req
          && netState == MOD
          && SM_mod_enable == TRUE
          && min(DLL_SM_current.myaddr, qlowman) == DLL_SM_current.myaddr
   destination: txMod0

   // net change not in progress, this node is moderator, but cannot remain moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus == 0
          && SM_listen_only == SM_listen_only_req
          && netState == MOD
          && (SM_mod_enable == FALSE
              || min(DLL_SM_current.myaddr, qlowman) != DLL_SM_current.myaddr)
   destination: rxMod0
   action:
      netState = NOTMOD;
      modcount = 0;
      lowcount = LOWCOUNTINIT;

   // net change in progress, this node is not moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus > 1 && SM_listen_only == SM_listen_only_req && netState != MOD
   destination: rxMod0
   action:
      tMinus = tMinus - 1;

   // net change in progress, this node is moderator, and can remain moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus > 1
          && SM_listen_only == SM_listen_only_req
          && netState == MOD
          && SM_mod_enable == TRUE
          && min(DLL_SM_current.myaddr, qlowman) == DLL_SM_current.myaddr
   destination: txMod0
   action:
      tMinus = tMinus - 1;

   // net change in progress, this node is moderator, but cannot remain moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus > 1
          && SM_listen_only == SM_listen_only_req
          && netState == MOD
```

```
          && (SM_mod_enable == FALSE
              || min(DLL_SM_current.myaddr, qlowman) != DLL_SM_current.myaddr)
   destination: rxMod0
   action:
      tMinus = tMinus - 1;
      netState = NOTMOD;
      modcount = 0;
      lowcount = LOWCOUNTINIT;

   // net change complete, this node becomes supernode
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus == 1 && DLL_SM_pending.myaddr == 0
   destination: txMod0
   action:
      handle_net_change();
      netState = MOD;
      SM_listen_only = FALSE;          // override listen_only

   // net change complete, but shall stop transmitting for a time
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   (       tMinus == 1
              && DLL_SM_pending.myaddr != 0
              && DLL_SM_pending.myaddr != DLL_SM_current.myaddr
            ||    SM_listen_only != SM_listen_only_req
              && SM_listen_only_req)
   destination: rxMod0
   action:
      handle_net_change();
      netState = WATCH;          // go to WATCH state
      watch_timer.start(1,25 sec);

   // net change complete, this node is not moderator, no significant changes
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   (       tMinus == 1
              && DLL_SM_pending.myaddr != 0
              && DLL_SM_pending.myaddr == DLL_SM_current.myaddr
            ||    SM_listen_only != SM_listen_only_req
              && !SM_listen_only_req)
          && netState != MOD
   destination: rxMod0
   action:
      handle_net_change();

   // net change complete, this node is moderator, no major changes, and can remain
moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   (       tMinus == 1
              && DLL_SM_pending.myaddr != 0
              && DLL_SM_pending.myaddr == DLL_SM_current.myaddr
            ||    SM_listen_only != SM_listen_only_req
              && !SM_listen_only_req)
          && netState == MOD
          && SM_mod_enable == TRUE
          && min(DLL_SM_pending.myaddr, qlowman) == DLL_SM_pending.myaddr
   destination: txMod0
   action:
      handle_net_change();

   // net change complete, this node is moderator, no major changes, but can't stay
moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   (       tMinus == 1
              && DLL_SM_pending.myaddr != 0
              && DLL_SM_pending.myaddr == DLL_SM_current.myaddr
            ||    SM_listen_only != SM_listen_only_req
              && !SM_listen_only_req)
          && netState == MOD
          && (SM_mod_enable == FALSE
              || min(DLL_SM_pending.myaddr, qlowman) != DLL_SM_pending.myaddr)
   destination: rxMod0
   action:
      handle_net_change();
      netState = NOTMOD;
      modcount = 0;
```

```
       lowcount = LOWCOUNTINIT;


//
// wait for moderator start
//
state: rxMod0

   // got it
   event:    ph_frame_indication == TRUE
   destination: rxMod1

   // timed out waiting for moderator
   event:    NUT_timer.count <= 20 usec
   destination: waitTone
   action:
      missed_moderator();
      housekeeping();

//
// receive the moderator
//
state: rxMod1

   // bad moderator
   event:    ph_frame_indication == FALSE
   destination: badMod
   action:
      DLL_event(DLL_EV_badFrame,macSrce);

   // get MAC source ID
   event:    RX_dataReady == TRUE
   destination: rxMod2
   action:
      macSrce = RX_rxData;
      nqlowman = min(nqlowman,macSrce);

//
// continue receiving moderator
//
state: rxMod2

   // bad DLPDU
   event:    ph_frame_indication == FALSE
   destination: badMod
   action:
      DLL_event(DLL_EV_badFrame,macSrce);

   // bad moderator DLPDU if it's null
   event:    RX_endMAC
   destination: badMod
   action:
      DLL_event(DLL_EV_badFrame,macSrce);

   // got an Lpacket, save the Lpacket for later, and go check the rest of the DLPDU
   event:    RX_receivedLpacket
   destination: rxMod3

//
// handle the end of a moderator DLPDU
//
state: rxMod3

   // bad DLPDU
   event:    ph_frame_indication == FALSE
   destination: badMod
   action:
      DLL_event(DLL_EV_badFrame,macSrce);

   // bad DLPDU
   event:    RX_abort
   destination: badMod
   action:
```

```
   DLL_event(DLL_EV_rxAbort);

// bad DLPDU if another Lpacket is in the moderator DLPDU
event:    RX_receivedLpacket
destination: badMod
action:
   DLL_event(DLL_EV_badFrame,macSrce);

// bad DLPDU
event:    RX_endMAC
condition:   !RX_FCSOK
destination: badMod
action:
   DLL_event(DLL_EV_badFrame,macSrce);

// dupnode!
event:    RX_endMAC
condition:   RX_FCSOK && macSrce == DLL_SM_current.myaddr
destination: endFrame
action:
   netState = DUPNODE;
   if(!dupflag)
      {
      dupflag = TRUE;
      DLL_event(DLL_EV_dupNode);
      }
   gen_timer.start(5,2 usec);
   // start hold timer after line is quiet
   holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// good DLPDU, but bad header
event:    RX_endMAC
condition:   RX_FCSOK
      && macSrce != DLL_SM_current.myaddr
      &&  (   RX_Lpacket.size != MODERATOR_SIZE
           || RX_Lpacket.ctl != MODERATOR_CTL
           || RX_Lpacket.service != MODERATOR_TAG
           || RX_Lpacket.dest != 0xFF)
destination: badMod

// moderator, but from an incorrect address (not lowman)
event:    RX_endMAC
condition:   RX_FCSOK
      && macSrce != DLL_SM_current.myaddr
      && RX_Lpacket.size == MODERATOR_SIZE
      && RX_Lpacket.ctl == MODERATOR_CTL
      && RX_Lpacket.service == MODERATOR_TAG
      && RX_Lpacket.dest == 0xFF
      && macSrce > qlowman
destination: endFrame
action:
   DLL_event(DLL_EV_invalidModAddress);
   gen_timer.start(5,2 usec);
   // start hold timer after line is quiet
   holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// this is a moderator from a supernode,
// or this node is at 0xFF so that all others look like a supernode
// adopt all the parameters from the moderator
event:    RX_endMAC
condition:   RX_FCSOK
      && macSrce != DLL_SM_current.myaddr
      && RX_Lpacket.size == MODERATOR_SIZE
      && RX_Lpacket.ctl == MODERATOR_CTL
      && RX_Lpacket.service == MODERATOR_TAG
      && RX_Lpacket.dest == 0xFF
      && macSrce <= qlowman
      && (macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
destination: waitTone
action:
   qlowman = min(qlowman, macSrce);
   if (macSrce == 0 && currentMod != 0) DLL_event(DLL_EV_supernode);
   currentMod = macSrce;
   DLL_currentMod_indication(currentMod);
```

```
      DLL_SM_current.NUT_length = moderator.NUT_length;
      DLL_SM_current.gb_prestart = moderator.gb_prestart;
      DLL_SM_current.gb_start = moderator.gb_start;
      DLL_SM_current.gb_center = moderator.gb_center;
      DLL_SM_current.modulus = moderator.modulus;
      DLL_SM_current.blanking = moderator.blanking;
      DLL_SM_current.slotTime = moderator.slotTime;
      DLL_SM_current.smax = moderator.smax;
      DLL_SM_current.umax = moderator.umax;
      processModerator(moderator);
      housekeeping();

   // good moderator received, everything matches local copy
   // resynchronize and continue
   event:    RX_endMAC
   condition:   RX_FCSOK
          && macSrce != DLL_SM_current.myaddr
          && RX_Lpacket.size == MODERATOR_SIZE
          && RX_Lpacket.ctl == MODERATOR_CTL
          && RX_Lpacket.service == MODERATOR_TAG
          && RX_Lpacket.dest == 0xFF
          && macSrce <= qlowman
          && !(macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
          && moderator.NUT_length == DLL_SM_current.NUT_length
          && moderator.gb_prestart == DLL_SM_current.gb_prestart
          && moderator.gb_start == DLL_SM_current.gb_start
          && moderator.gb_center == DLL_SM_current.gb_center
          && moderator.modulus == DLL_SM_current.modulus
          && moderator.blanking == DLL_SM_current.blanking
          && moderator.slotTime == DLL_SM_current.slotTime
          && moderator.smax == DLL_SM_current.smax
          && moderator.umax == DLL_SM_current.umax
   destination: waitTone
   action:
      qlowman = min(qlowman, macSrce);
      currentMod = macSrce;
      DLL_currentMod_indication(currentMod);
      processModerator(moderator);
      housekeeping();

   // The moderator doesn't match, so this node is a rogue.
   // Since in the guardband, go to rxmod for recovery
   event:    RX_endMAC
   condition:   RX_FCSOK
          && macSrce != DLL_SM_current.myaddr
          && RX_Lpacket.size == MODERATOR_SIZE
          && RX_Lpacket.ctl == MODERATOR_CTL
          && RX_Lpacket.service == MODERATOR_TAG
          && RX_Lpacket.dest == 0xFF
          && macSrce <= qlowman
          && !(macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
          && !(     moderator.NUT_length == DLL_SM_current.NUT_length
             && moderator.gb_prestart == DLL_SM_current.gb_prestart
             && moderator.gb_start == DLL_SM_current.gb_start
             && moderator.gb_center == DLL_SM_current.gb_center
             && moderator.modulus == DLL_SM_current.modulus
             && moderator.blanking == DLL_SM_current.blanking
             && moderator.slotTime == DLL_SM_current.slotTime
             && moderator.smax == DLL_SM_current.smax
             && moderator.umax == DLL_SM_current.umax)
          && in_guardband == TRUE
   destination: rxMod0
   action:
      qlowman = min(qlowman, macSrce);
      currentMod = macSrce;
      DLL_currentMod_indication(currentMod);
      netState = ROGUE;
      DLL_event(DLL_EV_rogue);

   // The moderator doesn't match, so this node is a rogue.
   // Since currently not in the guardband, go to endframe for recovery.
   event:    RX_endMAC
   condition:   RX_FCSOK
```

```
            && macSrce != DLL_SM_current.myaddr
            && RX_Lpacket.size == MODERATOR_SIZE
            && RX_Lpacket.ctl == MODERATOR_CTL
            && RX_Lpacket.service == MODERATOR_TAG
            && RX_Lpacket.dest == 0xFF
            && macSrce <= qlowman
            && !(macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
            && !(     moderator.NUT_length == DLL_SM_current.NUT_length
               && moderator.gb_prestart == DLL_SM_current.gb_prestart
               && moderator.gb_start == DLL_SM_current.gb_start
               && moderator.gb_center == DLL_SM_current.gb_center
               && moderator.modulus == DLL_SM_current.modulus
               && moderator.blanking == DLL_SM_current.blanking
               && moderator.slotTime == DLL_SM_current.slotTime
               && moderator.smax == DLL_SM_current.smax
               && moderator.umax == DLL_SM_current.umax)
            && in_guardband == FALSE
      destination: endFrame
      action:
         qlowman = min(qlowman, macSrce);
         currentMod = macSrce;
         DLL_currentMod_indication(currentMod);
         netState = ROGUE;
         DLL_event(DLL_EV_rogue);
         gen_timer.start(5,2 usec);
         // start hold timer after line is quiet
         holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// come here to handle a bad moderator DLPDU
// wait for link to become quiet, or until tone generation is due
//
state: badMod

   // This state is reached when this node identifies a moderator DLPDU,
   // but the DLPDU is subsequently reported bad so the state transfers
   // back to endFrame if this node is not in the guardband.
   event:          TRUE
   condition:   in_guardband == FALSE
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // the guardband should be ended NOW
   event:    NUT_timer.count <= 30 usec
   destination: waitTone
   action:
      missed_moderator();
      housekeeping();

   // else wait until the link is quiet
   event:    ph_lock_indication == FALSE
   destination: rxMod0


//
// send the moderator, wait for guardband center
//
state: txMod0

   event:    NUT_timer.count <= DLL_SM_current.gb_center
   destination: txMod1
   action:
      TX_sendHeader(DLL_SM_current.myaddr);      // send header

//
// send more
//
state: txMod1

   event:    TX_headerComplete
```

```
   destination: txMod2
   action:
      moderator.size = MODERATOR_SIZE;
      moderator.ctl = MODERATOR_CTL;
      moderator.service = MODERATOR_TAG;
      moderator.dest = 0xff;
      moderator.NUT_length = DLL_SM_current.NUT_length;
      moderator.smax = DLL_SM_current.smax;
      moderator.umax = DLL_SM_current.umax;
      moderator.slotTime = DLL_SM_current.slotTime;
      moderator.blanking = DLL_SM_current.blanking;
      moderator.gb_start = DLL_SM_current.gb_start;
      moderator.gb_center = DLL_SM_current.gb_center;
      moderator.usr = usr;
      moderator.interval_count = interval_count;
      moderator.modulus = DLL_SM_current.modulus;
      moderator.tMinus = tMinus;
      moderator.gb_prestart = DLL_SM_current.gb_prestart;
      moderator.spare = 0;

      TX_sendLpacket(moderator);       // send the Lpacket

//
// finish up
//
state: txMod2

   event:          TX_LpacketComplete
   destination:    txMod3
   action:
      TX_sendTrailer();

//
// at completion, transferring to waiting for tone
//
state: txMod3

   event:    TX_trailerComplete
   destination: waitTone
   action:
      currentMod = DLL_SM_current.myaddr;
      DLL_currentMod_indication(currentMod);
      housekeeping();


//
// wait for tone
//
state: waitTone

   // end of this NUT, and transferring to off-line
   event:    NUT_timer.count == 0
   condition:   SM_online == FALSE
   destination: offline
   action:
      DLL_tone_indication(interval_count);
      DLL_online_confirm(SM_online);


   // end of this NUT, start of another
   event:    NUT_timer.count == 0
   condition:   SM_online_req == TRUE
   destination: waitSlotZero
   action:
      DLL_tone_indication(interval_count);
      NUT_timer.restart();
      scheduled = TRUE;
      in_guardband = FALSE;

      // housekeeping

      // If this node detects line activity, but not moderators, attempt to recover.
      // Maybe problem is that the local node is performing housekeeping (and is
```

```
      // therefore deaf) during the time the moderator DLPDU is transmitted.

      if (!(netState == MOD || netState == NOTMOD)
         &&  qlowman != 255
         &&  ph_frame_indication == TRUE)
         {
         deafcount++;
         }
      else
         {
         deafcount = 0;
         }

      gen_timer.start(20 usec); // start timer for start of scheduled time

//
// wait for start of scheduled
//
state: waitSlotZero

   // start a new scheduled token pass
   event:    gen_timer.expired()
   destination: gap
   action:
      itr = -1;
      if (DLL_SM_current.myaddr == 0)  // supernode doesn't try to detect lowman
         {
         nqlowman = 0;
         qlowman = 0;
         }
      else                // else initialize the detector
         {
         nqlowman = 255;
         qlowman = 255;
         }
      slot_timer.restart();
      gen_timer.start(0,6 usec);
```

### 9.3 TxLLC

The TxLLC (transmit LLC) shall receive and buffer Lpackets from the upper layers. It shall pick the next Lpacket to be transmitted based on

– the order in which Lpackets were queued;

– attributes of the Lpacket;

– information provided by the Access Control Machine (ACM).

The TxLLC shall present the selection Lpacket to the ACM for transmission.

```
// DLL TX LLC State Machine Description

// This state machine accepts transmit requests from the DLS-user,
// queues and prioritizes them, and submits one Lpacket at a time
// to the ACM based on parameters received from the ACM.

////////////////////////////////////////////////////////////////////////
//
// type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;
typedef void *IDENTIFIER;

typedef enum {   M_0,
      M_1,
      M_ND_plus,
      M_ND_minus } M_SYMBOL;

// These are the three transmit priorities.
```

```
// hard assignments are so that the priority can be
// used as an index into an array of FIFOs
// note that HIGH and LOW are unscheduled

typedef enum {  SCHEDULED=0,
       HIGH=1,
       LOW=2} PRIORITY;

typedef enum {  OK,
       TXABORT,
       FLUSHED } TXSTATUS; // describes the result of a transmit request.

//
// Lpacket class
//
class Lpacket
   {
   public:

   // return the size octet of the current Lpacket
   USINT size;

   // return the ctl octet of the current Lpacket
   USINT ctl;

// Lpacket constants: masks for ctl octet

#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }

   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }

   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }

   // store next octet to the Lpacket
   void put_octet(USINT data);

   // get an octet from the Lpacket
   USINT &operator[](int index);

   Lpacket(void *p)       // constructor
      {
      }

   Lpacket(int size)         // constructor
      {
      }
   };

//
// superclass of Lpacket that defines an Lpacket to be transmitted
//
class txLpacket: public Lpacket
   {
   public:

   IDENTIFIER id;
   BOOL fixed;
```

```
   // constructors

   txLpacket(void *p): Lpacket(p)
      {
      }

   txLpacket(int size): Lpacket(size)  // size is size of PDU buffer
      {            // (Lpacket plus pads, in octets)
      }
   };


////////////////////////////////////////////////////////////////////////////
//
// interface to DLS-user
//
void DLL_xmit_fixed_request(
          IDENTIFIER id,
       USINT      Lpacket[],
          UINT       size,
          PRIORITY   priority,
          USINT      service,
          USINT      destID);

extern void DLL_xmit_fixed_confirm (IDENTIFIER id, TXSTATUS status);

void DLL_xmit_generic_request(
IDENTIFIER id,
          USINT      Lpacket[],
          UINT       size,
          PRIORITY   priority,
          USINT      tag[3]);

extern void DLL_xmit_generic_confirm (IDENTIFIER id, TXSTATUS status);

void DLL_flush-requests-by-QoS_request (PRIORITY priority);

extern void DLL_flush-requests-by-QoS_confirm (PRIORITY priority);

DLL_flush_single_request( PRIORITY priority, IDENTIFIER xmit_id );


////////////////////////////////////////////////////////////////////////////
//
// interface to ACM
//
txLpacket *pickLpacket(BOOL scheduled, int octetsLeft);// pick an Lpacket to send next
void LpacketSent(TXSTATUS status);        // the Lpacket was sent


////////////////////////////////////////////////////////////////////////////
//
// interface to station management
//
void SM_powerup(void);          // input indication: powerup has occurred


////////////////////////////////////////////////////////////////////////////
//
// a class to represent message FIFOs
//
class FIFO
   {
   public:

   void put(txLpacket lp);      // add an Lpacket to the FIFO
   txLpacket &get(void);        // remove an Lpacket from the FIFO
   txLpacket &peek(void);       // look at the first Lpacket in the FIFO
   void flush(void);         // delete all Lpackets in the FIFO
   void flush1(IDENTIFIER id);     // delete one Lpacket in the FIFO, given it's ID
   BOOL has_data();       // true if the FIFO is not empty
   };
```

```
FIFO fifo[3];             // at least three priority levels shall be provided


////////////////////////////////////////////////////////////////////////////
//
// TxLLC implementation
//

//
// powerup initialization
//
void SM_powerup(void)
   {
   fifo[SCHEDULED].flush();
   fifo[HIGH].flush();
   fifo[LOW].flush();
   }

//
// flush a particular FIFO
//
void DLL_flush-requests-by-QoS_request (PRIORITY priority)
   {
   fifo[priority].flush();
   DLL_flush-requests-by-QoS_confirm(priority);
   }

//
// flush a particular Lpacket
//
void DLL_flush_single_request (PRIORITY priority, IDENTIFIER id)
   {
   fifo[priority].flush1(id);
   }

//
// requests from DLS-user to submit Lpackets for transmission
//
void DLL_xmit_fixed_request(
            IDENTIFIER id,
            USINT      Lpacket[],
            UINT       size,
            PRIORITY   priority,
            USINT      service,
            USINT      destID)
   {
   int tmp;

   tmp = size + 4 + (size&1);    // size of DLSdu plus 4 octet fixed Lpacket header
                  // plus optional data pad

   txLpacket &lp(new txLpacket(tmp));  // create a new Lpacket buffer

   // build the PDU (Lpacket)
   lp[0] = tmp/2;             // size, in words
   lp[1] = 0x01 | ((size&1)<<2);    // ctl octet, plus data pad bit, if needed
   lp[2] = service;           // fixed tag service
   lp[3] = destID;            // destination MAC ID
   memcpy(&lp[4], Lpacket, size);      // copy the rest of the data
                  // there may be a pad octet sent after the DLSdu
   lp.id = id;                // store the user's id (for the confirmation)
   lp.fixed = TRUE;           // store type of Lpacket (fixed)
   fifo[priority].put(lp);       // queue the Lpacket
   }


void DLL_xmit_generic_request(
IDENTIFIER id,
          USINT      Lpacket[],
          UINT       size,
          PRIORITY   priority,
          USINT      tag[3])
   {
```

```
   int tmp;
   int pad;

   tmp = size + 6 + (size&1);   // octet size of DLSdu plus 6 octet GEN Lpacket header
                  //          plus data pad
   pad = 0;

   txLpacket &lp(new txLpacket(tmp));  // create a new Lpacket buffer

   // build the PDU (Lpacket)
   lp[0] = tmp/2;            // size, in words
   lp[1] = 0x12 | ((size&1)<<2);    // control octet,
                  // plus maybe a data pad bit,


   lp[2] = 0;             // tag pad octet (affects memory image of Lpacket)
   lp[3] = tag[0];        // generic tag
   lp[4] = tag[1];        // generic tag
   lp[5] = tag[2];        // generic tag
   memcpy(&lp[6], Lpacket, size);      // copy the rest of the data
                  // there may be pad octet after the DLSdu

   lp.id = id;            // store the user's identifier (for the confirm)
   lp.fixed = FALSE;         // store type of Lpacket (generic)

   fifo[priority].put(lp);       // queue the new Lpacket
   }


static txLpacket *picked = 0;       // remember which Lpacket was picked

//
// called by ACM to pick the next Lpacket to be sent out
//
txLpacket *pickLpacket(BOOL scheduled, int octetsLeft)
   {
   int wordsleft = (octetsLeft+1)/2;   // wordsleft is rounded up, accepting that in
some
                  // cases this results in an Lpacket not being sent

   if(scheduled)          // if scheduled, only look at scheduled FIFO
      {
      // if there is anything in the FIFO and it fits in the time left
      if(fifo[SCHEDULED].has_data()
      && fifo[SCHEDULED].peek().size <= wordsleft)
         {
         picked = &fifo[SCHEDULED].get(); // then pick it to be sent
         }
      else picked = 0;          // no Lpacket available
      }
   else               // if unscheduled -- look at all FIFOs in order
      {
      // if there is anything in the FIFO and it fits in the time left
      if(fifo[SCHEDULED].has_data()
         && fifo[SCHEDULED].peek().size <= wordsleft)
         {
         picked = &fifo[SCHEDULED].get(); // then pick it to be sent
         }
      else if(fifo[HIGH].has_data()       // ditto for HIGH
           && fifo[HIGH].peek().size <= wordsleft)
         {
         picked = &fifo[HIGH].get();
         }
      else if(fifo[LOW].has_data()        // ditto for LOW
           && fifo[LOW].peek().size <= wordsleft)
         {
         picked = &fifo[LOW].get();
         }
      else picked = 0;          // no Lpacket available
      }

   return picked;
   }
```

```
//
// confirmation from the ACM that the picked Lpacket was sent (or possibly aborted)
//
void LpacketSent(TXSTATUS status)
   {
   if(picked->fixed)               // if the Lpacket was fixed
      {
      DLL_xmit_fixed_confirm (picked->id, status);  // generate a fixed confirm
      }
   else                       // else
      {
      DLL_xmit_generic_confirm (picked->id, status);// generate a generic confirm
      }

   delete picked;                  // delete temp data
   picked = 0;
   }
```

### 9.4 RxLLC

The RxLLC (receive LLC) shall buffer Lpackets from the Receive Machine (RxM) as they are assembled. When the RxM indicates that a good DLPDU has concluded, all buffered Lpackets shall be presented to the upper layers in the order received. If the RxM indicates a bad DLPDU has concluded, all buffered Lpackets shall be discarded.

```
// DLL RX LLC State Machine Description

// This state machine gets Lpackets from the RxM,
// and DLPDU status from the deserializer.
// It handles quarantining, and passes quarantined
// Lpackets up to the DLS-user

////////////////////////////////////////////////////////////////////////////
//
// generic type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;

//
// Lpacket class
//
class Lpacket
   {
   public:

   // the size octet of the current Lpacket
   USINT size;

   // the ctl octet of the current Lpacket
   USINT ctl;

// Lpacket constants: masks for ctl octet

#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }
```

```
   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }

   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }


   // store next octet to the Lpacket
   void put_octet(USINT data);

   USINT &operator[](int index);

   // init for a new Lpacket
   void init(void);

   Lpacket(void *p) {} // constructor
   };


class rxLpacket: public Lpacket
   {
   public:
   USINT     source;         // the MAC ID of the node that sent this Lpacket
   int    memory_size;    // size of the Lpacket in memory
   BOOL      fixed;       // true if fixed , false if generic

   rxLpacket(void *p): Lpacket(p) {}
   };


////////////////////////////////////////////////////////////////////////////
//
// a class to represent message FIFOs
//
class FIFO
   {
   public:

   void put(rxLpacket lp);      // add an Lpacket to the fifo
   rxLpacket &get(void);        // remove an Lpacket from the fifo
   void flush(void);         // delete all Lpackets in the FIFO
   BOOL has_data();       // true if the fifo is not empty
   };
```

```
////////////////////////////////////////////////////////////////////////////
//
// a class  to represent generic tags
//
class gen_tag
   {
   USINT data[3];
   public:
   // constructor
   gen_tag(USINT first, USINT second, USINT third)
      {
      data[0] = first;
      data[1] = second;
      data[2] = third;
      }

   USINT &operator[](int index)
      {
      return data[index];
      }
   };


////////////////////////////////////////////////////////////////////////////
//
// internal variables
//

FIFO fifo;


////////////////////////////////////////////////////////////////////////////
//
// interface to PhL
//
BOOL ph_lock_indication;  // optimistic DLPDU detect (clock recovery is tracking)
BOOL ph_frame_indication; // pessimistic DLPDU detect (Manchester valid since the SD)

////////////////////////////////////////////////////////////////////////////
//
// interface to Deserializer
//
extern BOOL RX_endMAC;    // the end delimiter has been detected
extern BOOL RX_FCSOK;     // FCS on last DLPDU was OK
extern BOOL RX_abort;     // abort received on last DLPDU

////////////////////////////////////////////////////////////////////////////
//
// interface to RxM
//
BOOL RX_receivedLpacket;  // indication: an Lpacket has been received
rxLpacket RX_Lpacket(0);  // data: the Lpacket most recently received



////////////////////////////////////////////////////////////////////////////
//
// Interface to DLS-user
//
DLL_recv_fixed_indication (
          USINT Lpacket[],
          UINT  size,
          USINT service,
          USINT sourceID);

DLL_recv_generic_indication (
          USINT Lpacket[],
          UINT  size,
          gen_tag tag);
```

```
////////////////////////////////////////////////////////////////////////////
//
// interface to station management
//
BOOL SM_powerup;    // input indication: powerup has occurred

////////////////////////////////////////////////////////////////////////////
//
// RxLLC states
//

// wait for powerup

state: powerup

   event:    SM_powerup
   destination: idle
   action:
      fifo.flush();


// wait for a DLPDU to start
state: idle

   event:    ph_frame_indication == TRUE
   destination: getLpacket


state: getLpacket

   // stuff a new Lpacket into FIFO
   event:    RX_receivedLpacket     // wait for a new Lpacket to be assembled
   destination: getLpacket
   action:
      fifo.put(RX_Lpacket);        // stuff the Lpacket into the FIFO


   // if FCS is good, give all Lpackets to next layer up
   event:    RX_endMAC
   condition:   RX_FCSOK
   destination: idle
   action:
      rxLpacket &lp(0);

      while(fifo.has_data())
         {
         lp = fifo.get();

         if(lp.fixed)
            {
            DLL_recv_fixed_indication(
               &lp[4],      // rip off 4 octets of header to leave DLSdu
               lp.memory_size-4,
               lp[2],    // send service octet
               lp.source);  // send source MAC ID
            }
         else
            {
            DLL_recv_generic_indication(
               &lp[6],      // rip off 6 octets of header to leave DLSdu
               lp.memory_size-6,
               gen_tag(lp[3],lp[4],lp[5])); // send a three octet tag
            }
         }


   // if DLPDU is damaged, flush all the Lpackets in the FIFO
   event:    RX_endMAC
   condition:   !RX_FCSOK
   destination: idle
   action:
      fifo.flush();             // discard all Lpackets
```

### 9.5 Transmit machine (TxM)

The transmit machine (TxM) shall receive commands and data from the Access Control Machine to transmit the components of a DLPDU. It shall transmit a DLPDU by passing data octets and commands to the Serializer.

```
// DLL TxM State Machine Description

// This state machine gets commands and Lpackets from the ACM to
// build and transmit a DLPDU.  It uses the services of the
// Serializer to do this.

////////////////////////////////////////////////////////////////////////////
//
// generic type and constant definitions
//

typedef enum {FALSE=0, TRUE=1} BOOL;

typedef void *IDENTIFIER;

////////////////////////////////////////////////////////////////////////////
//
// Lpacket definition
//
// Lpacket constants: masks for ctl octet
//
#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

//
// Lpacket class
//
class Lpacket
   {
   public:

   USINT     size;
   USINT     ctl;

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }

   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }

   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }

   // get the next octet to be transmitted
   USINT get_next_octet(void);

   // is this Lpacket aborted?
   BOOL abort(void);
   };


////////////////////////////////////////////////////////////////////////////
//
// internal variables
//
int counter;
```

```
////////////////////////////////////////////////////////////////////////////
//
// interface from ACM and TxLLC
//
BOOL TX_sendHeader;          // command: send preamble, SD, source MAC ID
USINT myaddr;             // input: the source MAC ID
BOOL TX_headerComplete;          // reply: header has been sent

BOOL TX_sendLpacket;             // command: send an Lpacket
class Lpacket lp;            // input: the Lpacket to be sent
BOOL TX_LpacketComplete;         // reply: Lpacket has been sent

BOOL TX_sendTrailer;             // command: send FCS, ED
BOOL TX_done;             // reply: DLPDU is complete

BOOL TX_abort;               // status: DLPDU was aborted


////////////////////////////////////////////////////////////////////////////
//
// interface to Serializer
//
void SER_txRequest(BOOL state);     // command: turn transmitter on and off
void SER_sendData(USINT data);         // command: send a Manchester coded data octet
void SER_sendSD(void);           // command: send a start delimiter
void SER_sendED(void);           // command: send an end delimiter
void SER_clearFCS(void);         // command: clear the FCS accumulator
void SER_sendFCS(void);          // command: send the FCS

BOOL SER_operationComplete;         // reply: current operation is completed


////////////////////////////////////////////////////////////////////////////
//
// interface to station management
//
BOOL SM_powerup;          // event indication


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//
// TxM states

//
// wait for powerup
//
state: powerup

   event:    SM_powerup
   destination: idle

//
// wait for a DLPDU to send
//
state: idle

   // when commanded by ACM to start a DLPDU, send the first preamble
   event:    TX_sendHeader
   destination: pre0
   action:
      TX_done = FALSE;
      TX_headerComplete = FALSE;
      TX_abort = FALSE;
      SER_txRequest(TRUE);      // turn on transmitter
      SER_sendData(0xff);    // send first preamble octet

//
// send second preamble octet
//
state: pre0
```

```
   event:    SER_operationComplete
   destination: pre1
   action:
      SER_sendData(0xff);    // send second preamble octet

//
// send start delimiter
//
state: pre1

   event:       SER_operationComplete
   destination: sd
   action:
      SER_sendSD();       // send source MAC ID

//
// send MAC ID
//
state: sd

   event:    SER_operationComplete
   destination: finishHeader
   action:
      SER_clearFCS();
      SER_sendData(myaddr);

//
// wait for header complete
//
state: finishHeader

   event:    SER_operationComplete
   destination: nextLP
   action:
      TX_headerComplete = TRUE;

//
// wait for a command to either send an Lpacket or to terminate the DLPDU
//
state: nextLP

   // start a new Lpacket, grab the size octet
   event:       TX_sendLpacket
   destination: sendCtl
   action:
      counter = lp.wire_size() - 1;// note size zero is not a permitted value
      SER_sendData(lp.get_next_octet());
      TX_LpacketComplete = FALSE;

   // terminate DLPDU: send the FCS
   event:       TX_sendTrailer
   destination: sendFCS
   action:
      SER_sendFCS();

state: sendCtl

   // sent the CTL octet, ignoring tag pad if present
   event:    SER_operationComplete
   destination: sendData
   action:
      counter = counter - 1;
      SER_sendData(lp.get_next_octet());
      if(lp.tag_pad())    // if tag pad
         {
         lp.get_next_octet();   // discard next octet
         }

//
// send the rest of the data in an Lpacket
//
state: sendData
```

```
   // if aborted, send an abort sequence: SD followed immediately by ED
   event:    SER_operationComplete
   condition:   lp.abort()
   destination: abort1
   action:
      SER_sendData(0xff);    // required before the Start Delimiter to avoid
                  // violating run length requirement of phy layer

   // if more data, send the next octet
   event:    SER_operationComplete
   condition:   !lp.abort() && counter > 0
   destination: sendData
   action:
      counter = counter - 1;
      SER_sendData(lp.get_next_octet());


   // when done, wait for the next command
   event:    SER_operationComplete
   condition:   !lp.abort() && counter <= 0
   destination: nextLP
   action:
      if(lp.data_pad())      // if data pad
         {
         lp.get_next_octet();   // discard next octet
         }
      TX_LpacketComplete = TRUE;// signal that Lpacket is done

//
// send the ED
//
state: sendFCS

   event:    SER_operationComplete
   destination: sendED
   action:
      SER_sendED();

//
// wait for ED complete, then shut down and wait for next DLPDU
//
state: sendED

   event:    SER_operationComplete
   destination: idle
   action:
      SER_txRequest(FALSE);     // turn off transmitter
      TX_done = TRUE;     // signal that DLPDU is done

//
// sent an FF before the abort, when it's done send an Start Delimiter
//
state: abort1

   event:    SER_operationComplete
   destination: abort2
   action:
      SER_sendSD();

//
// send second part of the abort (End Delimiter)
//
state: abort2

   event:    SER_operationComplete
   destination: abort3
   action:
      SER_sendED();

//
// wait for abort complete, then shut down and wait for next DLPDU
//
state: abort3
```

```
   event:    SER_operationComplete
   destination: idle
   action:
      SER_txRequest(FALSE);     // turn off transmitter
      TX_abort = TRUE;    // assert error flag
      TX_LpacketComplete = TRUE;// signal that Lpacket is done
      TX_done = TRUE;     // signal that DLPDU is done
```

### 9.6 Receive machine (RxM)

The receive machine (RxM) shall receive framing information and data octets from the Deserializer. It shall decode Lpackets from the received octet stream for presentation to the ACM and the RxLLC. It shall accept generic and fixed Lpackets for which the corresponding tag filter is enabled, and discard Lpackets that are not enabled.

```
// DLL RxM State Machine Description

// This state machine gets octet symbols from the Deserializer and uses
// them to build Lpackets, which are then sent to the RxLLC.


////////////////////////////////////////////////////////////////////////////
//
// generic type and constant definitions
//

typedef enum {FALSE=0, TRUE=1} BOOL;

typedef void *IDENTIFIER;

////////////////////////////////////////////////////////////////////////////
//
// Lpacket definition
//
// Lpacket constants: masks for ctl octet
//
#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

//
// Lpacket class
//
class Lpacket
   {
   public:

   // the size octet of the current Lpacket
   USINT size;

   // the ctl octet of the current Lpacket
   USINT ctl;

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }

   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }

   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }
```

```
   // store next octet to the Lpacket
   void put_octet(USINT data);

   // get an octet from the Lpacket
   USINT operator[](int index);

   // init for a new Lpacket
   void init(void);
   };


class rxLpacket: public Lpacket
   {
   public:
   USINT  source;      // the MAC ID of the node that sent this Lpacket
   int memory_size; // size of the Lpacket in memory
   BOOL   fixed;    // true if fixed , false if generic
   };



////////////////////////////////////////////////////////////////////////////
//
// a class  to represent generic tags
//
class gen_tag
   {
   USINT data[3];

   public:

   // constructor
   gen_tag(USINT first, USINT second, USINT third)
      {
      data[0] = first;
      data[1] = second;
      data[2] = third;
      }

   USINT operator[](int index)
      {
      return data[index];
      }
   };


////////////////////////////////////////////////////////////////////////////
//
// the interface to the tag filter lookup tables -- the implementation is unspecified
//
class gen_screener
   {
   public:
   BOOL enable(gen_tag tag); // set tag, requires tag, TRUE=success/FALSE=failure
   BOOL disable(gen_tag tag);// clear tag, requires tag, TRUE=success/FALSE=failure
   BOOL lookup(gen_tag tag); // lookup tag, requires tag, TRUE=enabled/FALSE=disabled
   void init(void);    // powerup init & clear
   };

class fixed_screener
   {
   public:
   BOOL enable(USINT service);  // set tag, requires tag, TRUE=success/FALSE=failure
   BOOL disable(USINT service); // clear tag, requires tag, TRUE=success/FALSE=failure
   BOOL lookup(USINT service);  // lookup tag, requires tag,
TRUE=enabled/FALSE=disabled
   void init(void);    // powerup init & clear
   };


////////////////////////////////////////////////////////////////////////////
//
// internal variables
```

```
//
// the tag filtering tables
class gen_screener gen;
class fixed_screener fixed;

// MAC ID of source of DLPDU
USINT source;

// an octet counter
int counter;

// temps for generic screener tag octets
USINT gen0;
USINT gen1;


////////////////////////////////////////////////////////////////////////////
//
// interface to PhL
//
extern BOOL ph_lock_indication;  // optimistic DLPDU detect (clock recovery is
tracking)
extern BOOL ph_frame_indication; // pessimistic DLPDU detect (Manchester valid since
the SD)


////////////////////////////////////////////////////////////////////////////
//
// interface to Deserializer
//
extern BOOL RX_dataReady;     // an octet of a DLPDU is available
extern USINT RX_rxData;          // octet most recently received from DLPDU (source
MAC ID)
extern BOOL RX_endMAC;        // the end delimiter has been detected
extern BOOL RX_FCSOK;         // FCS on last DLPDU was OK
extern BOOL RX_abort;         // abort received on last DLPDU


////////////////////////////////////////////////////////////////////////////
//
// interface to station management

class DLL_config_data
   {
   public:
   USINT  myaddr;
   UINT   NUT_length;
   USINT  smax;
   USINT  umax;
   USINT  slotTime;
   USINT  blanking;
   USINT  gb_start;
   USINT  gb_center;
   USINT  interval_count;
   USINT  modulus;
   USINT  gb_prestart;
   };

extern DLL_config_data SM_current;     // the DLL config variables -- need my MAC ID
extern BOOL SM_powerup;          // input indication: powerup has occurred


////////////////////////////////////////////////////////////////////////////
//
// interface to RxM
//


// received data interface

BOOL RX_receivedLpacket;  // an Lpacket has been received
rxLpacket RX_Lpacket;     // the Lpacket most recently received
```

```
// interface to manage tag filters

extern enable_generic_confirm (IDENTIFIER id, BOOL result);
extern disable_generic_confirm (IDENTIFIER id, BOOL result);
extern enable_fixed_confirm (IDENTIFIER id, BOOL result);
extern disable_fixed_confirm (IDENTIFIER id, BOOL result);


void DLL_enable_generic_request (IDENTIFIER id, gen_tag tag)
   {
   enable_generic_confirm (id, gen.enable(tag));
   }

void DLL_disable_generic_request (IDENTIFIER id, gen_tag tag)
   {
   disable_generic_confirm (id, gen.disable(tag));
   }

void DLL_enable_fixed_request (IDENTIFIER id, USINT service)
   {
   enable_fixed_confirm (id, fixed.enable(service));
   }

void DLL_disable_fixed_request (IDENTIFIER id, USINT service)
   {
   disable_fixed_confirm (id, fixed.disable(service));
   }



////////////////////////////////////////////////////////////////////////////
//
//    RxM states
//

//
// wait for powerup
//
state: powerup

   event:    SM_powerup
   destination: idle
   action:
      RX_receivedLpacket = FALSE;

//
// wait for a DLPDU to start
//
state: idle

   event:    ph_frame_indication == TRUE
   destination: getID


state: getID

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // get the source MAC ID and save it for all the Lpackets in the DLPDU
   event:    RX_dataReady == TRUE
   destination: getSize
   action:
      source = RX_rxData;    // save the source MAC ID for subsequent Lpackets

state: getSize

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // get the size octet and init a new Lpacket
```

```
   event:    RX_dataReady == TRUE
   destination: getCtl
   action:
      RX_receivedLpacket = FALSE;      // re-initialize the interface
      RX_Lpacket.init();
      RX_Lpacket.source = source;      // save the source in the new Lpacket
      RX_Lpacket.put_octet(RX_rxData); // store the size octet


state: getCtl

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // get the CTL octet for a fixed Lpacket
   event:    RX_dataReady == TRUE && (RX_rxData & 0xFB) == 0x01  // fixed tag
   destination: getFixed0
   action:
      RX_Lpacket.put_octet(RX_rxData); // store the ctl octet
      counter = RX_Lpacket.wire_size() - 2;  // init the octet counter

   // get the CTL octet for a generic Lpacket
   event:    RX_dataReady == TRUE && (RX_rxData & 0xFB) == 0x12  // generic tag
   destination: getGen0
   action:
      RX_Lpacket.put_octet(RX_rxData); // store the ctl octet
      RX_Lpacket.put_octet(0);     // store the pad octet
      counter = RX_Lpacket.wire_size() - 2;  // init the octet counter


   // get the CTL octet for a generic Lpacket
   event:    RX_dataReady == TRUE
          && (RX_rxData & 0xFB) != 0x01
          && (RX_rxData & 0xFB) != 0x12
   destination: ignoreRest
   action:
      counter = RX_Lpacket.wire_size() - 2;  // init the octet counter


//
// handle service octet of fixed Lpacket
//
state: getFixed0

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // handle case if tag is enabled
   event:    RX_dataReady == TRUE
          && fixed.lookup(RX_rxData)
   destination: getFixed1
   action:
      RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
      counter--;             // count it

   // handle case if tag is disabled
   event:    RX_dataReady == TRUE
          && !fixed.lookup(RX_rxData)
   destination: ignoreRest
   action:
      counter--;             // count it

//
// handle dest octet of fixed Lpacket
//
state: getFixed1

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle
```

```
   // handle case if addressed to me or broadcast
   event:    RX_dataReady == TRUE
          && (RX_rxData == SM_current.myaddr || RX_rxData == 0xFF)
   destination: getRest
   action:
      RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
      counter--;             // count it
      RX_Lpacket.fixed = TRUE;


   // handle case if not addressed to me
   event:    RX_dataReady == TRUE
          && RX_rxData != SM_current.myaddr
          && RX_rxData != 0xFF
   destination: ignoreRest
   action:
      counter--;             // count it

//
// handle first tag octet of generic Lpacket
//
state: getGen0

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // save the next octet
   event:    RX_dataReady == TRUE
   destination: getGen1
   action:
      RX_Lpacket.put_octet(gen0=RX_rxData);  // save and put the next octet into
Lpacket
      counter--;             // count it

//
// handle second tag octet of generic Lpacket
//
state: getGen1

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // save the next octet
   event:    RX_dataReady == TRUE
   destination: getGen2
   action:
      RX_Lpacket.put_octet(gen1=RX_rxData);  // save and put next data octet into
Lpacket
      counter--;             // count it

//
// handle third octet of generic Lpacket
//
state: getGen2

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // handle case if local node is interested in this generic Lpacket
   event:    RX_dataReady == TRUE
          && gen.lookup(gen_tag(gen0, gen1, RX_rxData))
   destination: getRest
   action:
      RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
      counter--;             // count it
      RX_Lpacket.fixed = FALSE;


   // handle case if not interested in this generic Lpacket
   event:    RX_dataReady == TRUE
          && !gen.lookup(gen_tag(gen0,gen1,RX_rxData))
```

```
   destination: ignoreRest
   action:
      counter--;             // count it


state: getRest

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // get data octets except the last
   event:    RX_dataReady == TRUE
   condition:   counter > 1
   destination: getRest
   action:
      RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
      counter--;             // count it

   // get the last data octet; handle data pad, indicate arrived Lpacket; wait for
next Lpacket
   event:    RX_dataReady == TRUE
   condition:   counter == 1
   destination: getSize
   action:
      RX_Lpacket.put_octet(RX_rxData); // get the last octet
      if(RX_Lpacket.data_pad())    // if a data pad is requested,
         {
         RX_Lpacket.put_octet(0);  // insert a zero
         }

      // remove pad octets from data count
      RX_Lpacket.memory_size -= RX_Lpacket.size*2;

      RX_receivedLpacket = TRUE;    // notify that that an Lpacket has arrived


state: ignoreRest

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // get data octets except the last
   event:    RX_dataReady == TRUE
   condition:   counter > 1
   destination: ignoreRest
   action:
      counter--;             // count it

   // handle the last data octet; wait for next Lpacket
   event:    RX_dataReady == TRUE
   condition:   counter == 1
   destination: getSize
```

### 9.7 Serializer

For each command from the TxM, the Serializer shall produce the appropriate M_symbol stream, which shall be output to the medium via the PhL.

```
// DLL octet Serializer

// This state machine gets commands to transmit octet symbols from the TxM.
// It decomposes the octet symbols into M_symbols, and
// uses the services of the PHY layer to send these on the medium.
// This specifies the correct order of transmission of the various octet symbols.

// The FCS algorithm is the same as that used by HDLC

////////////////////////////////////////////////////////////////////////
//
// data types
```

```
//
typedef enum {FALSE, TRUE} BOOL;


////////////////////////////////////////////////////////////////////////
//
// interface to TxM
//

// requests (TxM to SER)

void SER_txRequest(BOOL state);     // command: turn transmitter on and off
void SER_sendData(USINT data);         // command: send a Manchester coded data octet
void SER_sendSD(void);           // command: send a start delimiter
void SER_sendED(void);           // command: send an end delimiter
void SER_clearFCS(void);         // command: clear the FCS accumulator
void SER_sendFCS(void);          // command: send the FCS

// indications (SER to TxM)

extern void SER_operationComplete(void);  // notify the TxM that an operation is
complete


////////////////////////////////////////////////////////////////////////
//
// interface to the PHY layer
//
typedef enum {M_0, M_1, M_ND_plus, M_ND_minus} M_SYMBOL;

BOOL ph_frame_request;        // transmit enable/disable
M_SYMBOL ph_data_request;        // data to be sent



////////////////////////////////////////////////////////////////////////
//
// internal variables
//
static unsigned int accum;       // the FCS accumulator, 16 bits, sign is irrelevant
                    // as long as zeros shift in from the right

////////////////////////////////////////////////////////////////////////
//
// internal subroutines
//

// add one octet to the FCS accumulator
// this is just one possible implementation of the HDLC FCS

static void FCS_octet(unsigned int d)
   {
   accum ^= d & 0xff;

   for(int i=0;i<8;i++)
      {
      accum = (accum>>1) ^ (0x8408 * (accum&1));// use FCS-CCITT polynomial
      }
   }


////////////////////////////////////////////////////////////////////////
//
// Serializer implementation
//

// requests (TxM to SER)

// enable transmitter
void SER_txRequest(BOOL state)
   {
   ph_frame_request = state;
   }
```

```
// add a data octet to the FCS and send it
void SER_sendData(USINT data)
   {
   FCS_octet(data);

   // transmit data bits 0 through 7

ph_data_request = (data & 0x01) ? M_1: M_0;
ph_data_request = (data & 0x02) ? M_1: M_0;
ph_data_request = (data & 0x04) ? M_1: M_0;
ph_data_request = (data & 0x08) ? M_1: M_0;
ph_data_request = (data & 0x10) ? M_1: M_0;
ph_data_request = (data & 0x20) ? M_1: M_0;
ph_data_request = (data & 0x40) ? M_1: M_0;
ph_data_request = (data & 0x80) ? M_1: M_0;

   SER_operationComplete();     // tell TxM that operation is complete
   }


// send a start delimiter
void SER_sendSD(void)
   {
   ph_data_request = M_ND_plus;
   ph_data_request = M_0;
   ph_data_request = M_ND_minus;
   ph_data_request = M_ND_plus;
   ph_data_request = M_ND_minus;
   ph_data_request = M_1;
   ph_data_request = M_0;
   ph_data_request = M_1;
   SER_operationComplete();     // tell TxM that operation is complete
   }


// send an end delimiter
void SER_sendED(void)
   {
   ph_data_request = M_1;
   ph_data_request = M_0;
   ph_data_request = M_0;
   ph_data_request = M_1;
   ph_data_request = M_ND_plus;
   ph_data_request = M_ND_minus;
   ph_data_request = M_ND_plus;
   ph_data_request = M_ND_minus;
   SER_operationComplete();     // tell TxM that operation is complete
   }


// initialize the FCS accumulator
void SER_clearFCS(void)
   {
   accum = 0xffff;        // init the accumulator to all ones
   SER_operationComplete();     // tell TxM that operation is complete
   }


// send the FCS
void SER_sendFCS(void)
   {
   int tmp;

   tmp = accum ^ 0xffff;        // invert the FCS before sending;
   SER_sendData(tmp);        // do not calculate FCS while sending itself
   SER_sendData(tmp>>8);
   SER_operationComplete();     // tell TxM that operation is complete
   }
```

## 9.8 Deserializer

### 9.8.1 Octet construction

The Deserializer shall accept M_symbols from ph_data_indication. M_symbols at the start of a frame shall be ignored until the ph_frame_indication line transitions from false to true. This transition shall start on the first data octet – the source MAC ID. The Deserializer shall then group subsequent sets of eight MAC data symbols into octets. As each subsequent data octet is ready, the Deserializer shall transition the RX_dataReady line from false to true.

### 9.8.2 FCS checking

A modified CCITT Cyclic Redundancy Check shall be performed on the transferred data to verify the frame check sequence (FCS).

The formal concepts for the FCS generator and checker are described in 5.3.8.2.

The method used for checking shall be the same as specified for HDLC (ISO/IEC 3309), with the exception that start and end delimiters shall be substituted for flags, and Manchester encoding shall be substituted for bit-stuffing. The FCS checker shall implement the polynomial $X^{16} + X^{12} + X^5 + 1$, and shall result in a two octets frame check sequence (FCS).

NOTE The last two data octets in the DLPDU (FCS octets) are calculated by the transmitting node so that, in the absence of errors, the remainder in the receiving node is always 0xF0B8 (see 5.3.8.2).

The receive FCS process shall begin by pre-setting the FCS checker to 0xFFFF when the ph_frame_indication line transitions from false to true. All subsequent data bits on the ph_data_indication line, excluding the end delimiter, shall be applied to the FCS checker. At end of receiving a DLPDU, the remainder (FCS) shall be compared to 0xF0B8.

### 9.8.3 End of DLPDU processing

Data octet processing shall proceed until ph_frame_indication transitions from true to false. This transition can happen mid-octet, but no action shall be taken until the start of the next octet. This time shall mark the end of reception for this frame.

The ph_status_indication received after ph_frame_indication transitions from true to false shall determine which of the following actions to take:

– if Normal – set RX_FCSOK to true if (FCS remainder == 0xF0B8); otherwise, set to false;
– if Abort – set Rx_abort to true; set Rx_FCSOK to false;
– if Invalid – set Rx_abort to false; set Rx_FCSOK to false.

## 9.9 DLL management

DLL Management shall buffer the station management variables that are required for the DLL to operate. It shall manage synchronized changes on these variables via interfaces to Station Management and the ACM.

```
// DLL Station Management Interface Description

// This component holds the station management variables that are
// subject to synchronized change.

////////////////////////////////////////////////////////////////////////
//
// type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;

typedef void *IDENTIFIER;     // The data type of the identifier is unspecified.
```

```
// config data definition

class DLL_config_data
   {
   public:
   USINT  myaddr;
   UINT   NUT_length;
   USINT  smax;
   USINT  umax;
   USINT  slotTime;
   USINT  blanking;
   USINT  gb_start;
   USINT  gb_center;
   USINT  interval_count;
   USINT  modulus;
   USINT  gb_prestart;
   };


////////////////////////////////////////////////////////////////////////////
//
// interface to station management
//
void DLL-tMinus-start-countdown-request (IDENTIFIER id, USINT start_count);
extern void DLL-tMinus-start-countdown-confirm (IDENTIFIER id, BOOL result);

void DLL_set_pending_request (IDENTIFIER id, DLL_config_data params);
extern void DLL_set_pending_confirm (IDENTIFIER id, BOOL result);

void DLL_get_pending_request (IDENTIFIER id);
extern void DLL_get_pending_confirm (IDENTIFIER id , DLL_config_data params);

void DLL_set_current_request (IDENTIFIER id, DLL_config_data params);
extern void DLL_set_current_confirm (IDENTIFIER id, BOOL result);

void DLL_get_current_request (IDENTIFIER id);
extern void DLL_get_current_confirm (IDENTIFIER id , DLL_config_data params);


////////////////////////////////////////////////////////////////////////////
//
// interface to ACM
//
extern void DLL_tminus_zero_indication ();// ACM calls this when tMinus==0 occurs
DLL_config_data DLL_SM_pending;     // the ACM needs to see both current and pending
DLL_config_data DLL_SM_current;
extern ACM_tMinus;            // poke the ACM's tMinus counter to get
                    // a synchronized change started.  The ACM
                    // continually writes over this every time the
                    // moderator is received, unless it is the moderator

void SM_supernode(void);         // if this node saw a moderator sent by a supernode,
                    // any pending SM update shall be cancelled


////////////////////////////////////////////////////////////////////////////
//
// internal variables
//
BOOL pending_changed = FALSE;


////////////////////////////////////////////////////////////////////////////
//
// write the pending copy of DLL_config
void DLL_set_pending_request (IDENTIFIER id, DLL_config_data params)
   {
   DLL_SM_pending = params;      // save the params in the pending buffer
   pending_changed = TRUE;
   DLL_set_pending_confirm (id, TRUE);
   }
```

```
// write the current copy of DLL_config
// ONLY USED WHEN OFFLINE
void DLL_set_current_request (IDENTIFIER id, DLL_config_data params)
   {
   DLL_SM_current = params;      // save the params in the pending buffer
   DLL_set_current_confirm (id, TRUE);
   }


// read the pending copy of DLL_config
void DLL_get_pending_request (IDENTIFIER id)
   {
   if(pending_changed == TRUE)      // if pending has been updated
      {
      // return the contents of the pending buffer
      DLL_get_pending_confirm (id , DLL_SM_pending);
      }
   else                // if pending buffer is invalid
      {
      // return the contents of the current buffer
      DLL_get_current_confirm (id , DLL_SM_current);
      }
   }

//
// read the current copy of DLL_config
//
void DLL_get_current_request (IDENTIFIER id)
   {
// return the contents of the current buffer
   DLL_get_current_confirm (id , DLL_SM_current);
   }

//
// called by the ACM when a tMinus countdown has completed
//
void DLL_tminus_zero_indication(void)
   {
   if(pending_changed)
      {
      DLL_SM_current = DLL_SM_pending;
      pending_changed = FALSE;
      }
   }

//
// start a synchronized change sequence
//
void DLL-tMinus-start-countdown-request (IDENTIFIER id, USINT start_count)
   {
   ACM_tMinus = start_count;     // the ACM polls this variable
   DLL-tMinus-start-countdown-confirm (id, TRUE);
   }

//
// cancel pending change if a supernode has been seen
//
void SM_supernode(void)
   {
   pending_changed = FALSE;
   }
```

# 10 Device Level Ring (DLR) protocol

## 10.1 General

Clause 10 defines the Device Level Ring (DLR) protocol, a Layer 2 protocol that provides media redundancy in a ring topology. The DLR protocol is intended primarily for

implementation in CP 2/2 end devices that have multiple Ethernet ports and embedded switch technology. The DLR protocol provides for fast network fault detection and reconfiguration in order to support the most demanding control applications.

Since the DLR protocol operates at Layer 2 (in the OSI network model), the presence of the ring topology and the operation of the DLR protocol are transparent to higher layer protocols such as TCP/IP and CP 2/2, with the exception of a DLR object that provides a configuration and diagnostic interface for CP 2/2.

A DLR network includes at least one node configured to be a ring supervisor, and any number of normal ring nodes. It is assumed that all the ring nodes have at least two Ethernet ports and incorporate embedded switch technology.

Non-DLR multi-port devices – switches or end devices – may be placed in the ring, subject to certain implementation constraints (e.g., no MAC table filtering). Non-DLR devices will also impact the worst-case ring recovery time.

The DLR protocol definition includes a number of aspects:

- a set of end node behaviors, for ring supervisors and normal ring nodes;
- protocol messages and associated state diagrams;
- implementation requirements for devices.

### 10.2 Supported topologies

The DLR protocol supports a simple, single-ring topology; it has no concept of multiple or overlapping rings. A network installation may however use more than one DLR-based ring, so long as each of the rings are isolated such that DLR protocol messages from one ring are not present on another ring.

The DLR protocol may coexist with, but does not interface with, standard network protocols such as IEEE Spanning Tree Protocols (STP, RSTP, MSTP), and also with vendor-specific redundancy protocols. That is, users may construct network topologies with DLR protocol rings connected to switches that are running Spanning Tree or other ring protocols, as shown in Figure 27.

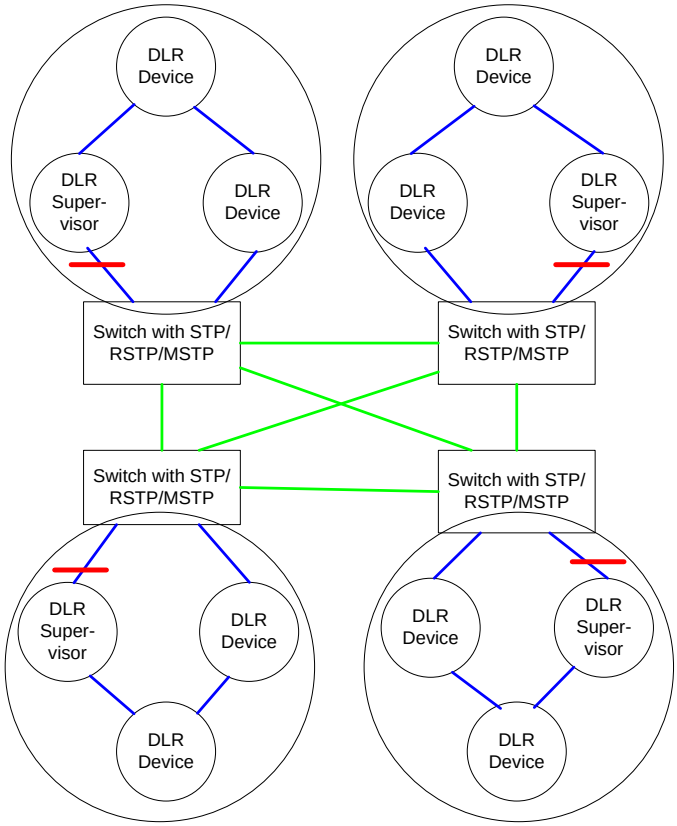



**Figure 27 – DLR rings connected to switches**

In Figure 27, each DLR ring is a separate DLR network, each with a ring supervisor. The supervisors are shown with one port in blocked mode, which is the case when there are no faults in the ring.

The switches to which the DLR rings are connected may run STP/RSTP/MSTP to prevent loop free operation when redundant paths are present (indicated by the green lines in Figure 27). Spanning Tree Protocol messages (BPDUs) that are sent by the switch on the DLR ring ports will be blocked by the DLR Ring Supervisor, so that the switches do not block the DLR ports (see IEEE 802.1D/IEEE 802.1Q STP/RSTP/MSTP considerations in 10.6.4).

The switches' ports to which the DLR devices are connected shall be configured properly in order ensure proper functioning of the network (see 7.9).

More complicated topologies combining DLR rings and non-DLR switches running STP/RSTP/MSTP may result in DLR ports being blocked in an undesirable manner. See 7.9 for additional information.

DLR supports redundant gateways for connecting with network infrastructure outside of the DLR network to the DLR network itself. See 10.8 for additional information.

## 10.3 Overview of DLR operation

### 10.3.1 Normal operation

Figure 28 shows the normal operation of a DLR network.

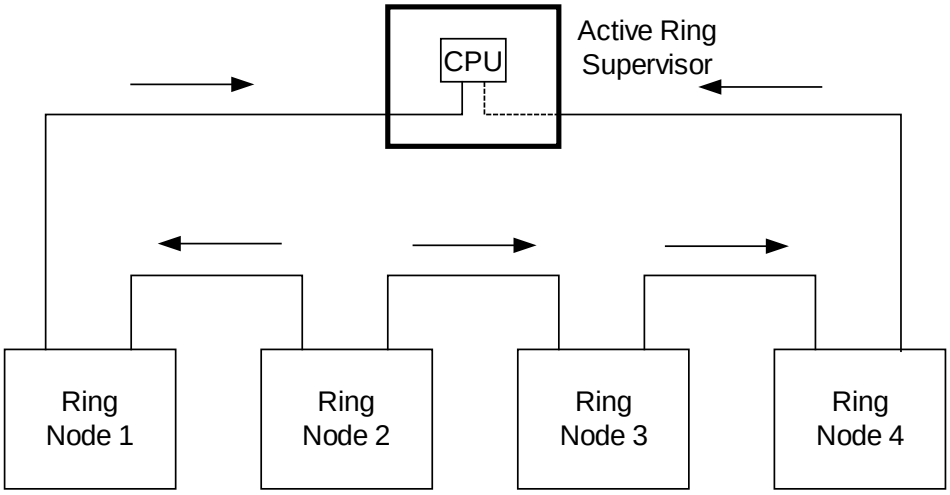



**Figure 28 – Normal operation of a DLR network**

and is assumed to have implemented an embedded switch. When a ring node receives a packet on one of its Ethernet ports, it determines whether the packet needs to be received by the ring node itself (e.g., the packet has the node's MAC address) or whether the packet shall be sent out the node's other Ethernet port.

The active ring supervisor blocks traffic on one of its ports with the exception of few special frames and does not forward traffic from one port to other. Because of this configuration a network loop is avoided and only one path exists between any two ring nodes during normal operation.

Figure 29 illustrates the use of Beacon and Announce frames sent by the active ring supervisor.

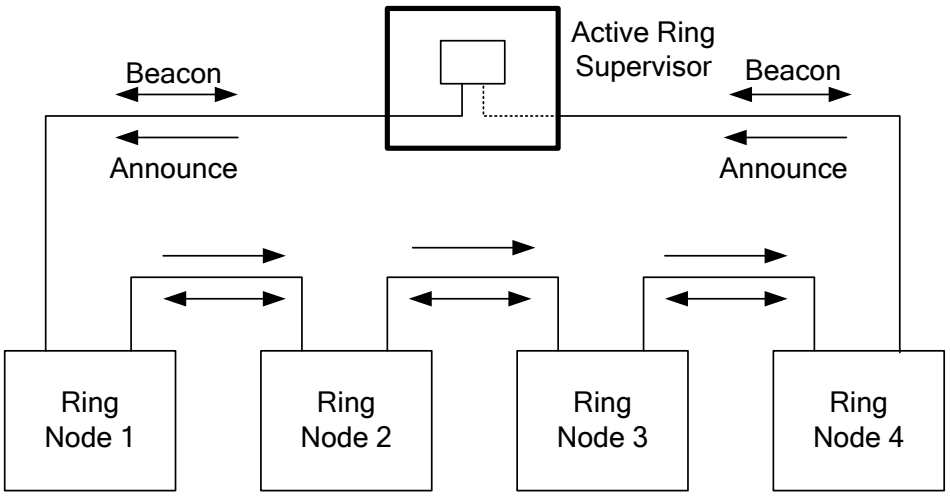



**Figure 29 – Beacon and Announce frames**

The active ring supervisor transmits a Beacon frame through both of its Ethernet ports once per beacon interval (400 µs by default). The supervisor also sends Announce frames once per second. The Beacon and Announce frames serve several purposes:

a) the presence of Beacon and Announce frames inform ring nodes to transition from linear topology mode to ring topology mode;

b) loss of Beacon frames at the supervisor enables detection of certain types of ring faults. (normal ring nodes are also able to detect and signal ring faults);

c) the Beacon frames carry a precedence value, allowing selection of an active supervisor when multiple ring supervisors are configured.

### 10.3.2 Link failures

#### 10.3.2.1 Common failures

The most common form of link failure includes the following cases:

a) link or other physical layer failure recognized by a node adjacent to the failure;

b) power failure or power cycling a ring node, recognized by the adjacent node as a link failure;

c) intentional media disconnect by user to bring new nodes online or to remove existing ones.

In the above cases, the nodes adjacent to the fault send a Link_Status message to the ring supervisor. Figure 30 shows ring nodes adjacent to a fault sending a Link_Status message to the ring supervisor.

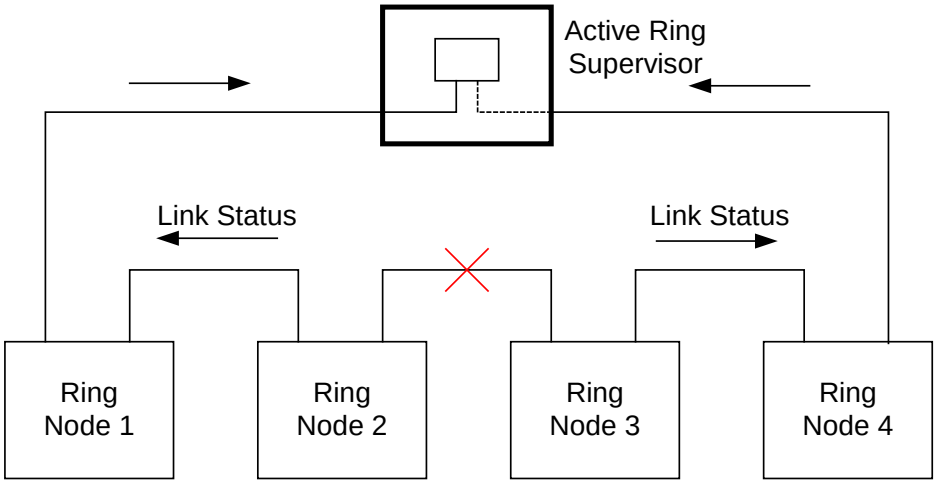



**Figure 30 – Link failure**

After receipt of the Link_Status message, the ring supervisor reconfigures the network by unblocking traffic on its previously blocked port and flushing its unicast MAC table. The supervisor immediately sends Beacon and Announce frames with the ring state value indicating that the ring is now faulted.

Ring nodes also flush their unicast MAC tables upon detecting loss of the beacon in one direction, or upon receipt of Beacon or Announce frames with the ring state value indicating the ring fault state. Flushing the unicast MAC tables at both supervisor and ring nodes is necessary for network traffic to reach its intended destination after the network reconfiguration.

Figure 31 shows the network configuration after a link failure, with the ring supervisor passing traffic through both of its ports.

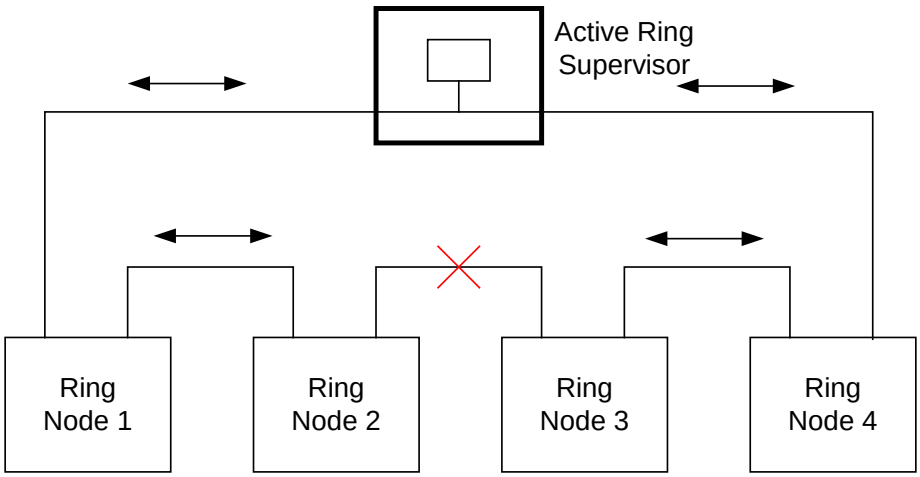



**Figure 31 – Network reconfiguration after link failure**

#### 10.3.2.2 Uncommon failures

In addition to the more common link failures, there is a class of uncommon failures:

a) higher level hardware/firmware component(s) on a ring node has failed leading to lost traffic, but the physical layer is functioning normally with power supply intact;

b) a chain of ring protocol unaware nodes are connected between protocol-aware nodes, and the failure has occurred somewhere in the middle of this chain.

In these cases, the ring supervisor will detect the loss of Beacon frames first on one port, and eventually on both of its ports. The supervisor will reconfigure the network as described in the 10.3.2.1. In addition, the ring supervisor will send a Locate_Fault frame to diagnose the fault location (see 10.5.3.7 on the Neighbor Check process).

#### 10.3.2.3 Partial network fault

It is possible for a partial network fault to occur such that traffic is lost in only one direction. The active ring supervisor detects a partial fault by monitoring the loss of Beacon frames on one port. When a partial fault is detected the active supervisor blocks traffic on one port and sets a status value in the DLR Object. The ring at this point will be segmented due to the partial fault, requiring user intervention.

#### 10.3.2.4 Rapid fault/restore cycles

Certain conditions such as a faulty network connector may cause the ring supervisor to detect a series of rapid fault/restore cycles. If left to persist, such a condition could result in network instability that is difficult to diagnose. When the active supervisor detects the rapid fault/restore condition (5 faults in a 30 s period), it sets a status value in the DLR object, and blocks traffic on one port. The user shall explicitly clear the condition via the DLR object.

### 10.4 Classes of DLR implementation

There are several classes of DLR implementation, as described below. Detailed requirements for each class of implementation are further specified in 10.5.

- Ring Supervisor
  This class of device is capable of being a ring supervisor. Such devices shall implement the required ring supervisor behaviors, including the ability to send and process Beacon frames at the default beacon interval of 400 µs. Smaller beacon intervals, as low as 100 µs, may be supported, but are not required.

- Ring Node, Beacon-based
  This class of device implements the DLR protocol, but without the ring supervisor capability. The device shall be able to process and act on the Beacon frames sent by the ring supervisor. Beacon-based ring nodes shall support beacon rates of 100 µs to 100 ms in order to accommodate all ring supervisor implementations.
- Ring Node, Announce-based
  This class of device implements the DLR protocol, but without the ring supervisor capability. In order to accommodate nodes that do not have the capacity to process Beacon frames, ring nodes may simply forward, but need not explicitly process, Beacon frames. Such nodes shall process Announce frames.

## 10.5 DLR behavior

### 10.5.1 DLR variables

Table 134 summarizes variables used in the DLR protocol behavior and messages. See 10.5.3 and 10.5.2 on Ring Node and Ring Supervisor behavior and DLR messages for further details. The DLR object (see 7.9) exposes these variables (with the exception of the Node State) via object attributes.

**Table 134 – DLR variables**

| DLR variable | Description |
|---|---|
| Node State | Internal state of a node's DLR state machine:<br>IDLE_STATE – initial state for non-supervisors, indicating linear topology mode<br>FAULT_STATE – initial state for enabled ring supervisor, or when ring fault has been detected (both supervisor and ring nodes)<br>NORMAL_STATE – normal function in ring topology mode |
| Ring State | State of the ring network, transmitted by ring supervisors in Beacon and Announce frames:<br>RING_NORMAL_STATE – Ring is functioning, with supervisor blocking traffic on one port<br>RING_FAULT_STATE – Fault detected, ring supervisor is not blocking traffic (also is the initial state transmitted in the Beacon and Announce frames) |
| Beacon Interval | Interval at which the ring supervisor sends Beacon frames. Supervisors shall support a range from 400 µs to 100 ms. The default value shall be 400 µs. Supervisors may support a Beacon Interval smaller than 400 µs, but this is not required. The absolute minimum Beacon Interval is 100 µs. Beacon-based ring nodes shall support beacon rates of 100 µs to 100 ms in order to accommodate all ring supervisor implementations |
| Beacon Timeout | Amount of time nodes shall wait before timing out reception of Beacon frames and taking the appropriate action (depending on whether supervisor or normal ring node). Supervisors shall support a range from 800 µs to 500 ms. The default shall be 1960 µs. Supervisors may support a Beacon Timeout of smaller than 800 µs but this is not required. The absolute minimum Beacon Timeout is 200 µs |
| Supervisor Precedence | Precedence value assigned to a ring supervisor, and transmitted in Beacon frames. Used to select active ring supervisor when multiple supervisors have been configured.<br>Default value is 0. Can be changed via the DLR object |
| DLR VLAN ID | VLAN ID used when sending DLR protocol frames. The VLAN ID is configured at the ring supervisor (via the DLR object), and is then detected by ring nodes when they receive and process the Beacon or Announce frames from the supervisor.<br>Default value is 0. Typically the VLAN ID does not need to be changed unless a commercial switch is being used in the ring |

### 10.5.2 Ring supervisor

#### 10.5.2.1 Startup

An enabled ring supervisor shall start in FAULT_STATE and configure both ports to forward frames. The supervisor shall send Beacon frames out both of its ports, with the Ring State set

to RING_FAULT_STATE. The supervisor shall also send Announce frames out both of its ports with the Ring State set to RING_FAULT_STATE.

Once the Beacon frames are received through both ports the supervisor shall transition to NORMAL_STATE, flush its unicast MAC address table and reconfigure one of its ports not to forward packets, except for the following, which shall be forwarded to the host for processing:

- Beacon frames with the supervisor's own MAC address (in general needed only for software implementations);
- Beacon frames from other ring supervisors;
- Link_Status/Neighbor_Status frames;
- Neighbor_Check request or response, and Sign_On: always forward received frames. For frames originated by the supervisor, only forward frames with the Source Port matching the blocked port.

Upon transition to NORMAL_STATE, the Ring State in the Beacon frames shall be set to RING_NORMAL_STATE. The ring supervisor shall also send an Announce frame out one port, with Ring State set to RING_NORMAL_STATE.

#### 10.5.2.2 Multiple ring supervisors

When multiple ring supervisors are configured, each supervisor sends Beacon frames when it comes online. The Beacon frames carry a supervisor precedence value. When a supervisor receives a Beacon frame, it checks the precedence value. If the precedence in the Beacon frame is higher than the receiving node's precedence value, the receiving node transitions to FAULT_STATE and becomes a backup supervisor. If the precedence values are the same, the node with the numerically higher MAC address becomes the active supervisor.

The backup supervisors configure their DLR parameters with the values obtained from the active supervisor's Beacon frames: Beacon Interval, Beacon Timeout, VLAN ID.

The backup supervisors continue to monitor both ports for timeout of the Beacon frames (no Beacons received within the Beacon Timeout period). If the Beacon has timed out on both ports, the backup supervisor waits for an additional Beacon Timeout period (during which time other nodes transition to linear mode), then begins sending its own Beacons so that a new supervisor can be selected.

#### 10.5.2.3 Sign on

In order to identify ring protocol participants, the active ring supervisor shall send a Sign_On frame when it transitions to NORMAL_STATE. See the Sign_On information in the DLR Messages (see 10.9.9).

#### 10.5.2.4 Normal ring operation

When in the NORMAL_STATE, the active ring supervisor shall send Beacon frames out both of its ports. It shall also send an Announce frame once per second out one port.

One of the active supervisor's ports shall be configured not to forward frames, with the exceptions as noted in 10.3.2.1.

#### 10.5.2.5 Ring fault detection

One of several possible events shall cause the active ring supervisor to transition to FAULT_STATE:

a) Beacon frame received from another supervisor with a higher precedence value;

b) loss of Beacon frames on either port for the period specified by the Beacon Timeout, indicating a break somewhere in the ring;

c) detection of loss of link with the neighboring node on either port;

d) Link_Status frame received from a ring node, indicating a ring node has detected a fault.

In all of the cases listed above, the active ring supervisor shall:

- transition to FAULT_STATE;
- flush its unicast MAC address table;
- unblock the blocked port;
- send Beacon frame out both ports, with Ring State set to RING_FAULT_STATE;
- send Announce frame out both ports, with Ring State set to RING_FAULT_STATE.

In addition, in case 2 above, the active ring supervisor shall initiate the Neighbor Check process by issuing a Locate Fault frame. The supervisor shall also issue its own Neighbor Check frame through the port(s) on which the beacon has timed out.

When in FAULT_STATE the ring supervisor shall continue to send Beacon frames, in order to detect ring restoration.

#### 10.5.2.6 Ring restoration

When the active ring supervisor is in FAULT_STATE, receipt of Beacon frames on both ports shall cause a transition to NORMAL_STATE. The active ring supervisor shall do the following:

- flush the unicast MAC address table;
- reconfigure its ports such that traffic is not forwarded on one port (with exceptions as noted previously);
- send Beacon frames with the Ring State set to RING_NORMAL_STATE;
- send Announce frames out one port with Ring State set to RING_NORMAL_STATE.

#### 10.5.2.7 Changing ring parameters

The following ring supervisor parameters may be changed via the DLR object (see 7.9):

- Supervisor precedence value;
- Beacon interval;
- Beacon timeout;
- VLAN ID;
- Supervisor enabled/disabled.

When any of the above parameters are changed on the active ring supervisor, the ring supervisor shall cease sending Beacon and Announce frames for two (current) beacon timeout periods, then shall send Beacon frames using the new parameters. Ceasing the Beacon frames allows beacon-based ring nodes to detect the new parameters when the Beacon is restored and triggers active supervisor negotiation based on the new parameters. Announce frame production is further suppressed for at least 2 new beacon timeout periods to make sure that the active supervisor is determined before any Announce frames with the new parameters are sent.

When parameters are changed on a backup ring supervisor, the behavior depends on the backup supervisor's new precedence value compared to the active supervisor's precedence value:

- new backup precedence value is greater than the current active ring supervisor's precedence or of equal precedence with numerically higher MAC address than active supervisor MAC address: backup shall immediately begin sending Beacon frames with the new parameter;
- new backup precedence value is less than the active supervisor's precedence or of equal precedence with numerically lower MAC address than active supervisor MAC address: modification to the Beacon Interval, Beacon Timeout, and VLAN ID shall be ignored.

### 10.5.3 Ring node

#### 10.5.3.1 Beacon versus Announce-based implementations

Ring nodes (that is, non-supervisor nodes) may have differing implementations depending on whether or not they are able to process the Beacon frames which by default are sent every 400 µs (but may be sent as fast as every 100 µs). Nodes that are able to process the Beacon frames generally have hardware assistance in implementing the DLR protocol, so that they do not burden the device's CPU with processing the Beacon frames.

Devices that would need to process the Beacon frames in the device's CPU can instead configure their embedded switch to simply pass the Beacon frames on the network without interpretation or further processing. Such devices shall however process the Announce frames, which also indicate the ring state but are sent at a much slower rate.

NOTE It is possible to implement a Beacon-based node without hardware assistance, provided the device's CPU has sufficient capacity to process the Beacon frames in addition to its other required functions.

It is desirable for device implementations to be Beacon-based rather than Announce-based, since better ring recovery performance results when ring nodes are able to process Beacon frames. See performance analysis in 10.10.4.

#### 10.5.3.2 Startup – Beacon-based

A Beacon-based ring node shall start up in IDLE_STATE, which presumes the network is in linear topology mode.

Upon receiving a Beacon frame through either port, the node shall transition to FAULT_STATE, which presumes the ring topology mode. The ring node shall flush its unicast MAC address table and save the ring supervisor parameters from the Beacon frame:

- Supervisor MAC address;
- Supervisor precedence value;
- Beacon timeout;
- VLAN ID.

Upon receiving Beacon frames through both ports and after receiving a Beacon frame from active ring supervisor with ring state field set to RING_STATE_NORMAL on either one of its ports, the node shall transition to NORMAL_STATE and flush its unicast MAC address table.

#### 10.5.3.3 Startup – Announce-based

An Announce-based ring node shall start up in IDLE_STATE, which presumes the network is in linear topology mode.

Upon receiving an Announce frame through either port, the node shall transition to the ring state indicated in the Announce frame. The ring node shall flush its unicast MAC address table and save the ring supervisor parameters from the Announce frame:

- Supervisor MAC address;

- Ring State;
- VLAN ID.

#### 10.5.3.4 Fault detection

One of several possible events shall cause a ring node to transition from NORMAL_STATE:

For Beacon-based nodes:

a) receipt of a Beacon frame with the Ring State set to RING_FAULT_STATE;
b) receipt of a Beacon frame with a different MAC address and higher precedence than the current ring supervisor;
c) loss of Beacon frames on both ports for the period specified by the Beacon Timeout, which causes the node to transition to IDLE_STATE (i.e., the topology is now linear);
d) loss of Beacon on a single port for the period specified by the Beacon Timeout.

For Announce-based nodes:

a) receipt of an Announce frame with the Ring State set to RING_FAULT_STATE;
b) loss of Announce frame for the Announce timeout duration, which causes the node to transition to IDLE_STATE (i.e., the topology is now linear).

In all of the cases listed above, the ring node shall:

- flush its unicast MAC address table;
- transition to FAULT_STATE (exception: loss of Beacon on both ports or loss of Announce causes transition to IDLE_STATE).

#### 10.5.3.5 Ring restoration

For ring nodes, the process for ring restoration is the same as the Startup case (see 10.5.3.2 and 10.5.3.3).

#### 10.5.3.6 Sign on process

The Sign_On frame is used to identify all ring participants. The active ring supervisor shall send a Sign_On frame when it transitions to NORMAL_STATE. The active supervisor transmits a Sign_On frame once every one minute while in NORMAL_STATE, until it receives a Sign_On that it sent out previously. Upon receiving such a frame the active supervisor will cease to send further Sign_On frames until next transition into NORMAL_STATE. The collected participant list can be accessed through the DLR object.

The Sign_On frame is a multicast message transmitted from one port of the active ring supervisor. The receiving ring participant node traps the Sign_On frame and forwards it only to the host CPU. The host CPU increments the number of nodes in list, add its own addresses to list and transmit the Sign_On frame only through the other port than the receiving port.

The Sign_On frame is transmitted from one ring participant node to the next in similar fashion and eventually reaches the active supervisor. The active supervisor can identify the Sign_On frame it sent out by confirming that the first entry is its own.

It is possible that the number of nodes in the ring is large enough that all nodes' addresses do not fit in the Sign_On frame. When a node receives the Sign_On frame, if adding the node's address would exceed the maximum frame size, the node does not add its address, but saves the port through which the frame was received and sends the Sign_On frame directly to the active ring supervisor.

When the ring supervisor receives the Sign_On frame sent to its unicast MAC address, it assumes this is due to the Sign_On frame size reaching its maximum. The supervisor restarts the Sign On process by sending a new Sign_On frame directly (unicast) to the node from which it received the unicast Sign_On frame.

Upon receiving the new Sign_On frame from the ring supervisor, the ring node adds its address to the Sign_On frame. The node then sends the Sign_On frame (multicast) through its other port than the saved port.

#### 10.5.3.7 Neighbor check process

When the active ring supervisor detects the loss of Beacon, it sends a Locate_Fault frame through both ports.

Upon receipt of the Locate_Fault frame, each ring node issues a Neighbor_Check request through both of its ports. The supervisor also issues its own Neighbor_Check request.

When any node receives a Neighbor_Check request frame it responds with a Neighbor_Check response frame through the port on which original request was received. If the node sending the Neighbor_Check request does not receive a response in 100 ms, it shall retry the request. After a total of 3 attempts, if no response is received after the overall 300 ms timeout expires, the node sends a Neighbor_Status frame to the ring supervisor.

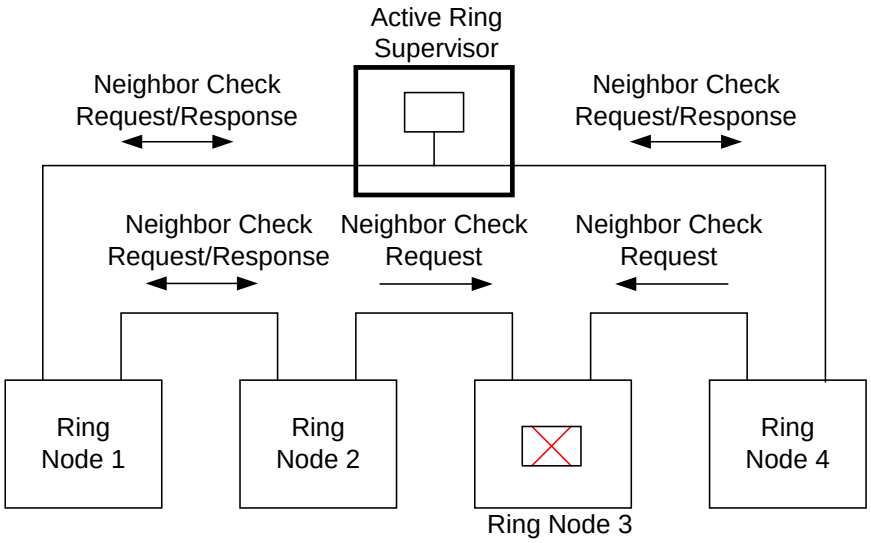



**Figure 32 – Neighbor Check process**

Figure 32 illustrates the Neighbor Check process. In this example, all healthy ring nodes respond, while the failed ring node 3 does not. Ring nodes 2 and 4 will each ultimately send a Neighbor_Status frame to the ring supervisor.

### 10.6 Implementation requirements

#### 10.6.1 Embedded switch requirements and recommendations

The following are general requirements and recommendations for all devices that implement embedded switch technology (whether implemented e.g. via commercially-available chips, FPGA, ASIC).

– ISO/IEC 8802-3 operation:

- auto-negotiation, with 10/100 Mbit/s, full/half duplex (required);

- forced setting of speed/duplex (required);
- ISO/IEC 8802-3 full duplex flow control (recommended).

– Auto MDIX (medium dependent interface crossover), in both auto-negotiate and forced speed/duplex modes (required):

NOTE This is a PHY and transformer issue, not an embedded switch issue.

- QoS:
  a) 2 queues (required); 4 (recommended);
  b) high priority queue for DLR frames, with strict priority scheduling for the high priority queue (required);
  c) prioritization via IEEE 802.1D/IEEE 802.1Q (required) and DSCP (highly recommended). Usage shall be consistent with the CP 2/2 QoS scheme specified in 7.9.6.4. For IP frames the embedded switch shall use the DSCP value. For non-IP frames the priority in the IEEE 802.1Q header shall be used.

– Broadcast rate limiting for host CPU (recommended). The broadcast threshold tolerated by a device is dependent on the host CPU. As a general recommendation, the broadcast rate limiting should be triggered when the broadcast traffic exceeds 1 % of link transmit time;

– Filtering of incoming unicast and multicast to host CPU (recommended, but in practice most all devices will require this).

### 10.6.2 DLR implementation requirements

The following implementation requirements apply to DLR nodes, whether ring supervisors or ring nodes:

- preserve IEEE 802.1Q VLAN Id and tag priority of ring protocol frames;
- disable IP multicast filtering on ring ports or flush multicast filtering table of ring ports on ring state transitions;
- configure multicast address for Beacon frames to be forwarded on ring ports, and to the host CPU for Beacon-based implementations;
- configure multicast address for Announce and Locate_Fault frames to be forwarded to the host CPU and on ring ports;
- configure multicast address for Neighbor_Check request/Response and Sign_On to be forwarded only to host CPU;
  – mechanism to flag the port through which such a frame was received from ring;
  – mechanism to forward such frames from host CPU on to ring only through the port it was intended to go out;
- configure unicast MAC address of active ring supervisor so that supervisor frames forwarded on both ports;
- flush unicast MAC address tables on ring state transitions (or disable learning);
- configure unicast MAC address of self so that it is not purged when MAC address table is flushed;
- implement the Interface Counters and Media Counters attributes of the Ethernet Link object (see 7.6) to aid in network monitoring;
- implement the QoS Object with, at a minimum, DSCP marking of CP 2/2 traffic generated by the device;
- recommended: configure access control list or another suitable mechanism to remove device's own frames from network when received (e.g., during ring startup/restoration);
- disable ISO/IEC 8802-3 and other hardware flow control mechanisms on ring ports
- the 802.3 methods to determine a link's presence when a port is configured to operate in forced speed/duplex mode frequently yield transient link state transitions when a cable is inserted. To prevent unnecessary ring state transitions during cable insertion, devices

shall ensure that the link is stable before enabling frame forwarding to/from the associated port or generating a Link_Status frame. This may typically be accomplished by adding some debounce (e.g. a delay of 10 ms to 100 ms, depending on hardware implementation) to achieve stable link transitions.

#### 10.6.3 IEC 61588 and CP 2/2.1 considerations

In order to support applications requiring time synchronization, multi-port devices are recommended to support the following capabilities with respect to IEC 61588 and CP 2/2.1:

- Implement IEC 61588 end-to-end transparent clock;
- Devices that also implement IEC 61588 ordinary/boundary clock functionality shall perform path delay measurement using Delay_Req/Delay_Resp frames whenever the ring state or network topology mode changes;
- Devices that implement IEC 61588 ordinary/boundary clock functionality and are connected to the ring network indirectly shall perform path delay measurement using Delay_Req/Delay_Resp frames per the CPF 2 specific IEC 61588 profile signaling message (see Time Sync object in IEC 61158-5-2 and IEC 61158-6-2).

Devices not implementing these features will suffer from poor synchronization accuracy for short periods after a network reconfiguration.

#### 10.6.4 IEEE 802.1D/IEEE 802.1Q STP/RSTP/MSTP considerations

In order for DLR to coexist with IEEE spanning tree protocols STP, RSTP and MSTP, the active ring supervisor shall not forward multicast frames with destination address 01:80:C2:00:00:00 (BPDU frames) from one ring port to other, irrespective of ring state. However, if the active ring supervisor has non-ring ports (e.g., in the case of a switch) it shall forward said multicast frames between those non-ring ports and only between one ring port and those non-ring ports. This behavior shall be implemented to ensure that any loop through the ring other than the one through active ring supervisor is detected correctly by STP/RSTP/MSTP.

### 10.7 Using non-DLR nodes in the ring network

#### 10.7.1 General considerations

The DLR protocol does not, by design, require that all nodes in the ring implement the protocol. Non-DLR nodes may be placed in the ring, provided they support certain required capabilities as outlined in 10.7.2 to 10.7.5.

The end user should be made aware that using non-DLR nodes in the ring can affect performance, lengthening the ring recovery times (see performance analysis in 10.10.4). For best performance, it is highly recommended that all nodes in the ring implement the DLR protocol.

Where the use of non-DLR devices cannot be avoided, it is highly recommended that such devices be connected to the network via a DLR-aware device such as a 3-port switch (2 ports connected to the network, 1 port connected to the non-DLR device).

When using non-DLR nodes directly in the ring, certain capabilities and/or configuration steps are required of those nodes in order for the DLR protocol to function properly. These capabilities and configuration steps are described further in 10.7.2 to 10.7.5.

#### 10.7.2 Non-DLR end devices

Examples of non-DLR devices might include an existing 2-port I/O module or drive that supports embedded switch technology but has not implemented the DLR protocol. Such devices may be inserted directly into the ring network. However in order to ensure proper functioning of the ring, the non-DLR end device shall have the following capabilities.

- Disable unicast MAC address learning (or not employ MAC address learning at all). Since Beacon frames will arrive on both ports with the supervisor's MAC address, address learning will cause the supervisor's MAC to bounce from one port to the next. If the supervisor is also an I/O device, the I/O communications can be interrupted.
- Disable multicast filtering on DLR ring ports. If not disabled, multicast messages may not be forwarded to other ring nodes.
- Support reception of IEEE 802.1Q frames and preserve VLAN Id and tag priority. DLR messages may be dropped if not supported, or not queued properly if tag priority is not preserved.
- Implement priority queues with highest priority queue used for DLR messages, with strict priority scheduling. If not supported, ring recovery performance may be affected.

It is the end user's responsibility to ensure that the non-DLR device supports the above capabilities.

### 10.7.3 Non-DLR switches

It is possible for commercially-available managed Ethernet switches that are not DLR aware to be placed directly in the ring. For example, a user may wish to connect a ring of end devices into a switch in order to connect the end devices into the user's larger network.

In order to simplify the switch configuration, it is advisable to connect the non-DLR switch to the ring via a DLR-aware device such as a simple 3-port switch. When the non-DLR switch is connected directly into the ring, a number of configuration steps are required. The specifics of the configuration steps may vary from vendor to vendor. In general, the configuration steps required are outlined in 10.7.4 and 10.7.5.

Using non-DLR switches can result in loss of unicast frames for some period of time following a ring fault or restoration. After a ring fault/restoration, the switch's MAC learning tables may be invalid as devices are now reachable through different ports. Until the MAC learning tables are updated as a result of devices sending frames, unicast frames may not reach the target devices.

For a CP 2/2 I/O connection, typically one cyclic production cycle will be lost. In some circumstances, more than a single production may be lost. For example, in the case where the T-O RPI is slower than the O-T RPI, O-T packets will not be delivered until the target device sends a packet to update the switch's MAC table. Or, if the device is a polled device (such as an explicit message server), or has only unidirectional connections, the device may not generate packets to update the switch's MAC learning table.

In such cases where undesirable unicast packet loss occurs, the user has several options:

a) configure static unicast MAC addresses for the switch ports connected to the ring;

b) disable unicast MAC learning in the switch;

c) use a DLR-capable device such as a 3-port switch to connect the larger switch to the ring.

As described in 10.2, DLR protocol rings can be used in topologies with switches that are running IEEE Spanning Tree Protocols (STP, RSTP, MSTP). In the most common configurations, as shown in 10.2, DLR rings can be connected to such managed switches without issue.

The DLR protocol does not support any non-DLR loop(s) passing through portions of a DLR ring, irrespective of whether a switch supports DLR or not and irrespective of whether it supports Spanning Tree Protocols. Figure 33 and Figure 34 show examples of unsupported DLR topologies, with non-DLR loops traversing the DLR ring. Such topologies can cause erratic network behavior and can be resolved manually by removing one or more of non-DLR redundant links.

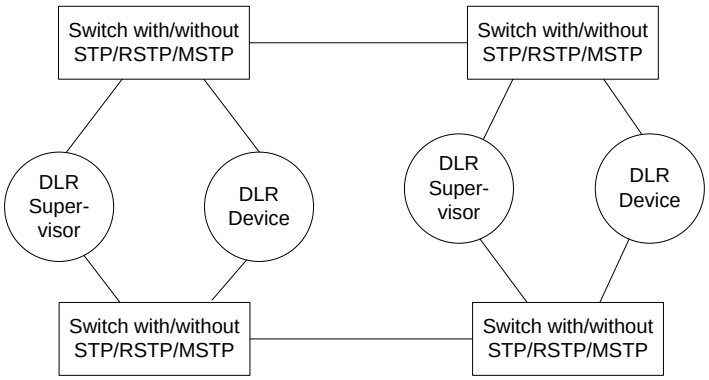



**Figure 33 – Unsupported topology – example 1**

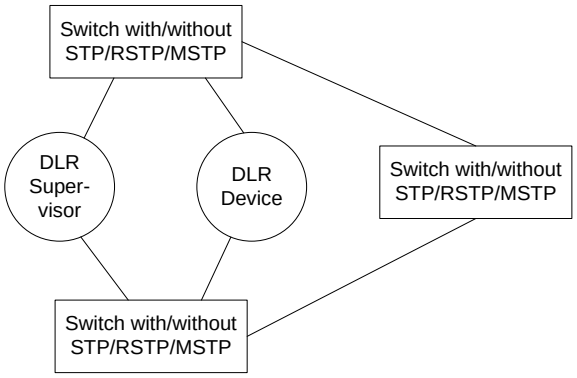



**Figure 34 – Unsupported topology – example 2**

### 10.7.4 Switch configuration for non-VLAN rings

Subclause 10.7.4 outlines the general procedure to configure a managed switch for a non-VLAN based ring network. Actual commands to configure a specific switch is switch dependant, but can be deduced from the procedure below.

a) Configure quality of service (QoS) to ensure deterministic behavior and high performance for ring operation. The switch shall support at least two queues, preferably four, and shall support strict priority scheduling for the highest priority queue. The highest priority queue on ports connected to ring network shall be configured for ring protocol frames (IEEE 802.1Q priority 7). While it is acceptable to share highest priority queue with IEC 61588 PTP event messages, it is strongly recommended that no other traffic be shared on this queue. For IP frames the switch shall use the DSCP value. For non-IP frames the priority in the IEEE 802.1Q header shall be used.

b) If required, configure the two ports of switch connected ring network to preserve IEEE 802.1Q tag priority of ring protocol frames when they pass through the ports. Most switches will need no specific configuration for this step.

c) Disable IP multicast filtering on the two ports of switch connected to ring. This step shall be done to facilitate uninterrupted delivery of CP 2/2 multicast connection data after a ring reconfiguration. Some switches provide a direct way to configure forwarding of all IP multicast traffic on a per port basis. Other switches provide a way to configure designation of individual ports as multicast router ports. Both methods achieve the same effect of disabling multicast filtering on specific ports. See IETF RFC 4541 for details.

d) Statically configure the three multicast addresses used by ring protocol to be forwarded only on two ports of switch connected to ring. This step shall be done to prevent multicast ring protocol frames from being forwarded on other ports of switch.

e) Configure unicast MAC addresses of all configured ring supervisors statically into the MAC table of switch such that unicast traffic destined for ring supervisors will be forwarded through both ports of switch connected to ring. This step shall be done to prevent switch from getting confused by bi-directional ring beacons from active ring supervisor.

### 10.7.5 Switch configuration for VLAN-based ring

Subclause 10.7.5 outlines the general procedure to configure a managed switch for a VLAN-based ring network. The actual configuration commands are switch-dependant, but can be deduced from the procedure below. It is assumed that all ring supervisors have been configured to use VLAN ID *ring_vlan_id* for ring protocol frames. The *ring_vlan_id* will be used only for ring protocol frames. It is assumed that the switch will use VLAN ID *default_vlan_id* for all untagged frame traffic including CP 2/2 frames.

a) Configure quality of service (QoS) to ensure deterministic behavior and high performance for ring operation. The switch shall support at least two queues, preferably four, and shall support strict priority scheduling for the highest priority queue. The highest priority queue on ports connected to ring network shall be configured for ring protocol frames (IEEE 802.1Q priority 7). While it is acceptable to share highest priority queue with IEC 61588 PTP event messages, it is strongly recommended that no other traffic be shared on this queue. For IP frames the switch shall use the DSCP value. For non-IP frames the priority in the IEEE 802.1Q header shall be used.

b) Create VLAN's for *ring_vlan_id* and *default_vlan_id* on switch.

c) Configure the two ports of switch connected ring to participate in VLAN's *ring_vlan_id* and *default_vlan_id*. The two ports shall be configured such that traffic on *default_vlan_id* will go untagged on egress and incoming untagged traffic will be assigned to *default_vlan_id*. The two ports shall be configured to preserve VLAN tag on *ring_vlan_id* when ring protocol frames pass through the ports. Care shall be taken to ensure that none of the other ports on switch participate in VLAN *ring_vlan_id*.

d) Disable IP multicast filtering on the two ports of switch connected to ring, but only for VLAN *default_vlan_id*. This step shall be done to facilitate bump less delivery of CP 2/2 multicast connection data after a ring reconfiguration. Some switches provide a direct way to configure forwarding of all IP multicast traffic on a per port basis. Other switches provide a way to configure designation of individual ports as multicast router ports. Both methods achieve the same effect of disabling multicast filtering on specific ports. See IETF RFC 4541 for details.

e) Configure unicast MAC addresses of all configured ring supervisors statically into the MAC table of switch such that unicast traffic destined for ring supervisors will be forwarded through both ports of switch connected to ring. This configuration shall be done for both VLAN's *ring_vlan_id* and *default_vlan_id*. This step shall be done to prevent switch from getting confused by bi-directional ring beacons from active ring supervisor. This step may be omitted, but will result in minor loss of performance.

## 10.8 Redundant gateway devices on DLR network

### 10.8.1 General

Redundant gateway devices on a DLR network allow multiple connections to the network infrastructure outside of the DLR network. If one of the gateway devices fails or the connection from a gateway device to the outside network infrastructure fails there will be an alternate path for communications. Subclause 10.8 specifies the behavior of redundant gateway devices on a DLR network.

### 10.8.2 Supported topologies

Figure 35 shows a DLR network connected to network infrastructure outside of DLR network through dual redundant gateway devices. In general, two or more redundant gateway devices

on a DLR network are supported. One of the gateway devices will function as the active gateway device, while others function as backup gateway devices. At any given time, only the active gateway device will forward traffic between the DLR network and the network infrastructure outside of the DLR network. The backup gateway devices will block all traffic between the DLR network and the network infrastructure outside of the DLR network, in effect isolating the two networks.

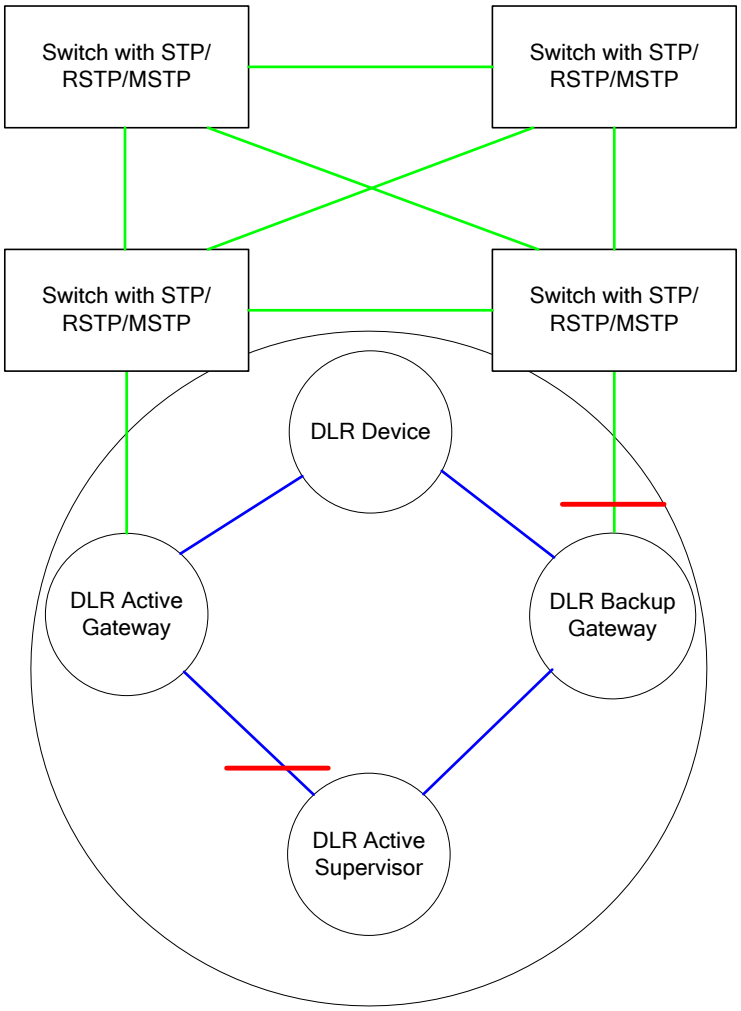



**Figure 35 – DLR ring connected to switches through redundant gateways**

When redundant gateway devices are used on a DLR network to provide multiple connections to network infrastructure outside of the DLR network, DLR aware non-redundant gateway devices and non-DLR aware switches shall not be used on the DLR network to connect to network infrastructure outside of the DLR network.

### 10.8.3 Redundant gateway capable device

Figure 36 shows the conceptual elements of a DLR redundant gateway capable device.

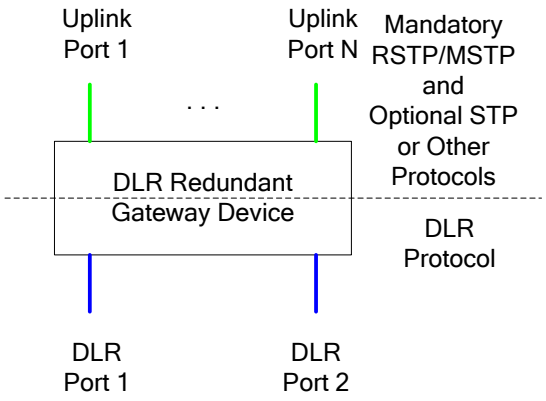



**Figure 36 – DLR redundant gateway capable device**

It has two DLR ports to connect to the DLR network and one or more uplink ports to connect to network infrastructure outside of the DLR network. All the non-DLR ports on a gateway device may not be uplink ports. Whether a non-DLR port can behave as an uplink port is dependent on both implementation and end user configuration.

The gateway device shall implement DLR protocol on its two DLR ports. It shall implement either IEEE 802.1D RSTP or IEEE 802.1Q MSTP on its uplink ports. It may optionally implement STP or other protocols on its uplink ports. It shall have the ability to block traffic from being forwarded between DLR ports and uplink ports when functioning as a backup gateway device. While blocking traffic in backup mode, it shall forward DLR traffic only between its two DLR ports and shall forward uplink port traffic only between its uplink ports. There are many ways to implement such blocking behavior. For example one way is to use additional VLAN tagging inside the gateway device.

Irrespective of whether a gateway is blocking or forwarding traffic between DLR ports and uplink ports, it shall never forward DLR protocol frames with the exception of Learning_Update frames between DLR ports and uplink ports. Only the active gateway device shall forward Learning_Update frames to uplink ports when traffic forwarding is enabled between DLR ports and uplink ports as specified in 10.10.4.

When a gateway switchover occurs, the new active gateway device shall send the topology change notification message on its uplink ports that is appropriate for the protocol currently enabled on its uplink ports. This requirement does not apply if no protocol is currently enabled on its uplink ports.

### 10.8.4 Redundant gateway device behavior

#### 10.8.4.1 General

A redundant gateway device shall implement two independent state machines. One state machine shall be for base DLR function, i.e. a Beacon based ring node or an Announce based ring node or a DLR supervisor as appropriate for the device capability. A second state machine shall be implemented for the redundant gateway function. Subclause 10.8.4 provides a general overview of redundant gateway device behavior. See 10.10.4 for details.

#### 10.8.4.2 Variables

Table 135 summarizes variables used in the redundant gateway behavior and messages. The DLR Object (see 7.9) exposes these variables (with the exception of the Gateway State and Active Listen Timeout) via object attributes.

**Table 135 – Redundant gateway variables**

| Gateway variable | Description |
|---|---|
| Gateway State | Internal state of a node's gateway state machine.<br>IDLE_STATE – initial state for gateways and when redundant gateway function has not been enabled.<br>ACTIVE_LISTEN_STATE – when a gateway is actively transmitting Advertise frames and is also listening for Advertise frames to detect the presence of another active gateway with traffic forwarding blocked between its DLR ports and uplink ports.<br>ACTIVE_NORMAL_STATE – normal function in active gateway mode.<br>BACKUP_NORMAL_STATE – normal function in backup gateway mode.<br>FAULT_STATE – while uplink connection fault persists. |
| Redundant Gateway Enable | A Boolean value to indicate if redundant gateway function is enabled on a redundant gateway capable device. The default value is FALSE. |
| Gateway Precedence | Precedence value assigned to a gateway, and transmitted in Advertise frames. Used to select active gateway when multiple gateways have been configured.<br>Default value is 0. Can be changed via the DLR Object. |
| Advertise Interval | Interval at which the active gateway node sends Advertise frames. Gateways shall support a range from 1000 microseconds to 100 milliseconds. The default value shall be 2 000 microseconds. Gateways may support an Advertise Interval smaller than 1 000 microseconds, but this is not required. The absolute minimum Advertise Interval is 200 microseconds. |
| Advertise Timeout | Amount of time backup gateway nodes shall wait before timing out reception of Advertise frames and taking the appropriate action. Gateways shall support a range from 2 500 microseconds to 500 milliseconds. The default shall be 5 000 microseconds. Gateways may support an Advertise Timeout smaller than 2 500 microseconds, but this is not required. The absolute minimum Advertise Timeout is 500 microseconds. |
| Learning Update Enable | A Boolean value to indicate if all nodes in the DLR network should transmit Learning_Update frames when they receive a Flush_Tables frame from an active gateway. This parameter is encoded by the active gateway in its Flush_Tables frame. The Learning_Update frames from DLR devices accelerate the new network topology learning by all non-DLR switches outside the DLR network after an active gateway switchover. The default value is TRUE. |
| Active Listen Timeout | Amount of time a gateway device shall wait in ACTIVE_LISTEN_STATE listening for Advertise frames with higher precedence from another gateway device. The value is equal to current Advertise Timeout duration. |

#### 10.8.4.3 Startup

A gateway device shall start (on power up or when redundant gateway function is enabled) in IDLE_STATE with traffic forwarding blocked between DLR ports and uplink ports. If redundant gateway function is enabled, it shall transition to ACTIVE_LISTEN_STATE with traffic forwarding blocked between its DLR ports and uplink ports. If redundant gateway function is disabled, it shall enable traffic forwarding between its DLR ports and uplink ports.

Immediately upon transition to ACTIVE_LISTEN_STATE and once every Advertise Interval thereafter, the gateway shall transmit Advertise frames through its DLR ports (see Figure 37). It shall also listen for Advertise frames transmitted by another active gateway device on its DLR ports. If an Advertise frame is received from another active gateway device with higher Precedence or in case of a tie, if the source MAC address of the other gateway is numerically greater than its own MAC address, it shall transition to BACKUP_NORMAL_STATE with traffic forwarding blocked between its DLR ports and uplink ports, and shall stop transmission of Advertise frames.

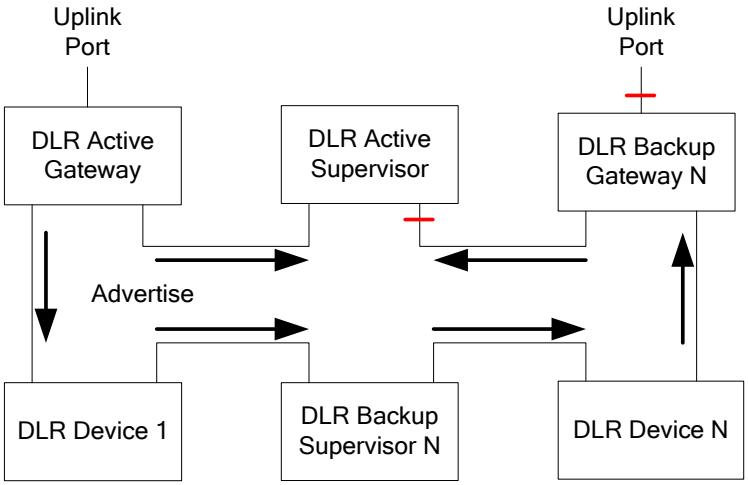



**Figure 37 – Advertise frame**

#### 10.8.4.4 Normal operation

While in ACTIVE_LISTEN_STATE, if no Advertise frames are received or no Advertise frames with higher Precedence (or numerically higher source MAC address in case of a tie) are received for the duration of Active Listen Timeout, it shall transition to ACTIVE_NORMAL_STATE and shall become the active gateway device with traffic forwarding enabled between its DLR ports and uplink ports. Immediately upon transition to ACTIVE_NORMAL_STATE and once every Advertise Interval thereafter, the gateway shall transmit Advertise frames through its DLR ports.

Immediately upon transition to ACTIVE_NORMAL_STATE, an active gateway device shall transmit a Flush_Tables frame to all DLR nodes and shall flush its own unicast and multicast address learning filter tables. Upon receiving a Flush_tables frame, a DLR node shall flush its unicast and multicast MAC address learning tables. If the Learning Update Enable field in the Flush_Tables frame is set to TRUE, a DLR node shall also transmit a Learning_Update frame to accelerate new network topology learning by non-DLR switches outside of the DLR network.

While in ACTIVE_NORMAL_STATE, if an Advertise frame is received from another active gateway device with higher Precedence (or numerically higher source MAC address in case of a tie), the receiver shall transition to BACKUP_NORMAL_STATE with traffic forwarding blocked between its DLR ports and uplink ports, and shall stop transmission of Advertise frames.

#### 10.8.4.5 Uplink fault detection

While in ACTIVE_LISTEN_STATE or BACKUP_NORMAL_STATE, if the physical connections are lost on all its uplink ports or a higher level connection fault is detected through an optional protocol implemented on the uplink ports, a gateway device shall transition to FAULT_STATE with traffic forwarding blocked between its DLR ports and uplink ports. It shall not transmit Advertise frames in FAULT_STATE.

While in ACTIVE_NORMAL_STATE, if the physical connections are lost on all its uplink ports or a higher level connection fault is detected through an optional protocol implemented on the uplink ports, an active gateway device shall transmit an Advertise frame with FAULT_STATE encoded in the frame and shall transition to FAULT_STATE with traffic forwarding blocked between its DLR ports and uplink ports. It shall not transmit Advertise frames in FAULT_STATE thereafter.

#### 10.8.4.6 Gateway switchover

While in BACKUP_NORMAL_STATE, if an Advertise frame is received from an active gateway with FAULT_STATE encoded or if no Advertise frames are received from any active gateway for the duration of the Advertise Timeout, a backup gateway device shall transition to ACTIVE_LISTEN_STATE with traffic forwarding blocked between its DLR ports and uplink ports. It shall then behave as described in 10.8.4.3 for ACTIVE_LISTEN_STATE.

#### 10.8.4.7 Uplink restoration

When a faulted uplink connection is restored and the Precedence of the active gateway is higher than Precedence of itself (or has a numerically higher MAC address than its own MAC address in case of a tie), a gateway device shall transition from FAULT_STATE to BACKUP_NORMAL_STATE and shall block traffic forwarding between its DLR ports and uplink ports.

When a faulted uplink connection is restored and the Precedence of active gateway is lower than Precedence of itself (or has a numerically lower MAC address than its own MAC address in case of a tie) or there is no active gateway transmitting Advertise frames, a gateway device shall transition from FAULT_STATE to ACTIVE_LISTEN_STATE and shall block traffic forwarding between its DLR ports and uplink ports. It shall then behave as described in 10.8.4.3 for ACTIVE_LISTEN_STATE.

#### 10.8.4.8 Partial network fault

It is possible for a partial network fault to occur such that traffic is lost in only one direction. In such a situation, it is possible for a backup gateway device to misdiagnose that the active gateway device is lost. To prevent multiple gateway devices from enabling traffic forwarding between their DLR ports and uplink ports, an active gateway device shall transition to FAULT_STATE and shall block traffic forwarding between its DLR ports and uplink ports if it receives an Advertise frame with ACTIVE_NORMAL_STATE as the gateway state from another active gateway device with lower Precedence (or numerically lower MAC address in case of a tie). It shall indicate the condition through the Redundant Gateway Status attribute in the DLR object. It shall set the Active Gateway Address and Active Gateway Precedence attributes to that of the other active gateway with lower precedence. The user shall explicitly clear this condition via the DLR object using the Clear_Gateway_Partial_Fault service.

### 10.9 DLR messages

#### 10.9.1 General

Subclauses 10.9.2 to 10.9.9 specify the DLR protocol messages. Several items of importance should be noted:

a) DLR messages, with the exception of Learning_Update, are sent using the IEEE 802.1Q frame format. Messages shall be transmitted using the highest priority (7), which shall be preserved as the DLR frames are transferred through the LAN.

b) To distinguish the DLR frame data types from CIP data types, the UINTx convention is used to indicate an x-bit, unsigned integer value in tables included in 10.8. Multi-octet data types within the DLR frames shall be encoded high order octet first (big endian format).

c) Table 136 specifies the destination MAC addresses used for DLR messages.

**Table 136 – MAC addresses for DLR messages**

| MAC address | Usage |
|---|---|
| 01-21-6C-00-00-01 | Beacon |
| 01-21-6C-00-00-02 | Neighbor_Check request, Neighbor_Check response, Sign_On |
| 01-21-6C-00-00-03 | Announce, Locate_Fault, Flush_Tables |

| MAC address | Usage |
|---|---|
| 01-21-6C-00-00-04 | Advertise |
| 01-21-6C-00-00-05 | Learning_Update |
| MAC address of active ring supervisor | Link_Status / Neighbor_Status |

### 10.9.2 Common frame header

DLR messages, with the exception of Learning_Update, shall be sent using the IEEE 802.1Q frame format, as specified in Table 137.

**Table 137 – IEEE 802.1Q common frame header format**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 0 | Destination MAC Address | UINT8[6] | |
| 6 | Source MAC Address | UINT8[6] | |
| 12 | IEEE 802.1Q Tag Type | UINT16 | = 0x8100 |
| 14 | IEEE 802.1Q Tag control | UINT16 | = 0xE000 + VLAN_ID (default value = 0) |
| 16 | Ring EtherType | UINT16 | = 0x80E1 |
| 18 | Ring Sub-type | UINT8 | = 0x02 |
| 19 | Ring Protocol Version | UINT8 | = 0x01 |

Table 138 specifies the common fields that are used in the DLR message payload.

**Table 138 –DLR message payload fields**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | Identifies the DLR message type (see Table 139) |
| 21 | Source Port | UINT8 | Identifies the port on the device through which this message originated:<br>0x00 – Port 1 or Port 2<br>0x01 – Port 1<br>0x02 – Port 2 |
| 22 | Source IP Address | UINT32 | Source IP address of the sending node. If none is configured, the value shall be 0 |
| 26 | Sequence Id | UINT32 | DLR message sequence Id. When originating a message, a node shall increment the Sequence Id by 1. When responding to a message (Neighbor_Check response and Sign_On), a node shall use the Sequence Id from the original request message |
| 30 | Dependent on DLR message type | | See Table 140 to Table 148 |
| ... | FCS | UINT32 | ISO/IEC 8802-3 FCS |

**Table 139 – DLR frame types**

| Value | Name | Frame format |
|---|---|---|
| 0x01 | Beacon | Common Frame Format<br>See Table 137 and Table 138 |
| 0x02 | Neighbor_Check_Request | |
| 0x03 | Neighbor_Check_Response | |
| 0x04 | Link_Status / Neighbor_Status | |
| 0x05 | Locate_Fault | |

| Value | Name | Frame format |
|---|---|---|
| 0x06 | Announce | |
| 0x07 | Sign_On | |
| 0x08 | Advertise | |
| 0x09 | Flush_Tables | |
| 0x0A | Learning_Update | See 10.9.12 |

### 10.9.3 Beacon frame

The Beacon frame is sent by the active ring supervisor to determine the health of the ring topology. Table 140 specifies the format of the Beacon frame.

**Table 140 – Format of the Beacon frame**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x01 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 26 | Sequence Id | UINT32 | |
| 30 | Ring State | UINT8 | See Table 141 |
| 31 | Supervisor Precedence | UINT8 | From the DLR object |
| 32 | Beacon Interval | UINT32 | From the DLR object |
| 36 | Beacon Timeout | UINT32 | In microseconds (µs) |
| 40 | Reserved | UINT8[20] | Sender shall set to zero, receiver shall ignore |
| 60 | FCS | UINT32 | |

Table 141 specifies the ring state values.

**Table 141 – Ring State values**

| Ring State | Value |
|---|---|
| RING_NORMAL_STATE | 0x01 |
| RING_FAULT_STATE | 0x02 |

### 10.9.4 Neighbor_Check request

The Neighbor_Check_request is sent by a ring node as a result of receiving a Locate_Fault frame, and by an active ring supervisor when Beacon loss has been detected. The format of the Neighbor_Check_request is specified in Table 142.

**Table 142 – Format of the Neighbor_Check request**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x02 |
| 21 | Source Port | UINT8 | = 0x1 or 0x2 |
| 22 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 26 | Sequence Id | UINT32 | |
| 30 | Reserved | UINT8[30] | Sender shall set to zero, receiver shall ignore |
| 60 | FCS | UINT32 | |

### 10.9.5 Neighbor_Check_response

The Neighbor_Check_response frame is sent in response to the Neighbor_Check_request. The format of the Neighbor_Check_response is specified in Table 143.

**Table 143 – Format of the Neighbor_Check response**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x03 |
| 21 | Source Port | UINT8 | = 0x01 or 0x02 |
| 22 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 26 | Sequence Id | UINT32 | = Sequence Id of Neighbor_Check request |
| 30 | Request Source Port | UINT8 | = Source Port of Neighbor_Check request |
| 31 | Reserved | UINT8[29] | Sender shall set to zero, receiver shall ignore |
| 60 | FCS | UINT32 | |

### 10.9.6 Link_Status/Neighbor_Status

A Link_Status frame is sent when a ring node has detected a link failure or in response to a Locate_Fault frame when one of its ports is not active. A Neighbor_Status frame is sent when the Neighbor Check process has timed out. These frames share the same format, differentiated by the Link/Neighbor Status Flag, which is specified in Table 144.

**Table 144 – Format of the Link_Status/Neighbor_Status frame**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x04 |
| 21 | Source Port | UINT8 | = 0 |
| 22 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 26 | Sequence Id | UINT32 | |
| 30 | Link/Neighbor Status | UINT8 | Interpreted as a bit map, see Table 145 |
| 31 | Reserved | UINT8[29] | Sender shall set to zero, receiver shall ignore |
| 60 | FCS | UINT32 | |

Table 145 specifies the Link/Neighbor status values.

**Table 145 – Link/Neighbor status values**

| Bit(s) | Name | Definition |
|---|---|---|
| 0 (least significant) | Port 1 active | Set to 1 if node's port 1 is active |
| 1 | Port 2 active | Set to 1 if node's port 2 is active |
| 2-6 | Reserved | Sender shall set to zero, receiver shall ignore |
| 7 | Link/Neighbor Status flag | Set to 0 if frame is Link_Status frame<br>Set to 1 if frame is Neighbor_Status frame |

### 10.9.7 Locate_Fault

The Locate_Fault frame is sent by the active ring supervisor as a result of loss of Beacon frames, and as a result of the Verify_Fault_Location service sent to the DLR object. The Locate_Fault frame format is specified in Table 146.

**Table 146 – Format of the Locate_Fault frame**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x05 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 26 | Sequence Id | UINT32 | |
| 30 | Reserved | UINT8[30] | Sender shall set to zero, receiver shall ignore |
| 60 | FCS | UINT32 | |

### 10.9.8 Announce

The Announce frame is sent by the active ring supervisor to indicate a change in the ring state, and to notify the ring topology to announce-based nodes. The Announce frame format is specified in Table 147.

**Table 147 – Format of the Announce frame**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x06 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 26 | Sequence Id | UINT32 | |
| 30 | Ring State | UINT8 | As in the Beacon frame |
| 31 | Reserved | UINT8[29] | Sender shall set to zero, receiver shall ignore |
| 60 | FCS | UINT32 | |

### 10.9.9 Sign_On

The Sign_On frame is sent initially by the active ring supervisor, in order to identify all participating ring nodes. The receiving ring participant node traps the Sign_On frame and forwards it only to the host CPU. The host CPU increments the number of nodes in list, add its

own addresses to list and transmits the Sign_On frame only through the other port than the receiving port. The Sign_On frame format is specified in Table 148.

**Table 148 – Format of the Sign_On frame**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x07 |
| 21 | Source Port | UINT8 | = 0x01 or 0x02 |
| 22 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 26 | Sequence Id | UINT32 | |
| 30 | Number of Nodes in List | UINT16 | |
| 32 | Node 1 MAC Address | UINT8[6] | Node 1 is always the active supervisor |
| 38 | Node 1 IP Address | UINT32 | = 0x0, if no IP address |
| 42 | Node 2 MAC Address | UINT8[6] | |
| 48 | Node 2 IP Address | UINT32 | = 0x0, if no IP address |
| … | Reserved | UINT8[…] | Sender shall set to zero, receiver shall ignore |
| N | FCS | UINT32 | N shall be padded to at least 60, if needed |

### 10.9.10 Advertise

The Advertise frame is sent by the active gateway to indicate presence of active gateway and to notify gateway faults. The Advertise frame format is specified in Table 149.

**Table 149 – Format of the Advertise frame**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x08 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 26 | Sequence ID | UINT32 | |
| 30 | Gateway State | UINT8 | See Table 150 |
| 31 | Gateway Precedence | UINT8 | From the DLR object |
| 32 | Advertise Interval | UINT32 | From the DLR object |
| 36 | Advertise Timeout | UINT32 | From the DLR object |
| 40 | Learning Update Enable | UINT8 | From the DLR object |
| 41 | Reserved | UINT8[19] | |
| 60 | FCS | UINT32 | |

**Table 150 – Gateway state values**

| Gateway state | Value |
|---|---|
| ACTIVE_LISTEN_STATE | 0x01 |
| ACTIVE_NORMAL_STATE | 0x02 |
| FAULT_STATE | 0x03 |

### 10.9.11 Flush_Tables

The Flush_Tables frame is sent by the new active gateway to indicate that DLR nodes should flush their unicast and multicast MAC address learning tables. The Flush_Tables frame format is specified in Table 151.

**Table 151 – Format of the Flush_Tables frame**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x09 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 26 | Sequence Id | UINT32 | |
| 30 | Learning Update Enable | UINT8 | From the DLR object |
| 31 | Reserved | UINT8[29] | |
| 60 | FCS | UINT32 | |

### 10.9.12 Learning_Update

The Learning_Update frame is sent by all DLR nodes upon receiving a Flush_Tables frame from active gateway and if Learning Update Enable field is set to TRUE to accelerate learning of new network topology by non-DLR switches outside of DLR network. The Learning_Update frame is an untagged IEEE 802.3 frame as shown below, so that VLAN aware non-DLR switches outside of DLR network will not drop it. The Learning_Update frame format is specified in Table 152.

**Table 152 – Format of the Learning_Update frame**

| Octet | Field | Type | Remarks |
|---|---|---|---|
| 0 | Destination MAC Address | UINT8[6] | |
| 6 | Source MAC Address | UINT8[6] | |
| 12 | Ring EtherType | UINT16 | = 0x80E1 |
| 14 | Ring Sub-type | UINT8 | = 0x02 |
| 15 | Ring Protocol Version | UINT8 | = 0x01 |
| 16 | Frame Type | UINT8 | = 0x0A |
| 17 | Source Port | UINT8 | = 0x0 |
| 18 | Source IP Address | UINT32 | = 0x0, if source has no IP address |
| 22 | Sequence ID | UINT32 | |
| 26 | Reserved | UINT8[34] | |
| 60 | FCS | UINT32 | |

## 10.10 State diagrams and state-event-action matrices

### 10.10.1 Beacon-based ring node

Figure 38 shows the state transition diagram for the beacon frame based non-supervisor ring node.

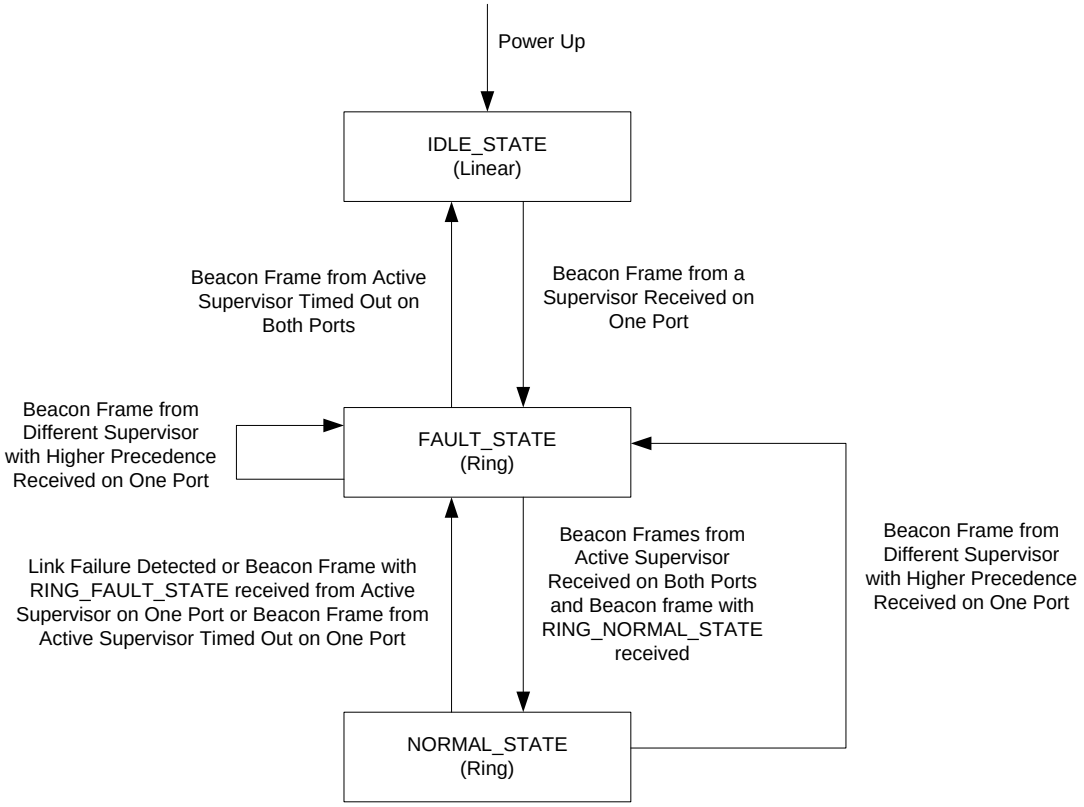



**Figure 38 – State transition diagram for Beacon frame based non-supervisor ring node**

Table 153 specifies the parameter values for the beacon frame for a non-supervisor ring node.

**Table 153 – Parameter values for Beacon frame based non-supervisor ring node**

| Parameter | Value |
|---|---|
| Active supervisor precedence | Obtained from Beacon frame |
| Active supervisor MAC address | Obtained from Beacon frame |
| Ring protocol VLAN ID | Obtained from Beacon frame |
| Beacon timeout duration | Obtained from Beacon frame |
| Neighbor check timeout duration | 100 ms |
| Maximum retry limit for Neighbor_Check request frame | 3 (total number of tries) |

The following requirements apply to the state-event-action matrix.

- LastBcnRcvPort is a bit string variable used to track port(s) on which last Beacon frame was received, with bit definitions specified in Table 154.
- MAC address 11-22-33-44-55-66 shall be encoded as 0x112233445566 for numerical comparison in case of supervisor precedence tie.
- Nodes shall statically configure unicast MAC address of current active ring supervisor into unicast MAC learning table to be forwarded only through both ring ports.

**Table 154 – LastBcnRcvPort bit definitions**

| Bit | Definition |
|---|---|
| 0 | Set if a beacon frame from the active supervisor has been received on Port 1 |
| 1 | Set if a beacon frame from the active supervisor has been received on Port 2 |
| 2-N | Not used; set to 0 |

Table 155 specifies the state-event-action matrix for the Beacon frame based non-supervisor ring node.

**Table 155 – State-event-action matrix for Beacon frame based non-supervisor ring node**

| Event No. | Current state | Event | Action(s)[a,b] |
|---|---|---|---|
| 1 | None | Power up | Initialize and transition to IDLE_STATE |
| 2 | IDLE_STATE | Beacon frame from a ring supervisor received on port 1 | Save as active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 1.<br>Start beacon timeout timer for port 1.<br>Transition to FAULT_STATE and flush unicast MAC address learning table |
| 3 | IDLE_STATE | Beacon frame from a ring supervisor received on port 2 | Save as active supervisor precedence, MAC address, VLAN ID and beacon timeout.<br>Set LastBcnRcvPort to 2.<br>Start beacon timeout timer for port 2.<br>Transition to FAULT_STATE and flush unicast MAC address learning table |
| 4 | IDLE_STATE | Link lost on port 1 | Stop forwarding frames on port 1 |
| 5 | IDLE_STATE | Link lost on port 2 | Stop forwarding frames on port 2 |
| 6 | IDLE_STATE | Link restored on port 1 | Start forwarding frames on port 1 |
| 7 | IDLE_STATE | Link restored on port 2 | Start forwarding frames on port 2 |
| 8 | IDLE_STATE | Frame from self received on port 1 or port 2 | Do not forward frame on network and drop it (If capable of doing so) |
| 9 | IDLE_STATE | Locate_Fault or Neighbor_Check request or Neighbor_Check response or Sign_On frame received on port 1 or port 2 | Ignore |
| 10 | FAULT_STATE | Beacon frame from active ring supervisor received on port 1 and LastBcnRcvPort is 1 | Restart port 1 beacon timeout timer |
| 11 | FAULT_STATE | Beacon frame from active ring supervisor received on port 2 and LastBcnRcvPort is 2 | Restart port 2 beacon timeout timer |
| 12 | FAULT_STATE | Beacon timeout timer expired for port 1 and LastBcnRcvPort is 1 | Stop neighbor check timeout timers for both ports, transition to IDLE_STATE and flush unicast MAC address learning table |
| 13 | FAULT_STATE | Beacon timeout timer expired for port 2 and LastBcnRcvPort is 2 | Stop neighbor check timeout timers for both ports, transition to IDLE_STATE and flush unicast MAC address learning table |

| Event No. | Current state | Event | Action(s)[a,b] |
|---|---|---|---|
| 14 | FAULT_STATE | Beacon frame from active ring supervisor received on port 2 and LastBcnRcvPort is 1 or 3 | Set LastBcnRcvPort to 3<br>Start/restart port 2 beacon timeout timer<br>If the ring state of beacon frame is notRING_NORMAL_STATE, return.<br>Else, stop neighbor check timeout timers for both ports.<br>Transition to NORMAL_STATE and flush unicast MAC address learning table |
| 15 | FAULT_STATE | Beacon frame from active ring supervisor received on port 1 and LastBcnRcvPort is 2 or 3 | Set LastBcnRcvPort to 3<br>Start/restart port 1 beacon timeout timer<br>If the ring state of beacon frame is notRING_NORMAL_STATE, return.<br>Else, stop neighbor check timeout timers for both ports.<br>Transition to NORMAL_STATE and flush unicast MAC address learning table |
| 16 | FAULT_STATE | Beacon frame from different ring supervisor MAC address than active ring supervisor is received on port 1 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon timeout timers for both ports.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 1.<br>Start beacon timeout timer for port 1.<br>Stay in FAULT_STATE and flush unicast MAC address learning table |
| 17 | FAULT_STATE | Beacon frame from different ring supervisor MAC address than active ring supervisor is received on port 2 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon timeout timers for both ports.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 2.<br>Start beacon timeout timer for port 2.<br>Stay in FAULT_STATE and flush unicast MAC address learning table |
| 18 | FAULT_STATE | Link lost on port 1 and LastBcnRcvPort is 1 | Stop port 1 beacon timeout timer and neighbor check timeout timers for both ports.<br>Transition to IDLE_STATE and flush unicast MAC address learning table |
| 19 | FAULT_STATE | Link lost on port 1 and LastBcnRcvPort is 2 | Send Link_Status frame to active ring supervisor |
| 20 | FAULT_STATE | Link lost on port 2 and LastBcnRcvPort is 2 | Stop port 2 beacon timeout timer and neighbor check timeout timers for both ports.<br>Transition to IDLE_STATE and flush unicast MAC address learning table |
| 21 | FAULT_STATE | Link lost on port 2 and LastBcnRcvPort is 1 | Send Link_Status frame to active ring supervisor. |
| 22 | FAULT_STATE | Link restored on port 1 | Start forwarding frames on port 1 |
| 23 | FAULT_STATE | Link restored on port 2 | Start forwarding frames on port 2 |

| Event No. | Current state | Event | Action(s)[a,b] |
|---|---|---|---|
| 24 | FAULT_STATE | Locate_Fault frame received from active ring supervisor on port 1 or port 2 | If link is not active on port 1 or port 2 send Link_Status frame to active ring supervisor.<br>Else clear neighbor status for port 1 and 2, send Neighbor_Check_Request for port 1 and 2, and start neighbor check timeout timer for port 1 and 2. |
| 25 | FAULT_STATE | Neighbor_Check request frame received on port 1 | Send Neighbor_Check response frame on port 1 |
| 26 | FAULT_STATE | Neighbor_Check request frame received on port 2 | Send Neighbor_Check response frame on port 2 |
| 27 | FAULT_STATE | Neighbor_Check response frame received on port 1 | Stop neighbor check timeout timer for port 1 and save neighbor status for port 1 |
| 28 | FAULT_STATE | Neighbor_Check response frame received on port 2 | Stop neighbor check timeout timer for port 2 and save neighbor status for port 2 |
| 29 | FAULT_STATE | Neighbor check timer timeout on port 1 | If number of retries have not exceeded maximum limit, send Neighbor_Check request on port 1 and start neighbor check timeout timer for port 1.<br>Else, send Neighbor_Status frame to active ring supervisor |
| 30 | FAULT_STATE | Neighbor check timer timeout on port 2 | If number of retries have not exceeded maximum limit, send Neighbor_Check request on port 2 and start neighbor check timeout timer for port 2.<br>Else, send Neighbor_Status frame to active ring supervisor |
| 31 | FAULT_STATE | Sign_On frame received on port 1 or port 2 | Ignore |
| 32 | NORMAL_STATE | Link lost on port 1 | Send Link_Status frame to active ring supervisor.<br>Stop beacon timeout timer for port 1.<br>Set LastBcnRcvPort to 2, transition to FAULT_STATE and flush unicast MAC address learning table |
| 33 | NORMAL_STATE | Link lost on port 2 | Send Link_Status frame to active ring supervisor.<br>Stop beacon timeout timer for port 2.<br>Set LastBcnRcvPort to 1, transition to FAULT_STATE and flush unicast MAC address learning table |
| 34 | NORMAL_STATE | Beacon frame from active ring supervisor with RING_FAULT_STATE received on port 1 | If the ring state of previous beacon frame received on port 1 was not RING_NORMAL_STATE, return.<br>Else, stop beacon timeout timer for port 2.<br>Set LastBcnRcvPort to 1, transition to FAULT_STATE and flush unicast MAC address learning table |
| 35 | NORMAL_STATE | Beacon frame from active ring supervisor with RING_FAULT_STATE received on port 2 | If the ring state of previous beacon frame received on port 2 was not RING_NORMAL_STATE, return.<br>Else, stop beacon timeout timer for port 1.<br>Set LastBcnRcvPort to 2, transition to FAULT_STATE and flush unicast MAC address learning table |
| 36 | NORMAL_STATE | Beacon timeout timer expired for port 1 | Set LastBcnRcvPort to 2, transition to FAULT_STATE and flush unicast MAC address learning table |
| 37 | NORMAL_STATE | Beacon timeout timer expired for port 2 | Set LastBcnRcvPort to 1, transition to FAULT_STATE and flush unicast MAC address learning table |

| Event No. | Current state | Event | Action(s)[a,b] |
|---|---|---|---|
| 38 | NORMAL_STATE | Beacon frame from active ring supervisor received on port 1 | Restart port 1 beacon timeout timer |
| 39 | NORMAL_STATE | Beacon frame from active ring supervisor received on port 2 | Restart port 2 beacon timeout timer |
| 40 | NORMAL_STATE | Beacon frame from different ring supervisor MAC address than active ring supervisor is received on port 1 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon timeout timers for both ports.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 1.<br>Start beacon timeout timer for port 1.<br>Transition to FAULT_STATE and flush unicast MAC address learning table |
| 41 | NORMAL_STATE | Beacon frame from different ring supervisor MAC address than active ring supervisor is received on port 2 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon timeout timers for both ports.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 2.<br>Start beacon timeout timer for port 2.<br>Transition to FAULT_STATE and flush unicast MAC address learning table |
| 42 | NORMAL_STATE | Locate_Fault frame received from active ring supervisor on port 1 or port 2 | Ignore |
| 43 | NORMAL_STATE | Neighbor_Check request or Neighbor_Check response frame received on port 1 or port 2 | Ignore |
| 44 | NORMAL_STATE | Sign_On frame received on port 1 | Add node data to participant list and send frame through port 2 |
| 45 | NORMAL_STATE | Sign_On frame received on port 2 | Add node data to participant list and send frame through port 1 |
| 46 | IDLE_STATE | Flush_Tables frame received | Ignore |
| 47 | NORMAL_STATE | Flush_Tables frame received | Flush unicast and multicast MAC address learning tables.<br>If Learning Update Enable field in Flush_Tables frame is set to TRUE, send Learning_Update frame. |
| 48 | FAULT_STATE0 | Flush_Tables frame received | Flush unicast and multicast MAC address learning tables.<br>If Learning Update Enable field in Flush_Tables frame is set to TRUE, send Learning_Update frame. |

[a] Network Topology attribute shall be updated at events 1, 2, 3, 12, 13, 18 and 20 to appropriate value.

[b] Network Status attribute shall be updated at events 1, 2, 3, 8, 12, 13, 14, 15, 18, 20, 32, 33, 34, 35, 36, 37, 40 and 41 to appropriate value.

### 10.10.2 Announce-based ring node

Figure 39 shows the state transition diagram for Announce frame based non-supervisor ring node.

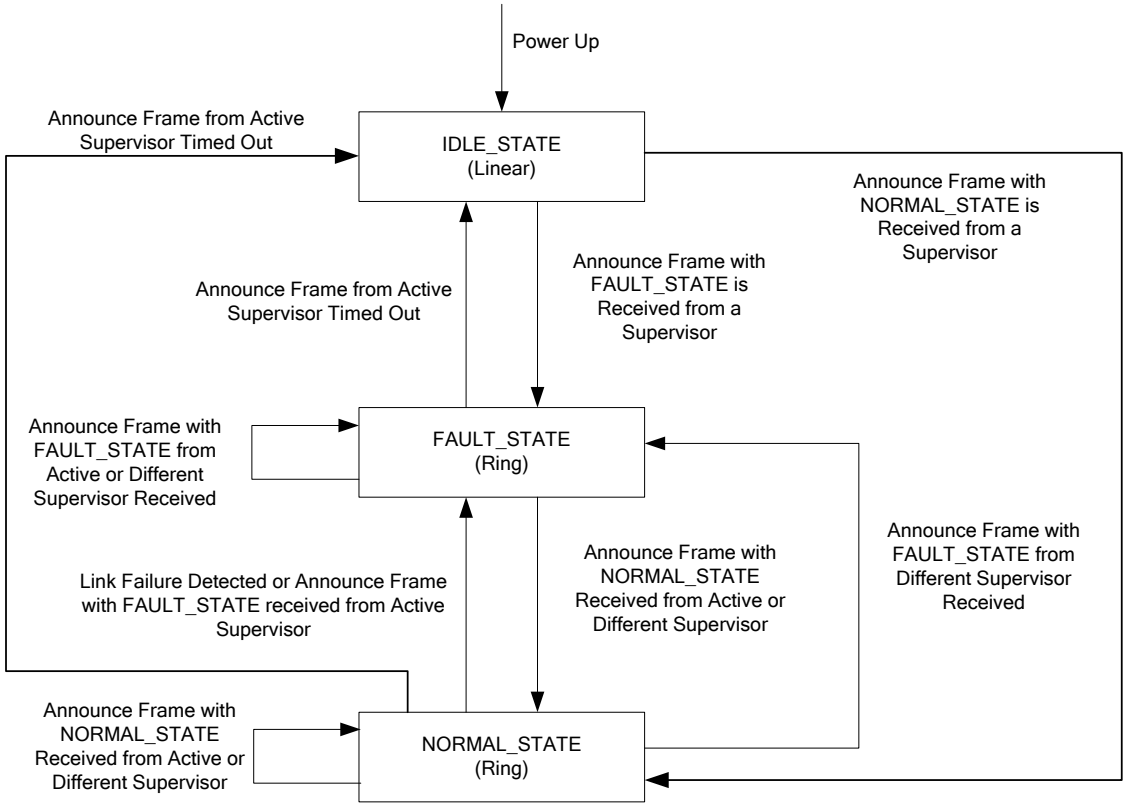



**Figure 39 – State transition diagram for Announce frame based non-supervisor ring node**

Table 156 specifies the parameter values for the Announce frame based non-supervisor ring node.

**Table 156 – Parameter values for Announce frame based non-supervisor ring node**

| Parameter | Value |
|---|---|
| Active supervisor MAC address | Obtained from Announce frame |
| Ring protocol VLAN ID | Obtained from Announce frame |
| Announce timeout duration | 2,5 s |
| Neighbor check timeout duration | 100 ms |
| Maximum retry limit for Neighbor_Check request frame | 3 (total number of tries) |

The following requirements apply to the state-event-action matrix.

- Sequence count shall be verified for all Announce frames from the active ring supervisor using modular unsigned 32-bit integer arithmetic to deal with sequence number rollover. Only Announce frames with sequence count greater than last Announce frame shall be processed. Others shall be silently discarded.
- Nodes shall statically configure unicast MAC address of current active ring supervisor into unicast MAC learning table to be forwarded only through both ring ports.
- Nodes shall statically configure multicast MAC address of beacon frames into MAC learning table to be forwarded only through both ring ports and not to CPU.

Table 157 specifies the state-event-action matrix for the Announce frame based non-supervisor ring nodes.

**Table 157 – State-event-action matrix for Announce frame based non-supervisor ring node**

| Event No. | Current State | Event | Action(s)[a,b] |
|---|---|---|---|
| 1 | None | Power up | Initialize and transition to IDLE_STATE |
| 2 | IDLE_STATE | Announce frame from a ring supervisor received on port 1 or port 2 | Save as active supervisor MAC address, VLAN ID and sequence count.<br>Start announce timeout timer.<br>Transition to ring state in Announce frame and flush unicast MAC address learning table |
| 3 | IDLE_STATE | Link lost on port 1 | Stop forwarding frames on port 1 |
| 4 | IDLE_STATE | Link lost on port 2 | Stop forwarding frames on port 2 |
| 5 | IDLE_STATE | Link restored on port 1 | Start forwarding frames on port 1 |
| 6 | IDLE_STATE | Link restored on port 2 | Start forwarding frames on port 2 |
| 7 | IDLE_STATE | Frame from self received on port 1 or port 2 | Do not forward frame on network and drop it (if capable of doing so) |
| 8 | IDLE_STATE | Locate_Fault or Neighbor_Check request or Neighbor_Check response or Sign_On frame received on port 1 or port 2 | Ignore |
| 9 | FAULT_STATE | Announce frame from active ring supervisor received on port 1 or port 2 with higher sequence count than last Announce frame | Save sequence count.<br>Update VLAN ID (if different).<br>Restart announce timeout timer.<br>If ring state in Announce frame is RING_NORMAL_STATE and links on both ports are active, stop neighbor check timeout timers for both ports, transition to NORMAL_STATE and flush unicast MAC address learning table.<br>Else, stay in FAULT_STATE |
| 10 | FAULT_STATE | Announce timeout timer expired | Stop neighbor check timeout timers for both ports, transition to IDLE_STATE and flush unicast MAC address learning table |
| 11 | FAULT_STATE | Announce frame from different ring supervisor MAC address than active ring supervisor is received on port 1 or port 2 | Restart announce timeout timer and stop neighbor check timeout timers for both ports.<br>Save new active supervisor ,MAC address, VLAN ID and sequence count.<br>If ring state in Announce frame is RING_NORMAL_STATE and links on both ports are active, transition to NORMAL_STATE and flush unicast MAC address learning table.<br>Else, stay in FAULT_STATE |
| 12 | FAULT_STATE | Link lost on port 1 | If link on port 2 is active, send Link_Status frame to active ring supervisor.<br>Else, stop announce timeout timer, stop neighbor check timeout timers for both ports, transition to IDLE_STATE and flush unicast MAC address learning table |
| 13 | FAULT_STATE | Link lost on port 2 | If link on port 1 is active, send Link_Status frame to active ring supervisor.<br>Else, stop announce timeout timer, stop neighbor check timeout timers for both ports, transition to IDLE_STATE and flush unicast MAC address learning table |
| 14 | FAULT_STATE | Link restored on port 1 | Start forwarding frames on port 1 |
| 15 | FAULT_STATE | Link restored on port 2 | Start forwarding frames on port 2 |

| Event No. | Current State | Event | Action(s)[a,b] |
|---|---|---|---|
| 16 | FAULT_STATE | Locate_Fault frame received from active ring supervisor | If links are active on both ports, clear neighbor status for both ports, send Neighbor_Check request frame on both ports and start neighbor check timeout timer for both ports.<br>Else send Link_Status frame to active ring supervisor |
| 17 | FAULT_STATE | Neighbor_Check request frame received on port 1 | Send Neighbor_Check response frame on port 1 |
| 18 | FAULT_STATE | Neighbor_Check request frame received on port 2 | Send Neighbor_Check response frame on port 2 |
| 19 | FAULT_STATE | Neighbor_Check response frame received on port 1 | Stop neighbor check timeout timer for port 1 and save neighbor status for port 1 |
| 20 | FAULT_STATE | Neighbor_Check response frame received on port 2 | Stop neighbor check timeout timer for port 2 and save neighbor status for port 2 |
| 21 | FAULT_STATE | Neighbor check timer timeout on port 1 | If number of retries have not exceeded maximum limit, send Neighbor_Check request on port 1 and start neighbor check timeout timer for port 1.<br>Else, send Neighbor_Status frame to active ring supervisor |
| 22 | FAULT_STATE | Neighbor check timer timeout on port 2 | If number of retries have not exceeded maximum limit, send Neighbor_Check request on port 2 and start neighbor check timeout timer for port 2.<br>Else, send Neighbor_Status frame to active ring supervisor |
| 23 | FAULT_STATE | Sign_On frame received on port 1 or port 2 | Ignore |
| 24 | NORMAL_STATE | Link lost on port 1 | Send Link_Status frame to active ring supervisor.<br>Transition to FAULT_STATE and flush unicast MAC address learning table |
| 25 | NORMAL_STATE | Link lost on port 2 | Send Link_Status frame to active ring supervisor.<br>Transition to FAULT_STATE and flush unicast MAC address learning table |
| 26 | NORMAL_STATE | Announce frame from active ring supervisor received on port 1 or port 2 with higher sequence count than last Announce frame | Save sequence count.<br>Update VLAN ID (if different).<br>Restart announce timeout timer.<br>If ring state in Announce frame is RING_FAULT_STATE, transition to FAULT_STATE and flush unicast MAC address learning table<br>Else, stay in NORMAL_STATE |
| 27 | NORMAL_STATE | Announce timeout timer expired | Transition to IDLE_STATE and flush unicast MAC address learning table |
| 28 | NORMAL_STATE | Announce frame from different ring supervisor MAC address than active ring supervisor is received on port 1 or port 2 | Restart announce timeout timer.<br>Save new active supervisor MAC address, VLAN ID and sequence count.<br>If ring state in Announce frame is RING_FAULT_STATE, transition to FAULT_STATE and flush unicast MAC address learning table.<br>Else, stay in NORMAL_STATE |
| 29 | NORMAL_STATE | Locate_Fault frame received from active ring supervisor | Ignore |

| Event No. | Current State | Event | Action(s)[a,b] |
|---|---|---|---|
| 30 | NORMAL_STATE | Neighbor_Check request or Neighbor_Check response frame received on port 1 or port 2 | Ignore |
| 31 | NORMAL_STATE | Sign_On frame received on port 1 | Add node data to participant list and send frame through port 2 |
| 32 | NORMAL_STATE | Sign_On frame received on port 2 | Add node data to participant list and send frame through port 1 |
| 33 | IDLE_STATE | Flush_Tables frame received | Ignore |
| 34 | NORMAL_STATE | Flush_Tables frame received | Flush unicast and multicast MAC address learning tables.<br>If Learning Update Enable field in Flush_Tables frame is set to TRUE, send Learning_Update frame. |
| 35 | FAULT_STATE | Flush_Tables frame received | Flush unicast and multicast MAC address learning tables.<br>If Learning Update Enable field in Flush_Tables frame is set to TRUE, send Learning_Update frame. |

a Network Topology attribute shall be updated at events 1, 2, 10, 12, 13 and 27 to appropriate value.

b Network Status attribute shall be updated at events 1, 2, 7, 9, 10, 11, 12, 13, 24, 25, 26, 27 and 28 to appropriate value.

### 10.10.3 Ring supervisor

Figure 40 shows the state transition diagram for the ring supervisor.

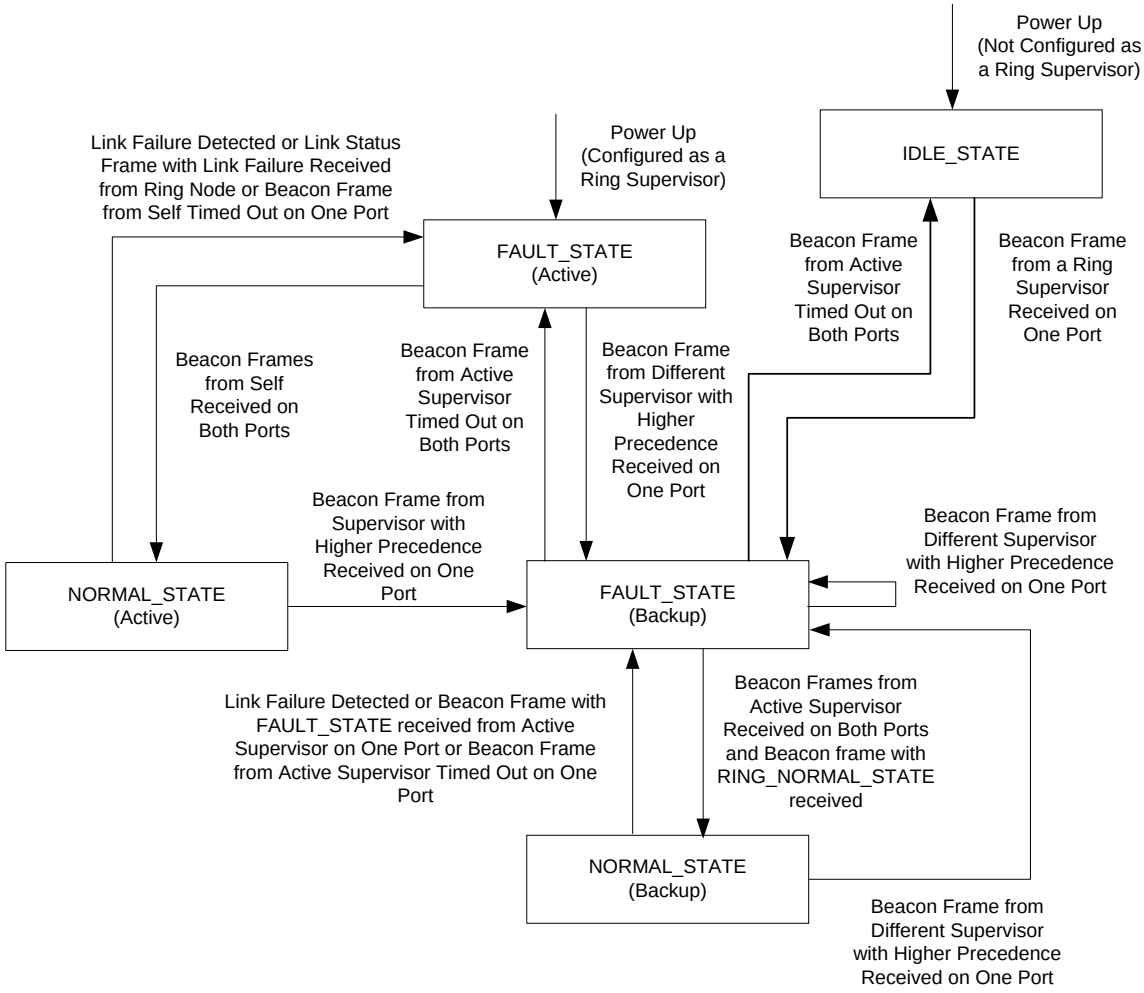



**Figure 40 – State transition diagram for ring supervisor**

The parameter values for the ring supervisor node are specified in Table 158.

**Table 158 – Parameter values for ring supervisor node**

| Parameter | Value |
|---|---|
| Active supervisor precedence | Default 0 or as configured |
| Active supervisor MAC address | As manufactured |
| Ring protocol VLAN ID | Default 0 or as configured |
| Beacon interval duration | Default 400 µs or as configured |
| Beacon timeout duration | Default 1 960 µs or as configured |
| Announce suppression duration | 2 times Beacon Timeout |
| Announce interval duration | 1 s |
| Active supervisor precedence (Backup) | Obtained from Beacon frame |
| Active supervisor MAC address (Backup) | Obtained from Beacon frame |
| Ring protocol VLAN ID (Backup) | Obtained from Beacon frame |
| Beacon timeout duration (Backup) | Obtained from Beacon frame |
| Neighbor check timeout duration | 100 ms |
| Maximum retry limit for Neighbor_Check request frame | 3 (total number of tries) |

| Parameter | Value |
|---|---|
| Switchover quiet period duration during ring supervisor switchover | 1 beacon timeout duration of last active ring supervisor |
| Sign on timeout duration | 60 s |

The following requirements apply to the state-event-action matrix:

- LastBcnRcvPort is a bit string variable used to track port(s) on which last Beacon frame was received, with bit definitions specified in Table 159.
- MAC address 11-22-33-44-55-66 shall be encoded as 0x112233445566 for numerical comparison in case of supervisor precedence tie.
- Nodes not configured as ring supervisor or acting as backup ring supervisor shall statically configure unicast MAC address of current active ring supervisor into unicast MAC learning table to be forwarded only through both ring ports.
- If the attributes of an active supervisor such as beacon interval duration, beacon timeout duration, precedence and VLAN ID are changed, then the active supervisor shall follow special behavior before applying new values. If the active supervisor was in NORMAL_STATE (Active), then it shall stop all timers, leave port 2 in blocked state and stop sending beacon and announce frames for 2 times (current) beacon timeout duration. If the active supervisor was in FAULT_STATE (Active), then it shall stop all timers, leave port 2 in forwarding state and stop sending beacon and announce frames for 2 times (current) beacon timeout duration. At the end of the wait period it shall start from event 1 in Table 160. See also 10.5.2.7 (Changing Ring Parameters).
- If a backup supervisor cannot support the currently active Beacon Interval and/or Beacon Timeout, the backup supervisor shall indicate the condition via the DLR object, and shall not take over as active supervisor in the event of a ring reconfiguration.
- If the precedence attribute of a backup supervisor is reconfigured online to a higher value than current active ring supervisor, then it shall stop all timers and start from event 1 in Table 160 immediately. See also 10.5.2.7 (Changing Ring Parameters).
- If a non-supervisory is node is enabled as a ring supervisor online and the configured precedence value is higher than current active ring supervisor, then it shall stop all timers and start from event 1 in Table 160 immediately. For all other changes, it shall just stay as a backup supervisor in current state.
- The active ring supervisor shall not forward multicast frames with destination address 01:80:C2:00:00:00 (BPDU) from one ring port to other, irrespective of ring state. However, if the active ring supervisor has non-ring ports, it shall forward said multicast frames between those non-ring ports and between only one ring port and those non-ring ports.

**Table 159 – LastBcnRcvPort bit definitions**

| Bit | Definition |
|---|---|
| 0 | Set if a beacon frame from the active supervisor has been received on Port 1 |
| 1 | Set if a beacon frame from the active supervisor has been received on Port 2 |
| 2-N | Not used; set to 0 |

Table 160 specifies the state-event-actions matrix for the ring supervisor node.

**Table 160 – State-event-action matrix for ring supervisor node**

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 1 | None | Power up | Initialize.<br>If not configured as ring supervisor transition to IDLE_STATE and return.<br>Else, set LastBcnRcvPort to 0.<br>Enable forwarding of all frames on both ports .<br>Transition to FAULT_STATE (Active).<br>Send Beacon frame through both ports and start beacon interval timer.<br>Start announce interval timer with announce suppression duration |
| 2 | IDLE_STATE | Beacon frame from a ring supervisor received on port 1 | Save as active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 1.<br>Start beacon timeout timer for port 1.<br>Transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 3 | IDLE_STATE | Beacon frame from a ring supervisor received on port 2 | Save as active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 2.<br>Start beacon timeout timer for port 2.<br>Transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 4 | IDLE_STATE | Link lost on port 1 | Stop forwarding frames on port 1 |
| 5 | IDLE_STATE | Link lost on port 2 | Stop forwarding frames on port 2 |
| 6 | IDLE_STATE | Link restored on port 1 | Start forwarding frames on port 1 |
| 7 | IDLE_STATE | Link restored on port 2 | Start forwarding frames on port 2 |
| 8 | IDLE_STATE | Frame from self received on port 1 or port 2 | Do not forward frame on network and drop it |
| 9 | IDLE_STATE | Locate_Fault or Neighbor_Check request or Neighbor_Check response or Sign_On frame or Link_Status frame or Neighbor_Status frame received on port 1 or port 2 or Verify_Fault_Location service request received | Ignore |
| 10 | FAULT_STATE (Active) | Beacon interval timer expired | Send Beacon frame through both ports and start beacon interval timer |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 11 | FAULT_STATE<br>(Active) | Beacon frame from self received on port 1 | Set LastBcnRcvPort to (LastBcnRcvPort \| 0x1).<br>Restart beacon timeout timer for port 1.<br>If LastBcnRcvPort is not equal to 3, return.<br>Else, stop neighbor check timeout timers for both ports.<br>Block all traffic on port 2 except for special ring protocol frames.<br>Transition to NORMAL_STATE (Active) and flush unicast MAC address learning table.<br>Send Announce frame on port 1 and restart announce interval timer with announce interval duration.<br>Send Beacon frame on both ports and restart beacon interval timer.<br>Send Sign_On frame on port 1 and start sign on timeout timer |
| 12 | FAULT_STATE<br>(Active) | Beacon frame from self received on port 2 | Set LastBcnRcvPort to (LastBcnRcvPort \| 0x2).<br>Restart beacon timeout timer for port 2.<br>If LastBcnRcvPort is not equal to 3, return.<br>Else, stop neighbor check timeout timers for both ports.<br>Block all traffic on port 2 except for special ring protocol frames.<br>Transition to NORMAL_STATE (Active) and flush unicast MAC address learning table.<br>Send Announce frame on port 1 and restart announce interval timer with announce interval duration.<br>Send Beacon frame on both ports and restart beacon interval timer.<br>Send Sign_On frame on port 1 and start sign on timeout timer |
| 13 | FAULT_STATE<br>(Active) | Beacon frame from different supervisor MAC address than self received on port 1 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than self, drop new Beacon frame and return.<br>Else, stop beacon interval timer, stop beacon timeout timers for both ports, stop neighbor check timeout timers for both ports and stop announce interval timer.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout.<br>Set LastBcnRcvPort to 1.<br>Start beacon timeout timer for port 1.<br>Transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 14 | FAULT_STATE (Active) | Beacon frame from different supervisor MAC address than self received on port 2 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than self, drop new Beacon frame and return.<br>Else, stop beacon interval timer, stop beacon timeout timers for both ports, stop neighbor check timeout timers for both ports and stop announce interval timer.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout.<br>Set LastBcnRcvPort to 2.<br>Start beacon timeout timer for port 2.<br>Transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 15 | FAULT_STATE (Active) | Beacon timeout timer expired for port 1 | Set LastBcnRcvPort to (LastBcnRcvPort & ~0x1) |
| 16 | FAULT_STATE (Active) | Beacon timeout timer expired for port 2 | Set LastBcnRcvPort to (LastBcnRcvPort & ~0x2) |
| 17 | FAULT_STATE (Active) | Link lost on port 1 | Save self as last active node on port 1.<br>Stop forwarding frames on port 1 |
| 18 | FAULT_STATE (Active) | Link lost on port 2 | Save self as last active node on port 2.<br>Stop forwarding frames on port 2 |
| 19 | FAULT_STATE (Active) | Link restored on port 1 | Start forwarding frames on port 1 |
| 20 | FAULT_STATE (Active) | Link restored on port 2 | Start forwarding frames on port 2 |
| 21 | FAULT_STATE (Active) | Link_Status frame received on port 1 | If Link_Status frame does not report a link failure, drop frame and return.<br>Else, save source node as last active node on port 1 |
| 22 | FAULT_STATE (Active) | Link_Status frame received on port 2 | If Link_Status frame does not report a link failure, drop frame and return.<br>Else, save source node as last active node on port 2 |
| 23 | FAULT_STATE (Active) | Neighbor_Status frame received on port 1 | If Neighbor_Status frame does not report a neighbor node failure, drop frame and return .<br>Else, save source node as last active node on port 1 |
| 24 | FAULT_STATE (Active) | Neighbor_Status frame received on port 2 | If Neighbor_Status frame does not report a neighbor node failure, drop frame and return.<br>Else save source node as last active node on port 2 |
| 25 | FAULT_STATE (Active) | Announce interval timer expired | Send Announce frame through both ports and start announce interval timer with announce interval duration |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 26 | FAULT_STATE<br>(Active) | Verify_Fault_Location service request received | Send Locate_Fault frame on port 1 and port 2.<br>If link is active on port 1, clear neighbor status for port 1, send Neighbor_Check request frame on port 1 and start neighbor check timeout timer for port 1.<br>Else, save self as last active node on port 1.<br>If link is active on port 2, clear neighbor status for port 2, send Neighbor_Check request frame on port 2 and start neighbor check timeout timer for port 2.<br>Else, save self as last active node on port 2 |
| 27 | FAULT_STATE<br>(Active) | Neighbor_Check request frame received on port 1 | Send Neighbor_Check response frame on port 1 |
| 28 | FAULT_STATE<br>(Active) | Neighbor_Check request frame received on port 2 | Send Neighbor_Check response frame on port 2 |
| 29 | FAULT_STATE<br>(Active) | Neighbor_Check response frame received on port 1 | Stop neighbor check timeout timer for port 1 and save neighbor status for port 1 |
| 30 | FAULT_STATE<br>(Active) | Neighbor_Check response frame received on port 2 | Stop neighbor check timeout timer for port 2 and save neighbor status for port 2 |
| 31 | FAULT_STATE<br>(Active) | Neighbor check timer timeout on port 1 | If number of retries have not exceeded maximum limit, send Neighbor_Check request on port 1 and start neighbor check timeout timer for port 1.<br>Else, save self as last active node on port 1 |
| 32 | FAULT_STATE<br>(Active) | Neighbor check timer timeout on port 2 | If number of retries have not exceeded maximum limit, send Neighbor_Check request on port 2 and start neighbor check timeout timer for port 2.<br>Else, save self as last active node on port 2 |
| 33 | FAULT_STATE<br>(Active) | Sign_On frame received on port 1 or port 2 | Ignore |
| 34 | NORMAL_STATE<br>(Active) | Beacon interval timer expired | Send Beacon frame through both ports and start beacon interval timer |
| 35 | NORMAL_STATE<br>(Active) | Beacon frame from self received on port 1 | Restart beacon timeout timer for port 1 |
| 36 | NORMAL_STATE<br>(Active) | Beacon frame from self received on port 2 | Restart beacon timeout timer for port 2 |
| 37 | NORMAL_STATE<br>(Active) | Beacon frame from different supervisor MAC address than self received on port 1 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon interval timer, stop beacon timeout timers for both ports, stop announce interval timer and stop sign on timeout timer.<br>Enable forwarding of all frames on port 2.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 1.<br>Start beacon timeout timer for port 1.<br>Transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 38 | NORMAL_STATE<br>(Active) | Beacon frame from different supervisor MAC address than self received on port 2 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon interval timer, stop beacon timeout timers for both ports, stop announce interval timer and stop sign on timeout timer.<br>Enable forwarding of all frames on port 2.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 2.<br>Start beacon timeout timer for port 2.<br>Transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 39 | NORMAL_STATE<br>(Active) | Beacon timeout timer expired for port 1 | Stop sign on timeout timer.<br>Set LastBcnRcvPort to 2.<br>Enable forwarding of all frames on port 2.<br>Transition to FAULT_STATE (Active) and flush unicast MAC address table.<br>Send Announce frame on both ports and restart announce interval timer with announce interval duration.<br>Send Beacon frame and restart beacon interval timer.<br>Send Locate_Fault frame.<br>Clear neighbor status for port 1 and 2, send Neighbor_Check_Request for port 1 and 2, and start neighbor check timeout timer for port 1 and 2. |
| 40 | NORMAL_STATE<br>(Active) | Beacon timeout timer expired for port 2 | Stop sign on timeout timer.<br>Set LastBcnRcvPort to 1.<br>Enable forwarding of all frames on port 2.<br>Transition to FAULT_STATE (Active) and flush unicast MAC address table.<br>Send Announce frame on both ports and restart announce interval timer with announce interval duration .<br>Send Beacon frame and restart beacon interval timer.<br>Send Locate_Fault frame.<br>Clear neighbor status for port 1 and 2, send Neighbor_Check_Request for port 1 and 2, and start neighbor check timeout timer for port 1 and 2 |
| 41 | NORMAL_STATE<br>(Active) | Link lost on port 1 | Save self as last active node on port 1.<br>Stop sign on timeout timer.<br>Set LastBcnRcvPort to 2.<br>Enable forwarding of all frames on port 2.<br>Transition to FAULT_STATE (Active) and flush unicast MAC address table.<br>Send Announce frame on both ports and restart announce interval timer with announce interval duration.<br>Send Beacon frame and restart beacon interval timer |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 42 | NORMAL_STATE<br>(Active) | Link lost on port 2 | Save self as last active node on port 2.<br>Stop sign on timeout timer.<br>Set LastBcnRcvPort to 1.<br>Enable forwarding of all frames on port 2.<br>Transition to FAULT_STATE (Active) and flush unicast MAC address table.<br>Send Announce frame on both ports and restart announce interval timer with announce interval duration.<br>Send Beacon frame and restart beacon interval timer |
| 43 | NORMAL_STATE<br>(Active) | Link_Status frame received on port 1 | If Link_Status frame does not report a link failure, drop frame and return.<br>Else, save source node as last active node on port 1.<br>Stop sign on timeout timer.<br>Enable forwarding of all frames on port 2.<br>Transition to FAULT_STATE (Active) and flush unicast MAC address table.<br>Send Announce frame on both ports and restart announce interval timer with announce interval duration.<br>Send Beacon frame and restart beacon interval timer |
| 44 | NORMAL_STATE<br>(Active) | Link_Status frame received on port 2 | If Link_Status frame does not report a link failure, drop frame and return.<br>Else, save source node as last active node on port 2.<br>Stop sign on timeout timer.<br>Enable forwarding of all frames on port 2.<br>Transition to FAULT_STATE (Active) and flush unicast MAC address table.<br>Send Announce frame on both ports and restart announce interval timer with announce interval duration.<br>Send Beacon frame and restart beacon interval timer |
| 45 | NORMAL_STATE<br>(Active) | Neighbor status frame received on port 1 or port 2 | Ignore |
| 46 | NORMAL_STATE<br>(Active) | Announce interval timer expired | Send Announce frame through port 1 and start announce interval timer with announce interval duration |
| 47 | NORMAL_STATE<br>(Active) | Verify_Fault_Location service request received | Ignore |
| 48 | NORMAL_STATE<br>(Active) | Neighbor_Check request or Neighbor_Check response frame received on port 1 or port 2 | Ignore |
| 49 | NORMAL_STATE<br>(Active) | Sign_On frame received on port 1 or port 2 | If first entry in participant list is not self, drop frame and return.<br>Else, save ring participants count and participant list from frame and stop sign on timeout timer |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 50 | NORMAL_STATE (Active) | Sign on timeout timer expired | Send Sign_On frame through port 1 and start sign on timeout timer |
| 51 | FAULT_STATE (Backup) | Beacon frame from active ring supervisor received on port 1 and LastBcnRcvPort is 1 | Restart port 1 beacon timeout timer |
| 52 | FAULT_STATE (Backup) | Beacon frame from active ring supervisor received on port 2 and LastBcnRcvPort is 2 | Restart port 2 beacon timeout timer |
| 53 | FAULT_STATE (Backup) | Beacon timeout timer expired for port 1 and LastBcnRcvPort is 1 | If not configured as a ring supervisor, stop neighbor check timeout timers for both ports, transition to IDLE_STATE, flush unicast MAC address learning table and return.<br>Else, stop neighbor check timeout timers for both ports.<br>Wait for switchover quiet period duration.<br>Set LastBcnRcvPort to 0.<br>Enable forwarding of all frames on both ports .<br>Transition to FAULT_STATE (Active).<br>Send Beacon frame through both ports and start beacon interval timer.<br>Start announce interval timer with announce suppression duration |
| 54 | FAULT_STATE (Backup) | Beacon timeout timer expired for port 2 and LastBcnRcvPort is 2 | If not configured as a ring supervisor, stop neighbor check timeout timers for both ports, transition to IDLE_STATE, flush unicast MAC address learning table and return.<br>Else, stop neighbor check timeout timers for both ports.<br>Wait for switchover quiet period duration.<br>Set LastBcnRcvPort to 0.<br>Enable forwarding of all frames on both ports .<br>Transition to FAULT_STATE (Active).<br>Send Beacon frame through both ports and start beacon interval timer.<br>Start announce interval timer with announce suppression duration |
| 55 | FAULT_STATE (Backup) | Beacon frame from active ring supervisor received on port 2 and LastBcnRcvPort is 1 | Set LastBcnRcvPort to 3.<br>Start/reset port 2 beacon timeout timer.<br>If the ring state of beacon frame is not RING_NORMAL_STATE, return.<br>Else, stop neighbor check timeout timers for both ports.<br>Transition to NORMAL_STATE (Backup) and flush unicast MAC address learning table |
| 56 | FAULT_STATE (Backup) | Beacon frame from active ring supervisor received on port 1 and LastBcnRcvPort is 2 | Set LastBcnRcvPort to 3.<br>Start/reset port 1 beacon timeout timer.<br>If the ring state of beacon frame is not RING_NORMAL_STATE, return.<br>Else, stop neighbor check timeout timers for both ports.<br>Transition to NORMAL_STATE (Backup) and flush unicast MAC address learning table |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 57 | FAULT_STATE<br>(Backup) | Beacon frame from different ring supervisor MAC address than active ring supervisor is received on port 1 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon timeout timers for both ports.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 1.<br>Start beacon timeout timer for port 1.<br>Stay in FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 58 | FAULT_STATE<br>(Backup) | Beacon frame from different ring supervisor MAC address than active ring supervisor is received on port 2 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon timeout timers for both ports.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 2.<br>Start beacon timeout timer for port 2.<br>Stay in FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 59 | FAULT_STATE<br>(Backup) | Link lost on port 1 and LastBcnRcvPort is 1 | If not configured as a ring supervisor, stop neighbor check timeout timers for both ports, stop beacon timeout timers for both ports, transition to IDLE_STATE, flush unicast MAC address learning table and return.<br>Else, stop neighbor check timeout timers for both ports and stop beacon timeout timers for both ports.<br>Wait for switchover quiet period duration.<br>set LastBcnRcvPort to 0.<br>Enable forwarding of all frames on both ports .<br>Transition to FAULT_STATE (Active).<br>Send Beacon frame through both ports and start beacon interval timer.<br>Start announce interval timer with announce suppression duration |
| 60 | FAULT_STATE<br>(Backup) | Link lost on port 1 and LastBcnRcvPort is 2 | Send Link Status frame to active ring supervisor |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 61 | FAULT_STATE (Backup) | Link lost on port 2 and LastBcnRcvPort is 2 | If not configured as a ring supervisor, stop neighbor check timeout timers for both ports, stop beacon timeout timers for both ports, transition to IDLE_STATE, flush unicast MAC address learning table and return.<br>Else, stop neighbor check timeout timers for both ports and stop beacon timeout timers for both ports.<br>Wait for switchover quiet period duration.<br>set LastBcnRcvPort to 0.<br>Enable forwarding of all frames on both ports .<br>Transition to FAULT_STATE (Active).<br>Send Beacon frame through both ports and start beacon interval timer.<br>Start announce interval timer with announce suppression duration |
| 62 | FAULT_STATE (Backup) | Link lost on port 2 and LastBcnRcvPort is 1 | Send Link Status frame to active ring supervisor |
| 63 | FAULT_STATE (Backup) | Link restored on port 1 | Start forwarding frames on port 1 |
| 64 | FAULT_STATE (Backup) | Link restored on port 2 | Start forwarding frames on port 2 |
| 65 | FAULT_STATE (Backup) | Locate_Fault frame received from active ring supervisor on port 1 or port 2 | If link is not active on port 1 or port 2 send Link_Status frame to active ring supervisor<br>Else clear neighbor status for port 1 and 2, send Neighbor_Check_Request for port 1 and 2, and start neighbor check timeout timer for port 1 and 2. |
| 66 | FAULT_STATE (Backup) | Neighbor_Check request frame received on port 1 | Send Neighbor_Check response frame on port 1 |
| 67 | FAULT_STATE (Backup) | Neighbor_Check request frame received on port 2 | Send Neighbor_Check response frame on port 2 |
| 68 | FAULT_STATE (Backup) | Neighbor_Check response frame received on port 1 | Stop neighbor check timeout timer for port 1 and save neighbor status for port 1 |
| 69 | FAULT_STATE (Backup) | Neighbor_Check response frame received on port 2 | Stop neighbor check timeout timer for port 2 and save neighbor status for port 2 |
| 70 | FAULT_STATE (Backup) | Neighbor check timer timeout on port 1 | If number of retries have not exceeded maximum limit, send Neighbor_Check request on port 1 and start neighbor check timeout timer for port 1.<br>Else, send Neighbor_Status frame to active ring supervisor |
| 71 | FAULT_STATE (Backup) | Neighbor check timer timeout on port 2 | If number of retries have not exceeded maximum limit, send Neighbor_Check request on port 2 and start neighbor check timeout timer for port 2.<br>Else, send Neighbor_Status frame to active ring supervisor |
| 72 | FAULT_STATE (Backup) | Sign_On frame or Link_Status frame or Neighbor_Status frame received on port 1 or port 2 or Verify_Fault_Location service request received | Ignore |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 73 | NORMAL_STATE (Backup) | Link lost on port 1 | Send Link Status frame to active ring supervisor.<br>Stop beacon timeout timer for port 1.<br>Set LastBcnRcvPort to 2, transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 74 | NORMAL_STATE (Backup) | Link lost on port 2 | Send Link Status frame to active ring supervisor.<br>Stop beacon timeout timer for port 2.<br>Set LastBcnRcvPort to 2, transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 75 | NORMAL_STATE (Backup) | Beacon frame from active ring supervisor with RING_FAULT_STATE received on port 1 | If the ring state of previous beacon frame received on port 1 was not RING_NORMAL_STATE, return.<br>Else, stop beacon timeout timer for port 2.<br>Set LastBcnRcvPort to 1, transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 76 | NORMAL_STATE (Backup) | Beacon frame from active ring supervisor with RING_FAULT_STATE received on port 2 | If the ring state of previous beacon frame received on port 2 was not RING_NORMAL_STATE, return.<br>Else, stop beacon timeout timer for port 1.<br>Set LastBcnRcvPort to 2, transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 77 | NORMAL_STATE (Backup) | Beacon timeout timer expired for port 1 | Set LastBcnRcvPort to 2, transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 78 | NORMAL_STATE (Backup) | Beacon timeout timer expired for port 2 | Set LastBcnRcvPort to 1, transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 79 | NORMAL_STATE (Backup) | Beacon frame from active ring supervisor received on port 1 | Restart port 1 beacon timeout timer |
| 80 | NORMAL_STATE (Backup) | Beacon frame from active ring supervisor received on port 2 | Restart port 2 beacon timeout timer |
| 81 | NORMAL_STATE (Backup) | Beacon frame from different ring supervisor MAC address than active ring supervisor is received on port 1 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon timeout timers for both ports.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 1.<br>Start beacon timeout timer for port 1.<br>Transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |

| Event No. | Current State | Event | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 82 | NORMAL_STATE (Backup) | Beacon frame from different ring supervisor MAC address than active ring supervisor is received on port 2 | If new supervisor has lower precedence (or numerically lower MAC address in case of tie) than active supervisor, drop new Beacon frame and return.<br>Else, stop beacon timeout timers for both ports.<br>Save new active supervisor precedence, MAC address, VLAN ID and beacon timeout duration.<br>Set LastBcnRcvPort to 2.<br>Start beacon timeout timer for port 2.<br>Transition to FAULT_STATE (Backup) and flush unicast MAC address learning table |
| 83 | NORMAL_STATE (Backup) | Locate_Fault frame or Neighbor_Check request frame or Neighbor_Check response frame or Link_Status frame or Neighbor_Status frame received on port 1 or port 2 or Verify_Fault_Location service request received | Ignore |
| 84 | NORMAL_STATE (Backup) | Sign_On frame received on port 1 | If first entry in participant list is self, drop frame and return.<br>Else, add node data to participant list and send frame on port 2 |
| 85 | NORMAL_STATE (Backup) | Sign_On frame received on port 2 | If first entry in participant list is self, drop frame and return.<br>Else, add node data to participant list and send frame on port 1 |
| 86 | IDLE_STATE | Flush_Tables frame received | Ignore |
| 87 | NORMAL_STATE (Active) | Flush_Tables frame received | Flush unicast and multicast MAC address learning tables.<br>If Learning Update Enable field in Flush_Tables frame is set to TRUE, send Learning_Update frame. |
| 88 | FAULT_STATE (Active) | Flush_Tables frame received | Flush unicast and multicast MAC address learning tables.<br>If Learning Update Enable field in Flush_Tables frame is set to TRUE, send Learning_Update frame. |
| 89 | NORMAL_STATE (Backup) | Flush_Tables frame received | Flush unicast and multicast MAC address learning tables.<br>If Learning Update Enable field in Flush_Tables frame is set to TRUE, send Learning_Update frame. |
| 90 | FAULT_STATE (Backup) | Flush_Tables frame received | Flush unicast and multicast MAC address learning tables.<br>If Learning Update Enable field in Flush_Tables frame is set to TRUE, send Learning_Update frame. |

| | |
|---|---|
| a | Network Topology attribute shall be updated at events 1, 2, 3, 53, 54, 59 and 61 to appropriate value. |
| b | Network Status attribute shall be updated at events 1, 2, 3, 8, 11, 12, 13, 14, 37, 38, 39, 40, 41, 42, 43, 44, 53, 54, 55, 56, 57, 58, 59, 61, 73, 74, 75, 76, 77, 78, 81 and 82 to appropriate value. |
| c | Ring Supervisor Status attribute shall be updated at events 1, 13, 14, 37, 38, 53, 54, 59 and 61 to appropriate value. |
| d | Ring Faults Count attribute shall be updated at events 1, 13, 14, 37, 38, 39, 40, 41, 42, 43, 44, 53, 54, 59, 61 to appropriate value. |
| e | Last Active Node 1 attribute shall be updated at events 1, 11, 12, 13, 14, 17, 21, 23, 26, 31, 41 and 43 to appropriate value. |
| f | Last Active Node 2 attribute shall be updated at events 1, 11, 12, 13, 14, 18, 22, 24, 26, 32, 42 and 44 to appropriate value. |
| g | Ring Protocol Participants Count attribute shall be updated at events 1, 13, 14, 37, 38 and 49 to appropriate value. |
| h | Ring Protocol Participants List attribute shall be updated at events 1, 13, 14, 37, 38 and 49 to appropriate value. |
| i | If in events 11 and 12, Beacon frames are received continuously for more than 3 consecutive beacon timeout durations on the same port without the other port receiving any Beacon frames, then that constitutes a partial network fault condition. When such a condition is detected, the active ring supervisor shall block all traffic on port 2 resulting in network segmentation and update network status attribute. This condition requires user intervention to resolve. The location of fault can be identified through Verify_Fault_Location service. |
| j | If in events 39, 40, 41, 42, 43, 44 the supervisor detects that 5 faults have occurred in a 30 s period, the supervisor shall remain in FAULT_STATE and block all traffic on port 2. Such a condition shall be explicitly cleared by the user via the Clear_Rapid_Faults service of the DLR object. |

### 10.10.4 Redundant gateway

Figure 41 shows the state transition diagram for a redundant gateway.

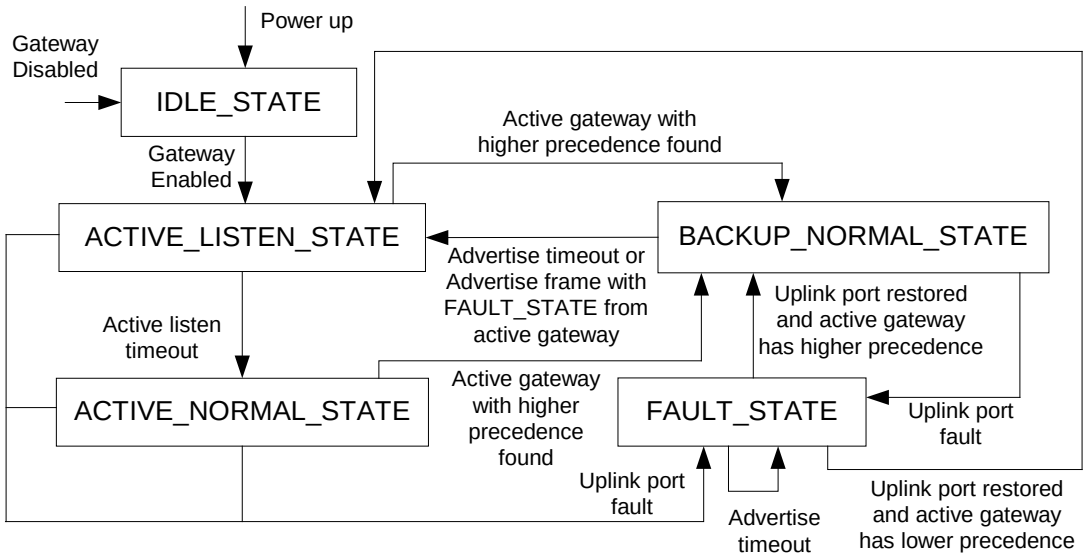



**Figure 41 – State transition diagram for redundant gateway**

Table 161 specifies the parameter values for a redundant gateway node.

**Table 161 – Parameter values for redundant gateway node**

| Parameter | Value |
|---|---|
| Active gateway Precedence | Default 0 or as configured |
| Active gateway MAC address | As manufactured |
| Advertise Interval duration | Default 2 000 µs or as configured |
| Advertise Timeout duration | Default 5 000 µs or as configured |
| Learning Update Enable | Default TRUE or as configured |
| Active Listen Timeout duration | 1 times Advertise Timeout duration |
| Active gateway Precedence (Backup) | Obtained from Advertise frame |
| Active gateway MAC address (Backup) | Obtained from Advertise frame |
| Learning Update Enable (Backup) | Obtained from Advertise frame |
| Advertise Interval duration (Backup) | Obtained from Advertise frame |
| Advertise Timeout duration (Backup) | Obtained from Advertise frame |

The following requirements apply to the state-event-action matrix.

- MAC address 11-22-33-44-55-66 shall be encoded as 0x112233445566 for numerical comparison in case of gateway Precedence tie.
- If the attributes of an active gateway such as Advertise Interval duration, Advertise Timeout duration, Precedence and Learning Update Enable are changed, the active gateway shall follow special behavior before applying new values. It should block traffic forwarding between DLR ports and uplink ports, stop all timers and stop sending Advertise frames for 1,5 times old Advertise Timeout duration. At the end of wait period it should start from event 1 in table below.
- If a backup gateway cannot support the currently active Advertise Interval and/or Advertise Timeout, the backup gateway shall indicate the condition via the DLR Object, and shall not take over as active gateway in the event of a gateway switchover.
- If the Precedence attribute of a backup gateway is reconfigured online to a higher value than current active gateway, then it should stop all timers and start from event 1 in table below immediately.
- Unlike other DLR frames, an active gateway device shall forward the Learning_Update frames to uplink ports when traffic forwarding is enabled between DLR ports and uplink ports.
- An active gateway device in ACTIVE_NORMAL_STATE shall declare partial network fault, if it receives Advertise frame with ACTIVE_NORMAL_STATE as gateway state from another active gateway device with lower Precedence (or numerically lower MAC address in case of a tie). Upon declaring partial network fault, it shall transition to FAULT_STATE, it shall stop all timers, shall stop sending Advertise frames and shall block traffic forwarding between DLR ports and uplink ports. It shall indicate the condition through the Redundant Gateway Status attribute in DLR object. It shall set the Active Gateway Address and Active Gateway Precedence attributes to that of the other active gateway with lower precedence. When the user clears the condition through Clear_Gateway_Partial_Fault service, the gateway shall clear the condition and start from event 1 in table below.
- When a gateway switchover occurs, the new active gateway device shall send the topology change notification message on its uplink ports that is appropriate for the protocol currently enabled on its uplink ports. This requirement does not apply if no protocol is currently enabled on its uplink ports.

Table 162 specifies the state-event-action matrix for a redundant gateway node.

**Table 162 – State-event-action matrix for redundant gateway node**

| Event No. | Current state | Event | Action(s)[a] |
|---|---|---|---|
| 1 | None | Power up | Initialize<br>Block traffic forwarding between DLR ports and uplink ports<br>Transition to IDLE_STATE |
| 2 | IDLE_STATE | Redundant gateway enabled | Start active listen timer for Active Listen Timeout duration<br>Send Advertise frame with ACTIVE_LISTEN_STATE as gateway state<br>Start advertise interval timer with Advertise Interval duration<br>Transition to ACTIVE_LISTEN_STATE |
| 3 | ACTIVE_LISTEN_STATE | Advertise frame from another active gateway with higher Precedence (or numerically higher MAC address in case of a tie) is received and gateway state field in frame is not set to FAULT_STATE | Save active gateway MAC address, Precedence, Advertise Interval and Timeout and Learning Update Enable fields<br>Stop advertise interval timer<br>Stop active listen timer<br>Start advertise timeout timer with Advertise Timeout duration<br>Transition to BACKUP_NORMAL_STATE |
| 4 | ACTIVE_LISTEN_STATE | Advertise interval timer expired | Send Advertise frame<br>Restart advertise interval timer with Advertise Interval duration |
| 5 | ACTIVE_LISTEN_STATE | Active listen timer expired | Send Advertise frame with ACTIVE_NORMAL_STATE as gateway state<br>Restart advertise interval timer with Advertise Interval duration<br>Enable traffic forwarding between DLR ports and uplink ports<br>Send Flush_Tables frame to all DLR nodes<br>Send Learning_Update frame if Learning Update Enable attribute of DLR Object is TRUE<br>Transition to ACTIVE_NORMAL_STATE |
| 6 | ACTIVE_LISTEN_STATE | Uplink connection or higher level failure | Send Advertise frame with FAULT_STATE as gateway state<br>Stop advertise interval timer<br>Stop active listen timer<br>Start advertise timeout timer with Advertise Timeout duration<br>Transition to FAULT_STATE |

| Event No. | Current state | Event | Action(s)[a] |
|---|---|---|---|
| 7 | ACTIVE_NORMAL_STATE | Advertise frame from another active gateway with higher Precedence (or numerically higher MAC address in case of a tie) is received and gateway state field in frame is not set to FAULT_STATE | Save active gateway MAC address, Precedence, Advertise Interval and Timeout and Learning Update Enable fields<br>Stop advertise interval timer<br>Start advertise timeout timer with Advertise Timeout duration<br>Block traffic forwarding between DLR ports and uplink ports<br>Transition to BACKUP_NORMAL_STATE |
| 8 | ACTIVE_NORMAL_STATE | Advertise interval timer expired | Send Advertise frame<br>Restart advertise interval timer with Advertise Interval duration |
| 9 | ACTIVE_NORMAL_STATE | Uplink connection or higher level failure | Send Advertise frame with FAULT_STATE as gateway state<br>Stop advertise interval timer<br>Start advertise timeout timer with Advertise Timeout duration<br>Block traffic forwarding between DLR ports and uplink ports<br>Transition to FAULT_STATE |
| 10 | FAULT_STATE | Uplink connection restored and active gateway has higher Precedence (or numerically higher MAC address in case of a tie) | Start advertise timeout timer with Advertise Timeout duration<br>Transition to BACKUP_NORMAL_STATE |
| 11 | FAULT_STATE | Uplink connection restored and active gateway has lower Precedence (or numerically lower MAC address in case of a tie) | Start active listen timer for Active Listen Timeout duration<br>Send Advertise frame with ACTIVE_LISTEN_STATE as gateway state<br>Start advertise interval timer with Advertise Interval duration<br>Transition to ACTIVE_LISTEN_STATE |
| 12 | FAULT_STATE | Advertise frame from active gateway received | Restart advertise timeout timer with Advertise Timeout duration |
| 13 | FAULT_STATE | Advertise timer timeout | Save active gateway MAC address and Precedence as all zeros |
| 14 | FAULT_STATE | Advertise frame from another active gateway with higher Precedence than current active gateway (or numerically higher MAC address in case of a tie) is received and gateway state field in frame is not set to FAULT_STATE | Save active gateway MAC address, Precedence, Advertise Interval and Timeout and Learning Update Enable fields |
| 15 | BACKUP_NORMAL_STATE | Uplink connection or higher level failure | Transition to FAULT_STATE |
| 16 | BACKUP_NORMAL_STATE | Advertise frame from active gateway received | Restart advertise timeout timer with Advertise Timeout duration |

| Event No. | Current state | Event | Action(s)[a] |
|---|---|---|---|
| 17 | BACKUP_NORMAL_STATE | Advertise timer timeout | Send Advertise frame with ACTIVE_LISTEN_STATE as gateway state<br>Start active listen timer with Active Listen Timeout duration<br>Start advertise interval timer with Advertise Interval duration<br>Transition to ACTIVE_LISTEN_STATE |
| 18 | BACKUP_NORMAL_STATE | Advertise frame from another active gateway with higher Precedence than current active gateway (or numerically higher MAC address in case of a tie) is received and gateway state field in frame is not set to FAULT_STATE | Save active gateway MAC address, Precedence, Advertise Interval and Timeout and Learning Update Enable fields<br>Restart advertise timeout timer with Advertise Timeout duration |

[a] Redundant gateway status attribute shall be updated on events 2, 3, 6, 7, 9, 10, 11, 15 and 17 to appropriate value.

## 10.11 Performance analysis

### 10.11.1 General

The example performance calculations below use the key parameters/assumptions shown in Table 163.

**Table 163 – Parameters/assumptions for example performance calculations**

| Variable | Description | Example value |
|---|---|---|
| NumNodes | Number of nodes in ring network | 50 |
| BcnFrmDly | Delay due to a beacon frame in store and forward switching | 7 µs (64 octet frame with on wire overhead) |
| AvgEipFrmDly | Delay due to average CP 2/2 frame | 12 µs (128 octet frame with on wire overhead) |
| MaxFrmDly | Delay due to maximum size Ethernet frame | 124 µs (1 522 octet frame with on wire overhead) |
| MaxDlyNodePrcnt | Percentage of number of nodes in ring on which beacon will be delayed by maximum size Ethernet frame | 10 % |
| IntSwtchDly | Internal switching delay on every node | 5 µs |
| WirePrpgtDly | Signal propagation delay on 100 m of copper media | 1 µs |
| NodeProcDly | Processing delay on nodes for responding to ring frames and events | 25 µs |
| BcnIntrvl | Beacon interval, shall be less than half of the fastest connection RPI | 400 µs |

In order to provide predictable performance for network fault detection and reconfiguration all nodes directly on ring shall dedicate highest priority queue to ring protocol frames and shall implement strict priority scheduling for highest priority queue. With such a configuration, a ring protocol frame will encounter at most one lower priority frame delay on each node. For performance analysis, assume that all links on network operate at 100 Mbit/s speed and in full duplex mode.

Ring Beacon frames are 64 octets long including FCS and have an on wire overhead of 20 octets of which 8 octets are for preamble and start of frame delimiter pattern and 12 octets are for inter frame gap. Ring Beacon frames with 20 octet on wire overhead take approximately BcnFrmDly = 7 µs on wire.

It is assumed that ring Beacon frames will be delayed on most nodes by a lower priority frame with an average size of 128 octets including FCS. In a network with mostly CP 2/2 traffic, on some nodes ring Beacon frames may not be delayed at all, in some other nodes they may be delayed by 256 octet frames and in some others it may be delayed by frames between these two extremes, for an average of 128 octets. A 128 octet frame with 20 octet on wire overhead takes approximately AvgEipFrmDly = 12 µs on wire.

It is assumed that the nodes use store and forward switching architecture and that each node has an average internal switching overhead delay of IntSwtchDly = 5 µs. Assume propagation delay for copper media of 100 meters to be WirePrpgtDly = 1 µs. The total typical delay per node for ring Beacon frames is therefore:

TypclDlyPerNode = BcnFrmDly + AvgEipFrmDly + IntSwtchDly + WirePrpgtDly

TypclDlyPerNode = 7 + 12 + 5 + 1 = 25 µs

It is assumed that the ring Beacon frames would also be delayed on MaxDlyNodePrcnt = 10 % of nodes by maximum sized Ethernet frames of 1 522 octets each or some combination of large frames on more than 10 % of nodes that is equal to 10 % of nodes with maximum sized frames. Such frames may be present on network for any reason including for example configuration, HMI, web. A 1 522 octet frame with 20 octet on wire overhead takes approximately MaxFrmDly = 124 µs on wire. The total maximum delay per node for ring Beacon frames on these nodes is therefore:

MaxDlyPerNode = BcnFrmDly + MaxFrmDly + IntSwtchDly + WirePrpgtDly

MaxDlyPerNode = 7 + 124 + 5 + 1 = 137 µs

For a ring network comprised of NumNodes = 50 nodes, total maximum round trip time for Beacon frames is therefore:

MaxRndTripTime = (NumNodes × (1 – MaxDlyNodePrcnt) × TypclDlyPerNode)
+ (NumNodes × MaxDlyNodePrcnt × MaxDlyPerNode)

MaxRndTripTime = (50 × (1 – 0,1) × 25) + (50 × 0,1 × 137) = 1 810 µs

For same network, minimum delay per node is when Beacon frame is not delayed by any other frame and therefore:

MinDlyPerNode = BcnFrmDly + IntSwtchDly + PrpgtDly

MinDlyPerNode = 7 + 5 + 1 = 13 µs

For a ring network comprised of NumNodes = 50 nodes, total minimum round trip time is therefore:

MinRndTripTime = NumNodes × MinDlyPerNode

MinRndTripTime = 13 × 50 = 650 µs

In general, lower beacon interval provides faster ring recovery performance. Beacon interval shall be less than half of the fastest connection RPI in the network to prevent connection timeouts. Assume a beacon interval of BcnIntrvl = 400 µs which constitutes 1,75 % of network link transmit time and is suitable for high performance CIP motion connections with 1 ms RPI and also works for slower I/O connections.

For choosing beacon timeout, consider a first Beacon frame transmitted at time $Tx_1$ facing minimum round trip time delay and arriving at active supervisor ports at time:

$Rx_1 = Tx_1$ + MinRndTripTime $= Tx_1 + 650$ µs

Assume that a second Beacon frame transmitted at time:

$Tx_2 = Tx_1$ + BcnIntrvl $= Tx_1 + 400$ µs

is lost on route due to frame corruption.

A third Beacon frame transmitted at time $Tx_3 = Tx_2 + BcnIntrvl = Tx_2 + 400$ µs facing maximum roundtrip delay will arrive at active supervisor ports at time:

$$Rx_3 = Tx_3 + MaxRndTripTime = Tx_3 + 1\,810 \text{ µs}$$

The maximum arrival delay of third Beacon frame on active supervisor ports after first Beacon frame is therefore equal to:

$$Rx_3 - Rx_1 = (Tx_3 + 1\,810) - (Tx_1 + 650) = (Tx_1 + 400 + 400 + 1\,810) - (Tx_1 + 650) = 1\,960 \text{ µs}$$

Hence the beacon timeout duration shall be equal to:

$$BcnTimeout = 1\,960 \text{ µs}$$

Assume end node processing delay of NodeProcDly = 25 µs for responding to ring frames or events.

For the network described, following will be the worst case performance for ring nodes that rely on Beacon frame mechanism:

a) Faults that are detectable in physical layer. This is the most common type of faults. The worst case scenario happens when the fault occurs half way across ring network from active ring supervisor. It will take half round trip time for Link Status frames to reach from neighboring nodes of fault to active ring supervisor. It will take another half round trip time for Beacon frame with FAULT_STATE from active supervisor to reach farthest node or for ring nodes to timeout Beacon frames whichever happens earlier. End node processing delay is involved for three times, once at fault neighbor, once at supervisor and once at farthest node. The total worst case delay is therefore:

$$PhysclLyrFltDlyBcn = (2 \times 0{,}5 \times MaxRndTripTime) + (3 \times NodeProcDly)$$

$$PhysclLyrFltDlyBcn = 1\,810 + (3 \times 25) = 1\,885 \text{ µs}$$

b) Faults that are not detectable in physical layer. This type of faults is relatively rare. The worst case scenario happens when the fault occurs half way across ring network from active ring supervisor. It will take half round trip time for last Beacon frame from near fault location to reach active ring supervisor. It will take another beacon timeout period for active supervisor and ring nodes to timeout Beacon frames. End node processing delay is involved once at all nodes. The total worst case delay is therefore:

$$NonPhysclLyrFltDlyBcn = (0{,}5 \times MaxRndTripTime) + BcnTimeout + NodeProcDly$$

$$NonPhysclLyrFltDlyBcn = (0{,}5 \times 1\,810) + 1\,960 + 25 = 2\,890 \text{ µs}$$

c) Network restoration to normal mode of operation. It is equal to a total of one beacon interval for a beacon to be generated, one maximum round trip time for beacon and another half maximum round trip time for Beacon frame with RING_NORMAL_STATE to reach farthest node. End node processing delay is involved twice, once at ring supervisor and once at all nodes. The total worst case delay is therefore:

$$RingRstrDlyBcn = BcnIntrvl + (1{,}5 \times MaxRndTripTime) + (2 \times NodeProcDly)$$

$$RingRstrDlyBcn = 400 + (1{,}5 \times 1\,810) + (2 \times 25) = 3\,165 \text{ µs}$$

For the network described, following will be the worst case performance for ring nodes that rely on Announce frame mechanism:

a) Faults that are detectable in physical layer. This is the most common type of faults. The worst case scenario happens when the fault occurs half way across ring network from active ring supervisor. It will take half round trip time for Link Status frames to reach from neighboring nodes of fault to active ring supervisor. It will take another half round trip time for Announce frame with FAULT_STATE from active ring supervisor to reach farthest node. End node processing delay is involved for three times, once at fault neighbor, once at supervisor and once at farthest node. The total worst case delay is therefore

PhysclLyrFltDlyAnnc = (2 × 0,5 × MaxRndTripTime) + (3 × NodeProcDly)

PhysclLyrFltDlyAnnc = 1 810 + (3 × 25) = 1 885 µs

b) Faults that are not detectable in physical layer. This type of faults is relatively rare. The worst case scenario happens when the fault occurs half way across ring network from active ring supervisor. It will take half round trip time for last Beacon frame from near fault location to reach active ring supervisor. It will take another beacon timeout period for active supervisor to timeout Beacon frames. It will take another half round trip time for Announce frame with FAULT_STATE from active supervisor to reach farthest node. End node processing delay is involved twice, once at supervisor and once at end node. The total worst case delay is therefore:

NonPhysclLyrFltDlyAnnc = (2 × 0,5 × MaxRndTripTime) + BcnTimeout + (2 × NodeProc Dly)

NonPhysclLyrFltDlyAnnc = 1 810 + 1 960 + (2 × 25) = 3 820 µs

c) Network restoration to normal mode of operation. It is equal to a total of one beacon interval for a beacon to be generated, one maximum round trip time for beacon and another maximum round trip time for Announce frame to reach farthest node. End node processing delay is involved twice, once at supervisor and once at end node. This is the total delay for an Announce based node to flush its unicast MAC table. The ring would have been restored earlier per calculation for Beacon based nodes. The total worst case delay is therefore:

RingRstrDlyAnnc = BcnIntrvl + (2 × MaxRndTripTime) + (2 × NodeProcDly)

RingRstrDlyAnnc = 400 + (2 × 1 810) + (2 × 25) = 4 070 µs

Based on similar calculations, Table 164 provides configuration parameters and worst case performance numbers for different ring network node numbers.

NOTE Though these performance numbers will work for most cases, deviation from assumptions outlined above will require recalculation of numbers based on the procedure described above. For example, if any link in ring is set to operate at 10 Mbit/s speed and/or half duplex mode the numbers need to be adjusted (for 10 Mbit/s, multiply by 10).

**Table 164 – Example ring configuration parameters and performance**

| Number of ring nodes | Beacon interval (µs) | Round trip time[a] (µs) | Beacon timeout (µs) | Physical layer faults recovery delay[a] (µs) | Non-physical layer faults recovery delay for beacon frame based nodes (µs) | Non-physical layer faults recovery delay for announce frame based nodes (µs) | Ring restore delay for beacon frame based nodes (µs) | Ring restore delay for announce frame based nodes (µs) |
|---|---|---|---|---|---|---|---|---|
| 25 | 400 | 905 | 1 380 | 980 | 1 858 | 2335 | 1 808 | 2 260 |
| 50 (nominal network size) | 400 | 1 810 | 1 960 | 1 885 | 2 890 | 3 820 | 3 165 | 4 070 |
| 100 | 400 | 3 620 | 3 120 | 3 695 | 4 955 | 6 790 | 5 880 | 7 690 |
| 150 | 400 | 5 430 | 4 280 | 5 505 | 7 020 | 9 760 | 8 595 | 11 310 |
| 200 | 400 | 7 240 | 5 440 | 7 315 | 9 085 | 12 730 | 11 310 | 14 930 |
| 250 | 400 | 9 050 | 6 600 | 9 125 | 11 150 | 15 700 | 14 025 | 18 550 |

[a] Same for Beacon and Announce frames based nodes.

Important: When non-DLR nodes are present in the ring, recovery and restoration delays are as they are for Announce-based nodes, provided the non-DLR nodes follow the requirements

specified in 10.5. If non-DLR nodes do not follow the requirements, recovery and restoration delays are unpredictable.

When using non-DLR switches in the ring, unicast packet loss may occur, as described in 10.7.3 (Non-DLR Switches).

### 10.11.2 Redundant gateway switchover performance

The example performance calculations below use the key parameters/assumptions shown in Table 165.

**Table 165 – Variables for performance analysis**

| Variable | Description | Example value |
|---|---|---|
| NodeProcDly | Processing delay on nodes for responding to ring frames and events | 25 µs |
| MaxRndTripTime | 1 810 µs for 50 nodes from 10.11.1 | |
| AdvrtsIntrvl | Advertise Interval, should be more than MaxRndTripTime | 2 000 µs |
| AdvrtsTimeout | Advertise Timeout, should be 2,5 times AdvrtsIntrvl | 5 000 µs |
| ActiveListenTimeout | Equal to 1 times AdvrtsTimeout | 5 000 µs |
| LearnUpdtPropDly | Learning update frame propagation delay through DLR to non-DLR switches outside DLR network, should be greater than MaxRndTripTime | 5 000 µs |

Redundant gateway switchover performances for three different cases of failure, assuming Learning Update Enable has been set to TRUE, are as follows.

a) Uplink connection failure detected by active gateway at physical layer: 13,745 ms

This is the most common type of faults. The active gateway will detect physical layer failure and will send an Advertise frame with FAULT_STATE. It may take MaxRndTripTime for the Advertise frame to reach backup gateway device. The backup gateway device will take ActiveListenTimeout to enable traffic forwarding and send Flush_Tables frame. Flush_Tables frame may take MaxRndTripTime to reach farthest DLR node. Farthest DLR node will send Learning_Update frame which may take LearnUpdtPropDly to update non-DLR switches outside DLR network. End node processing delays are involved for 5 times in total. The total worst case delay is therefore:

$$\begin{aligned} \text{GtwyPhysclLyrFltDly} &= (2 \times \text{MaxRndTripTime}) + \text{ActiveListenTimeout} + \\ &\quad \text{LearnUpdtPropDly} + (5 \times \text{NodeProcDly}) \\ &= (2 \times 1\,810) + 5\,000 + 5\,000 + (5 \times 25) \\ &= 13\,745\ \mu\text{s}. \end{aligned}$$

b) Active gateway failure: 18,695 ms

Active gateway might fail just after sending last Advertise frame. It may take MaxRndTripTime to reach backup gateway device. The backup gateway device will wait for AdvrtsTimeout before declaring active gateway as lost. The backup gateway device will further take ActiveListenTimeout to enable traffic forwarding and send Flush_Tables frame. Flush_Tables frame may take MaxRndTripTime to reach farthest DLR node. Farthest DLR node will send Learning_Update frame which may take LearnUpdtPropDly to update non-DLR switches outside DLR network. End node processing delays are involved for 3 times in total. The total worst case delay is therefore:

$$\begin{aligned} \text{GtwyFailDly} &= (2 \times \text{MaxRndTripTime}) + \text{AdvrtsTimeout} + \text{ActiveListenTimeout} + \\ &\quad \text{LearnUpdtPropDly} + (3 \times \text{NodeProcDly}) \\ &= (2 \times 1\,810) + 5\,000 + 5\,000 + 5\,000 + (3 \times 25) \\ &= 18\,695\ \mu\text{s}. \end{aligned}$$

c) Higher layer uplink fault detection: 6,013 745 s

A higher layer uplink failure (e.g. something other than a link failure reported by the PHY) might be detected by active gateway through a protocol (e.g., RSTP) on uplink ports. This case depends on that protocol's high layer fault detection performance. For example with RSTP, the delay to detect a higher layer fault could be as high as 6 s (with typical defaults) to timeout RSTP hello frames. Once detected additional delays are same as GtwyPhysclLyrFltDly, since same set of actions need to take place. The total worst case delay for RSTP is therefore:

GtwyRstpLyrFltDly = RstpFltDly + GtwyPhysclLyrFltDly
= 6 000 000 + 13 745
= 6 013 745 µs.

# Annex A
(normative)

# Indicators and switches

## A.1 Purpose

A common approach to status indicators and switch behavior is recommended to provide a consistent human user interface for all devices conforming to this standard. These indicators, typically LEDs, assist maintenance and diagnosis of problems with the Media, Physical and Data Link Layers.

NOTE Throughout Annex A the term “network” is used to refer to that portion of the Ph-subnetwork and DL-subnetwork that is connected to the device making the observation.

## A.2 Indicators

### A.2.1 General indicator requirements

CPF 2 profiles have different levels of requirements for support of status indicators. However, if a device does support status indicators, they shall follow the specifications described in this annex for the selected CP.

It is strongly recommended to implement these indicators in any device for which no practical constraint prevents it.

NOTE This is because indicators help maintenance personnel to quickly identify a faulty unit or media (e.g. red indicators are typically used to indicate a fault condition).

Two types of status indicators may be provided:

– one module status indicator;
– one or several network status indicators.

These may be sometimes combined.

Additional indicators may be present; however, the naming and symbol conventions of the standard indicators shall not be employed for other indicators.

Safety devices have additional requirements, see IEC 61784-3-2.

### A.2.2 Common indicator requirements

#### A.2.2.1 Applicability of common requirements

The common indicator requirement shall only apply to indicators for which requirements are specified in this standard.

#### A.2.2.2 Visibility of indicators

Indicators shall be visible without removing covers or parts from the equipment. Indicators shall be easily seen in normal lighting. Any related labels and icons shall be visible whether or not the indicator is illuminated.

### A.2.2.3 Indicator flash rate

Unless otherwise indicated, a two state indicator shall have a flash rate of (1,0 ± 0,1) Hz. The indicator shall have a duty cycle of (50 ± 5) %.

NOTE The indicator is in the first state (on) for approximately 0,5 s and in the second state (off) for approximately 0,5 s.

### A.2.2.4 Module status indicator

#### A.2.2.4.1 Description

This single bicolor (green/red) indicator shall provide status of the entire device. It shall indicate whether or not the device is powered, configured, and operating properly.

NOTE A device with more than one communication port would have only one module status indicator.

#### A.2.2.4.2 States

The module status indicator states shall be as specified in Table A.1. The indicator states reflect the device states specified in the Identity object (see IEC 61158-5-2).

**Table A.1 – Module status indicator**

| Indicator state | Device state | Cause |
|---|---|---|
| Steady off | No power | There is no power applied to the device. |
| Steady green | Operational | The device is operating in a normal condition. |
| Flashing green | Standby | The device needs commissioning due to configuration missing, incomplete or incorrect. |
| Flashing red | Recoverable fault | The device has detected a recoverable major fault (see NOTE 1). |
| Steady red | Unrecoverable fault | The device has an unrecoverable major fault; may need replacing. |
| Flashing red / green | Device self testing | The device is executing its power up self tests. |
| NOTE 1 Indicator flash rates are given in A.2.2.3.<br>NOTE 2 An incorrect or inconsistent configuration would be considered a recoverable fault. | | |

Any indicator colors/states not defined in Table A.1 are reserved and shall not be used.

Safety devices have additional requirements, see IEC 61784-3-2.

### A.2.2.5 Time Sync status indication

A device shall provide indication of clock synchronization status. Table A.2 shows the suggested synchronization status for various indicators.

A device is considered synchronized when its local clock is synchronized with the master reference clock. However, an application may impose more stringent synchronization requirements.

**Table A.2 – Time Sync status indication**

| Visual status indicator | Suggested operation |
|---|---|
| Module status LED | Standby (flashing green) = not synchronized<br>Operational (Solid green) = synchronized + other device requirements |
| Multi-character alpha-numeric display | A device waiting for synchronization should display “Synching”<br>A device that is a PTP Master should display “Time Master”<br>A device that is a PTP Grandmaster should display “Time GM” |
| Web or Property Page (suggested) | Identity of the grandmaster, parent and local clock<br>Grandmaster and local clock quality<br>Synchronization status<br>Current system time<br>Current offset to master<br>Current port status |
| NOTE Module status indicator is specified in A.2.3, A.2.4 or A.2.5. | |

### A.2.3 Fieldbus specific indicator requirements (1)

#### A.2.3.1 General requirements (1)

Two categories of status indicators shall be provided for each CP 2/1 device:

– one module status indicator;
– two network status indicators.

#### A.2.3.2 Indicators at power up

At power up, the Type 2 indicators shall turn on to a fault (red) or reset (off) state. Setting the Type 2 indicators to fault at power up is preferred so that a fault state is reflected in the event that the module cannot successfully complete any of its power-up tests. During power up (prior to the start of steady state application execution), the Type 2 indicators shall be exercised as follows to allow for a visual inspection of their proper operation.

– For the Module Status (MS) indicator:
  - turn MS indicator green, all other Type 2 indicators off;
  - leave MS indicator on green for approximately 0,25 s;
  - turn MS indicator on red for approximately 0,25 s;
  - turn MS indicator on green.
– For the two Network Status (NS) indicators:
  - turn each indicator green for approximately 0,25 s;
  - turn each indicator red for approximately 0,25 s;
  - turn each indicator off.
– If additional Type 2 indicators are present:
  - turn each indicator green for approximately 0,25 s;
  - turn each indicator red for approximately 0,25 s;
  - turn each indicator off.

The visual inspection tests for the Network Status and any additional Type 2 indicators may be run sequentially or in parallel. After completion of the visual inspection tests, the Type 2 indicator(s) shall behave as specified in this standard.

#### A.2.3.3 Module status indicator

Behavior of the module status indicator shall be as specified in A.2.2.4.

The module status indicator shall be labeled with one of the following:

"MS";

"OK";

"status";

"mod status";

"module status".

#### A.2.3.4 Network status indicators

##### A.2.3.4.1 Description

The indication of network status shall require two bicolor (red/green) indicators. If the device has redundant PhLs (channels), each of the indicators shall be associated with one of the channels. If the device does not support channel redundancy, one of the indicators shall be associated with the single channel; however, two indicators shall still be required.

NOTE These indicators signify a number of conditions within the PhL; and when viewed together, they can reflect the state of the Data Link Layer.

The network status indicators do not usually reflect any errors received on the NAP; however, nodes which have only a NAP (no other PhL variants) may use the network status indicators to represent the status of the NAP. If this optional behavior is implemented, the state of both the indicators shall reflect the state of NAP communication, instead of the A and B channels.

##### A.2.3.4.2 States

###### A.2.3.4.2.1 Definitions

The following definitions shall be used to describe the indicator behavior:

**steady** – indicator shall be illuminated continuously;

**alternating** – both indicators, viewed together, shall alternate between two defined states, and the two indicators shall always be in opposite states, out of phase;

**flashing** – each indicator, viewed independently, shall alternate between two states, and if the indicators are flashing, they shall flash together, in phase.

###### A.2.3.4.2.2 Priority

The different states of the network status indicators shall follow a strict priority scheme. If more than one condition exists, the network status indicators shall display the state with the highest priority as specified in Table A.3.

**Table A.3 – Network status indicators**

| Priority | Indicator state | How to view | Cause |
|---|---|---|---|
| highest (1) | both steady off | viewed together | reset or no power |
| 2 | both steady red | | failed link interface |
| 3 | alternating red / green | | self test |
| 4 | alternating red / off | | bad node configuration (such as duplicate MAC ID) |
| 5 | steady off | viewed independently | channel disabled or channel not supported |
| 6 | flashing red / green | | invalid link configuration |
| 7 | flashing red / off | | link fault or no DLPDUs received |
| 8 | flashing green / off | | temporary channel error, or listen-only |
| lowest (9) | steady green | | normal operation |

#### A.2.3.4.2.3 Steady green

During normal operation, when DLPDUs are being received without detected errors, the network status indicator for any enabled channel shall be steady green.

#### A.2.3.4.2.4 Steady red

If the link interface has faulted, both indicators shall be steady red.

#### A.2.3.4.2.5 Steady off

If one of the PhL entities is disabled, the indicator for the disabled channel shall be off. A channel can be disabled either because the device has only a single PhL or because it has two PhLs but exists on a non-redundant link. Only deactivating the entire network interface shall cause both indicators to be extinguished (off).

#### A.2.3.4.2.6 Flashing green / off

A flashing green / off pattern shall signify a temporary channel error. The two channels shall be treated independently with respect to these errors. An error on channel A shall be indicated by any of three events:

`Ph_status_indication` from channel A does not equal Normal;
`DLL_badFCS_indication(CHA)` is asserted;
`DLL_event_indication(DLL_EV_errA)` is asserted.

Likewise, an error on channel B shall be indicated by any of three events:

`Ph_status_indication` from channel B does not equal Normal;
`DLL_badFCS_indication(CHB)` is asserted;
`DLL_event_indication(DLL_EV_errB)` is asserted.

If more than one error occurs in a single DLPDU, only one channel error shall be declared. There shall be at most one error event per DLPDU per channel. Each PhL shall have an independent `Ph_status_indication`. The Data Link Layer shall report errors on at least the channel selected for presentation to higher levels. The DLL may also report errors on the other channel.

The flashing green state shall be entered whenever a channel error is detected. The flashing green state shall transition to the steady green state if no errors are detected for 4 s ± 1 s on a given channel.

When the Data Link Layer is in a listen-only state, the indicator for any enabled channels shall flash green at priority 8 (see Table A.3).

NOTE The ControlNet object can be used to force the Data Link Layer to enter the listen-only state.

#### A.2.3.4.2.7 Flashing red / off

A flashing red / off pattern shall signify a severe link error. If the `Ph_frame_indication` from one of channels remains false for a 1,0 s ± 0,1 s period, the respective indicator shall assume the flashing red state for the next 1,0 s ± 0,1 s period.

For devices with redundant channels enabled, if the error rate for one of the channels exceeds that of the other channel by more than one error per N DLPDUs received, then the channel with the higher rate shall flash red. N shall be in the range $10^6$ through $2\times10^6$. The interval used to measure channel error rate shall be $5\times10^6$ DLPDUs.

#### A.2.3.4.2.8 Flashing red / green

A flashing red / green pattern shall signify a bad link configuration. The network status indicators shall enter the flashing red / green state if the network attachment monitor (NAM) indicates that the node has been asked to use invalid or unsupported link parameters.

To optionally indicate a receive queue overflow or a transmit queue underflow, the network status indicators shall enter the flashing red / green state. The network status indicators shall remain in this state at least 3 s even if the overflow condition has cleared.

#### A.2.3.4.2.9 Alternating red / off

An alternating red / off pattern shall signify an invalid node configuration. This state shall be entered if the `DLL_event_indication` has either of the following values:

`DLL_EV_dupNode`;
`DLL_EV_rogue`.

This state shall be left immediately when the error condition clears.

#### A.2.3.4.2.10 Alternating red / green

An alternating red / green pattern shall signify that the Data Link Layer is in a self-test mode. The network status indicators shall remain in this state for at least 3 s even if the self-test has completed.

### A.2.3.4.3 Labeling

The network status indicators shall be labeled using the symbols drawn in Figure A.1 and Figure A.2. The first channel shall be labeled with the outline of a channel icon. The second channel shall be labeled with a filled in channel icon. The first and second channels may be labeled with an uppercase A and B, respectively.

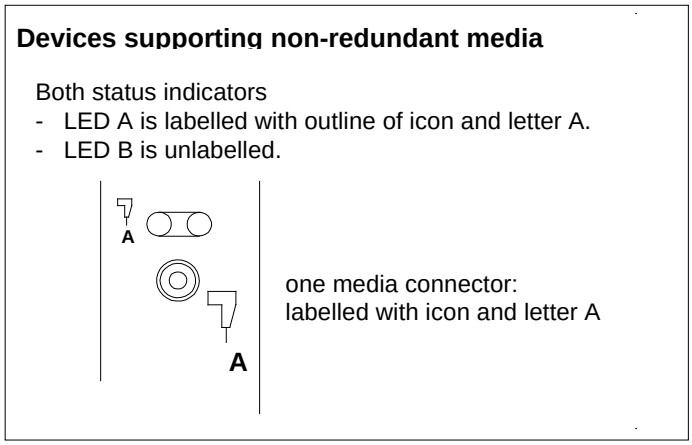



**Figure A.1 – Non redundant network status indicator labeling**

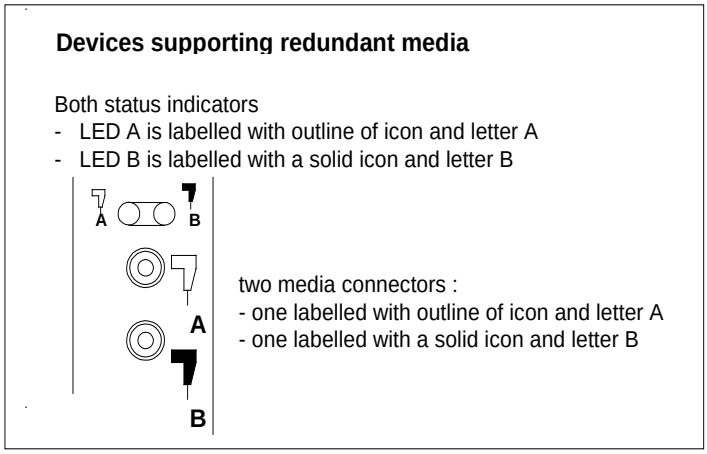



**Figure A.2 – Redundant network status indicator labeling**

### A.2.4 Fieldbus specific indicator requirements (2)

#### A.2.4.1 General requirements (2)

CP 2/2 does not require a device to have indicators.

Two types of status indicators may be provided:

– one module status indicator;

– one or several network status indicators.

NOTE 1 The fieldbus supporting consortia (see <www.odva.org>) defines an Industrial Performance Level that requires a device to support both the module status and network status indicators as specified in this standard.

NOTE 2 A device with more than one communication port would have only one module status indicator, but more than one network status indicator (one per port).

NOTE 3 Devices are encouraged to have an indicator that displays the state of link (e.g. link status, tx/rx, collision) following generally accepted industry practices (as used in devices such as switches).

#### A.2.4.2 Indicators at power up

An indicator test shall be performed at power-up. To allow a visual inspection, the following sequence shall be performed:

– first indicator green, all other indicators off;
– first indicator on green for approximately 0,25 s;
– first indicator on red for approximately 0,25 s;
– first indicator on green;
– second indicator (if present) on green for approximately 0,25 s;
– second indicator (if present) on red for approximately 0,25 s;
– second indicator (if present) off.

If other indicators are present, each indicator shall be tested in sequence as prescribed by the second indicator above. If a Module Status indicator is present, it shall be the first indicator in the sequence, followed by any Network Status indicator present. After completion of this power up test, the indicator(s) shall turn to a normal operational state.

#### A.2.4.3 Module status indicator

Behavior of the module status indicator shall be as specified in A.2.2.4.

The module status indicator shall be labeled with one of the following:

“MS”;
“Mod”;
“Mod status”;
“Module status”.

#### A.2.4.4 Network status indicators

##### A.2.4.4.1 Description

The network status indicator shall be a bicolor (red/green) indicator that represents the status of the network interface. Devices with multiple physical communication ports but a single IP address shall have a single network status indicator (e.g., a 2-port device supporting DLR). Devices that support multiple IP addresses may use a network status indicator for each IP address interface, or may use a single indicator for all interfaces (per behavior in Table A.4).

##### A.2.4.4.2 States

The network status indicator states shall be as specified in Table A.4.

**Table A.4 – Network status indicator**

| Indicator state | Summary | Cause |
|---|---|---|
| Steady off | Not powered,<br>no IP address | The device is powered off, or is powered on but with no IP address configured (Interface Configuration attribute of the TCP/IP Interface object) |
| Flashing green | No connections | An IP address is configured, but no Type 2 connections are established, and an Exclusive Owner connection has not timed out |
| Steady green | Connected | At least one Type 2 connection (any transport class) is established, and an Exclusive Owner connection has not timed out |
| Flashing red | Connection timeout | An Exclusive Owner connection for which this device is the target has timed out. The network status indicator shall return to steady green only when all timed out Exclusive Owner connections are reestablished.<br>Devices that support a single Exclusive Owner connection shall transition to steady green when any subsequent Exclusive Owner connection is established.<br>Devices that support multiple Exclusive Owner connections shall retain the O->T connection path information when an Exclusive Owner connection times out. The Network LED shall transition from flashing red to steady green only when all connections to the previously timed-out O->T connection points are reestablished.<br>Timeout of connections other than Exclusive Owner connections shall not cause the indicator to flash red.<br>The Flashing Red state applies to target connections only. Originators and Type 2 Routers shall not enter this state when an originated or routed Type 2 connection times out |
| Steady red | Duplicate IP | For devices that support duplicate IP address detection, the device has detected that (at least one of) its IP address is already in use |
| Flashing green and red | Self-test | The device is performing its power-on self-test (POST). During POST the network status indicator shall alternate flashing green and red |

When a single indicator is used to represent multiple IP address interfaces the state of any one interface shall be sufficient to modify the indicator state (per the behavior in the table):

– transition to flashing green when any one interface receives an IP address;

– transition to steady green when a Type 2 connection is established on any interface (and Exclusive Owner is not timed out).;

– transition to flashing red when an Exclusive Owner Type 2 connection times out on any interface;

– transition to steady red when any of the interfaces detects an IP address conflict.

Figure A.3 shows the network status indicator state diagram.

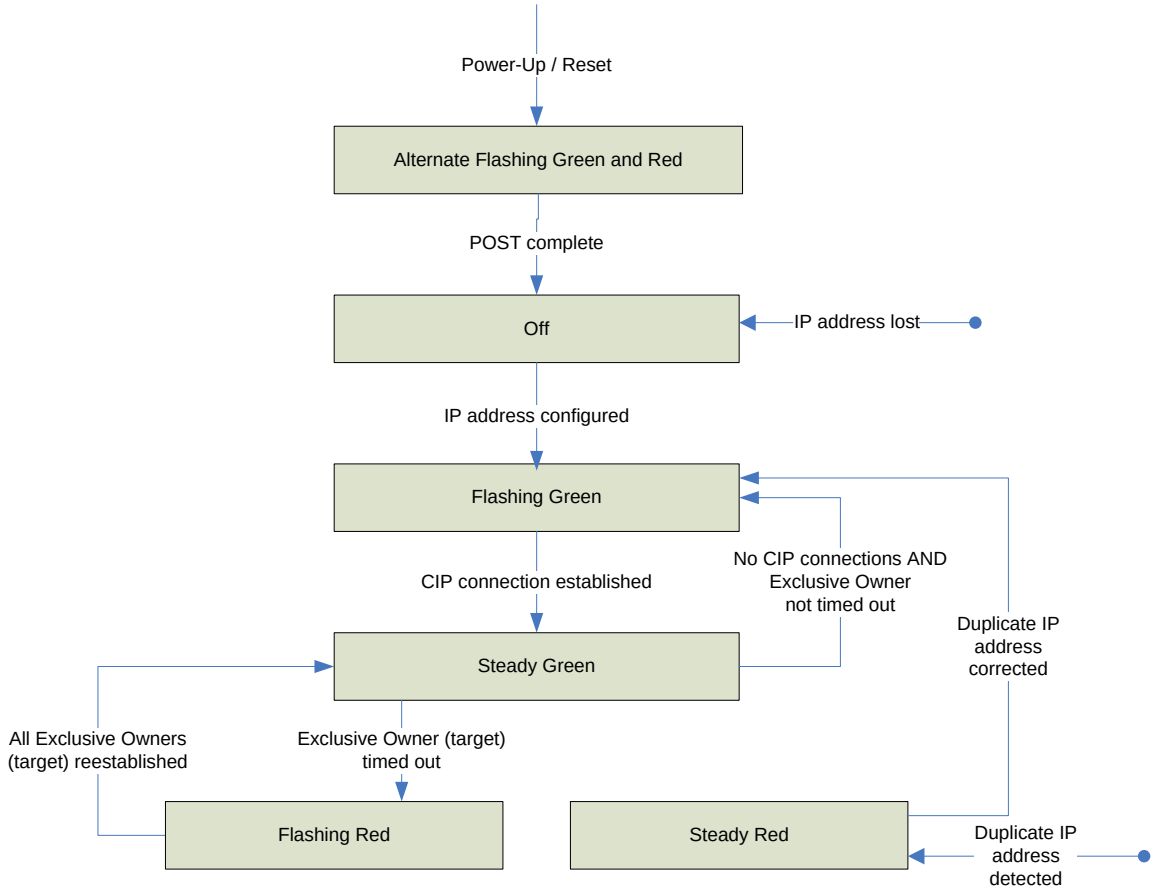



**Figure A.3 – Network status indicator state diagram**

#### A.2.4.4.3 Labeling

The network status indicator shall be labeled with one of the following:

"NS";

"Net";

"Net Status";

"Network Status".

### A.2.5 Fieldbus specific indicator requirements (3)

#### A.2.5.1 General requirements (3)

CP 2/3 does not require a device to have indicators.

If a device supports indicators, then it is recommended to provide:

– one module status indicator;
– one status indicators;

or a combined module/network status indicator.

A device may also support I/O status indicators.

If there is a conflict between turning an indicator on red versus green, the indicator shall be turned on red.

#### A.2.5.2 Indicators at power up

##### A.2.5.2.1 Module and network status indicators at power up

An indicator test shall be performed at power-up. To allow a visual inspection, the following sequence shall be performed:

– network status indicator off;
– module status indicator on green for approximately 0,25 s;
– module status indicator on red for approximately 0,25 s;
– module status indicator on green;
– network status indicator on green for approximately 0,25 s;
– network status indicator on red for approximately 0,25 s;
– network status indicator off.

If a device has other indicators each indicator should be tested in sequence.

##### A.2.5.2.2 Combined module/network status indicators at power up

An indicator test shall be performed at power-up. To allow a visual inspection, the following sequence shall be performed:

– combined module/network status indicator on green for approximately 0,25 s;
– combined module/network status indicator on red for approximately 0,25 s;
– combined module/network status indicator off.

If a device has other indicators each indicator should be tested in sequence.

#### A.2.5.3 Module status indicator

Behavior of the module status indicator shall be as specified in A.2.2.4.

The module status indicator should be labeled with one of the following:

“MS”;

“Module status”.

#### A.2.5.4 Network status indicator

##### A.2.5.4.1 Description

This bicolor (green/red) indicator provides the status of the communication link.

##### A.2.5.4.2 States

The network status indicator states shall be as specified in Table A.5. See the Link Access State Transition Diagram in IEC 62026-3:2008, 5.4 to compare the network status indicator to the Link Access State machine.

**Table A.5 – Network status indicator**

| Indicator state | Summary | Cause |
|---|---|---|
| Steady off | Not powered/ not on–line | Device is not on–line.<br>– The device has not completed the Dup_MAC_ID test yet.<br>– The device may not be powered, look at module status indicator.<br>– No network power present |
| Flashing green[a] | On–line, not connected | Device is on–line but has no connections in the established state.<br>– The device has passed the Dup_MAC_ID test, is on–line, but has no established connections to other nodes.<br>– For a Group 2 Only device it means that this device is not allocated to a master.<br>– For a UCMM capable device it means that the device has no established connections |
| Steady green | Link OK, on–line, connected | The device is on–line and has connections in the established state.<br>– For a Group 2 Only device it means that the device is allocated to a Master.<br>– For a UCMM capable device it means that the device has one or more established connections |
| Flashing red[a] | Connection time–out | One or more I/O Connections are in the Timed–Out state |
| Steady red | Critical link failure | Failed communication device. The device has detected an error that has rendered it incapable of communicating on the network (duplicate MAC ID, or Bus–off) |
| Flashing red / green[b] | Communication faulted and received an Identify Comm Fault request - long protocol | A specific Communication Faulted device. The device has detected a Network Access error and is in the Communication Fault state. The device has subsequently received and accepted an Identify Communication Faulted request - long protocol message |

[a] Indicator flash rates are given in A.2.2.3.

[b] Indicator flash rates are given in IEC 62026-3:2008, 5.2.4.5.4.

Any indicator colors/states not defined in Table A.5 are reserved and shall not be used.

Safety devices have additional requirements, see IEC 61784-3-2.

#### A.2.5.4.3 Labeling

The network status indicator should be labeled with one of the following:

"NS";

"Network Status".

### A.2.5.5 Combined module/network status indicator

#### A.2.5.5.1 Description

This bicolor (green/red) indicator provides limited device and communication status. The combined module/network status indicator indicates whether or not the device has power and is operating properly.

#### A.2.5.5.2 States

The combined module/network status indicator states shall be as specified in Table A.6. See the Link Access State Transition Diagram in IEC 62026-3:2008, 5.4 to compare the combined module/network status indicator to the Link Access State machine.

**Table A.6 – Combined module/network status indicator**

| Indicator state | Summary | Cause |
|---|---|---|
| Steady off | Device not powered/ not on–line | Device is not on–line.<br>– The device has not completed the Dup_MAC_ID test yet.<br>– The device may not be powered |
| Steady green | Device operational and on–line, connected | The device is operating in a normal condition and the device is on–line with connections in the established state.<br>– For a Group 2 Only device it means that the device is allocated to a Master.<br>– For a UCMM capable device it means that the device has one or more established connections |
| Flashing green[a] | Device operational and on–line, not connected<br>or<br>Device on–line and device needs commissioning | The device is operating in a normal condition and the device is on–line with no connections in the established state.<br>– The device has passed the Dup_MAC_ID test, is on–line, but has no established connections to other nodes.<br>– For a Group 2 Only device it means that this device is not allocated to a master.<br>– For a UCMM capable device it means that the device has no established connections.<br>– Configuration missing, incomplete or incorrect |
| Flashing red[a] | Minor Fault and/or connection time–out and/or no network power | Any one or more of the following conditions.<br>– Recoverable fault.<br>– One or more I/O connections are in the Timed–Out state.<br>– No network power present |
| Steady red | Critical fault or critical link failure | The device has an unrecoverable fault; may need replacing.<br>Failed communication device. The device has detected an error that has rendered it incapable of communicating on the network (duplicate MAC ID, or Bus–off) |
| Flashing red / green[b] | Communication faulted and received an identify comm fault request - long protocol | A specific Communication Faulted device. The device has detected a Network Access error and is in the Communication Fault state. The device has subsequently received and accepted an Identify Communication Faulted request - long protocol message |

[a] Indicator flash rates are given in A.2.2.3.

[b] Indicator flash rates are given in IEC 62026-3:2008, 5.2.4.5.4.

Any indicator colors/states not defined in Table A.6 are reserved and shall not be used.

A combined module/network status indicator shall not be used for safety devices.

#### A.2.5.5.3 Labeling

The combined module/network status indicator should be labeled with one of the following:

"MNS";

"Mod/Net Status";

"Module/Network Status".

#### A.2.5.6 I/O status indicator

##### A.2.5.6.1 Description

This bicolor (green/red) indicator provides information concerning the states of inputs and/or outputs.

The intent of the I/O Status indicator is to inform the user whether this device has outputs under control and whether any outputs or inputs are active (e.g. outputs active, inputs producing) or faulted. The indicator is intended to reflect the mode/state of the inputs and outputs, not necessarily the on/off condition of the I/O points themselves.

##### A.2.5.6.2 States

The specific definition of the I/O status indicator colors and states reside in the individual definitions of the corresponding application objects that specify the use of this indicator.

Table A.7 provide guidelines governing the use of the I/O status indicator.

**Table A.7 – I/O status indicator**

| Indicator state | Summary | Cause |
|---|---|---|
| Steady off | Output(s) inactive<br>Input(s) inactive | All outputs are inactive.<br>All inputs are inactive |
| Steady green | Output(s) active<br>Input(s) active | One or more outputs are active and under control, and no outputs are “faulted”.<br>One or more inputs are active and producing data, and no inputs are “faulted” |
| Flashing green | Output(s) idle | One or more outputs are idle and no outputs are active nor “faulted” |
| Flashing red | Output(s) faulted<br>Inputs(s) faulted | One or more outputs are “faulted”, maybe in the fault state.<br>One or more inputs are “faulted”, maybe in the fault state |
| Steady red | Output(s) forced off<br>Input unrecoverable fault | One or more outputs are forced off (may be an unrecoverable Fault).<br>One or more inputs has an unrecoverable fault |
| NOTE Indicator flash rates are given in A.2.2.3. | | |

Any indicator colors/states not defined in Table A.7 are reserved and shall not be used.

##### A.2.5.6.3 Labeling

The I/O status indicator should be labeled with one of the following:

“IO”;

“I/O”;

“I/O status”.

## A.3 Switches

### A.3.1 Common switch requirements

#### A.3.1.1 General

All switches shall behave in the same way for the same function. For example, all network address switches for devices shall behave in the same way.

If a device has switches, then it is recommended that the device provides the means by which the switch setting can be determined remotely using messaging services.

Safety devices have additional requirements, see IEC 61784-3-2.

#### A.3.1.2 Software override of switches

If a user accessible switch (e.g. MAC ID or bit rate switch) can be overwritten by software, then an indicator shall be present that indicates whether or not it has been overwritten.

If an attribute is configurable with a switch and it cannot be overwritten with software, then the device shall respond with the error “Attribute not settable” to a service which attempts to set the attribute.

#### A.3.1.3 Times that switches are valid

Switches shall comply with either of the following rules:

– the switch shall be live at all times;
– the switch shall be live only at power turn on (PTO).

### A.3.2 Fieldbus specific switch requirements (1)

#### A.3.2.1 Network address switches

##### A.3.2.1.1 Appearance

If the MAC ID is able to be set using switches, they shall be indicated in decimal format. This may be accomplished by using a rotary, thumbwheel, pushwheel, or other type of switch. The most significant digit shall be to the left or to the top of the least significant digit.

##### A.3.2.1.2 When to read switch

Network address switches shall only be read at power turn on (PTO). If the network address switches are changed after power turn on, a minor fault shall be detected and the module status indicator shall indicate a non-critical fault (flashing red). The MAC ID of the node used by the Data Link Layer shall remain at the value read during PTO. The current setting of the network address switches shall be stored in the ControlNet object so that another node may remotely diagnose the non-critical fault.

##### A.3.2.1.3 Labeling

The network address switches shall be labeled with one of the following:

“NA”;

“address”;

“net address”.

### A.3.3 Fieldbus specific switch requirements (2)

There are no switch requirements for CP 2/2.

### A.3.4 Fieldbus specific switch requirements (3)

#### A.3.4.1 Network address switches

##### A.3.4.1.1 Appearance

If DIP switches are used to set the MAC ID they shall be in binary format. If rotary, thumbwheel or pushwheel switches are used, then decimal format is required. The most significant digit shall always be to the left or to the top of the device when the user is trying to configure the switches.

##### A.3.4.1.2 Labeling

The network address switches shall be labeled with one of the following:

"NA";

"Node Address".

#### A.3.4.2 Bit rate switches

##### A.3.4.2.1 Appearance

If switches are used to set the CP 2/3 bit rate they shall be encoded as specified in Table A.8.

**Table A.8 – Bit rate switch encoding**

| Bit rate | Switch setting |
|---|---|
| 125 kbit/s | 0 |
| 250 kbit/s | 1 |
| 500 kbit/s | 2 |

##### A.3.4.2.2 Labeling

The bit rate switches shall be labeled with one of the following:

"DR";

"Data Rate".

# Bibliography

NOTE All parts of the IEC 61158 series, as well as IEC 61784-1 and IEC 61784-2 are maintained simultaneously. Cross -references to these documents within the text therefore refer to the editions as dated in this list of bibliographic references.

IEC 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-2:2014, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 61784-1:2014, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

IEC 61784-2:2014, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

ISO/IEC 8802 (all parts), *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements*

ISO/IEC 9314-2, *Information processing systems – Fibre Distributed Data Interface (FDDI) – Part 2: Token Ring Media Access Control (MAC)*

ISO/IEC 10731, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

ANSI X3.66, *Advanced data communication control procedures (ADCCP)*

ANSI X3.159, *Information Systems – Programming Language C*

ANSI X3J16, *Programming Language C++*

IETF RFC 791, *Internet Protocol*, available at <http://www.ietf.org>

IETF RFC 793, *Transmission Control Protocol*, available at <http://www.ietf.org>

IETF RFC 1350, *The TFTP Protocol (Revision 2)*, available at <http://www.ietf.org>

IETF RFC 4702, *The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option*, available at <http://www.ietf.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 1: Common Industrial Protocol (CIP™) – Edition 3.13, November 2012*, available at <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 2: EtherNet/IP™ Adaptation of CIP – Edition 1.14, November 2012*, available at <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 3: DeviceNet™ Adaptation of CIP – Edition 1.12, November 2011*, available at <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 4: ControlNet™ Adaptation of CIP – Edition 1.7, April 2011*, available at <http://www.odva.org>

_____

# SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

# RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

## Partie 4-2: Spécification du protocole de la couche liaison de données – Eléments de type 2

## AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.

L'attention est attirée sur le fait que l'utilisation du type de protocole associé est restreinte par les détenteurs des droits de propriété intellectuelle. En tout état de cause, l'engagement de renonciation partielle aux droits de propriété intellectuelle pris par les détenteurs de ces droits autorise l'utilisation d'un type de protocole de couche avec les autres protocoles de couche du même type, ou dans des combinaisons avec d'autres types autorisées explicitement par les détenteurs des droits de propriété intellectuelle pour ce type.

NOTE Les combinaisons de types de protocole sont spécifiées dans la CEI 61784-1 et la CEI 61784-2.

La Norme internationale CEI 61158-4-2 a été établie par le sous-comité 65C: Réseaux industriels, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

Cette troisième édition annule et remplace la deuxième édition parue en 2010. Cette édition constitue une révision technique.

Les principales modifications par rapport à l'édition précédente sont énumérées ci-dessous.

- Addition de conventions en 3.6
- Mises à jour de l'objet ControlNet en 7.2
- Addition de caractéristique d'attribut V/NV manquante en 7.5, 7.6 et 7.7
- Extensions et clarifications de l'objet d'interface TCP/IP en 7.5
- Extensions et clarifications de l'objet Ethernet Link en 7.6
- Extensions et clarifications de l'objet CCO en 7.8
- Extensions et mises à jour de l'objet DLR en 7.9
- Mises à jour de l'objet QoS en 7.10
- Addition de l'objet de port en 7.11
- Mises à jour des diagrammes d'états DL en 8.1 et en 9.2
- Extensions et mises à jour du protocole DLR à l'Article 10
- Mise à jour du comportement des indicateurs en A.2.2 et A.2.3
- Corrections rédactionnelles diverses

Le texte de cette norme est issu des documents suivants:

| FDIS | Rapport de vote |
|---|---|
| 65C/762/FDIS | 65C/772/RVD |

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61158, publiées sous le titre général *Réseaux de communication industriels – Spécifications des bus de terrain*, peut être consultée sur le site web de la CEI.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

**IMPORTANT – Le logo *"colour inside"* qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.**

# INTRODUCTION

La présente partie de la CEI 61158 appartient à la série de normes visant à faciliter l'interconnexion des composants du système d'automatisation. Elle est liée aux autres normes de la série telle que définie par le modèle de référence de bus de terrain « à trois couches » décrit dans la CEI 61158-1.

Le protocole de liaison de données assure un service de liaison de données en s'appuyant sur les services offerts par la couche physique. La présente norme a pour principal objet de préciser un ensemble de règles de communication, exprimées sous la forme de modes opératoires que doivent réaliser des entités de liaison de données homologues (DLE) au moment de la communication. Ces règles de communication ont pour vocation de fournir une base de développement stable visant à atteindre différents objectifs:

a) guider les implémenteurs et les concepteurs;

b) réaliser les essais et acquérir l'équipement;

c) dans un accord d'intégration des systèmes dans l'environnement de systèmes ouverts;

d) dans le cadre d'une meilleure compréhension des communications à contrainte de temps au sein de l'OSI.

La présente norme porte en particulier sur la communication et l'interfonctionnement des capteurs, des effecteurs et d'autres appareils d'automatisation. Grâce à cette norme associée à d'autres normes des modèles de référence OSI ou de bus de terrain, des systèmes par ailleurs incompatibles peuvent fonctionner ensemble, quelle que soit leur combinaison.

La commission électrotechnique internationale (CEI) attire l'attention sur le fait qu'il est déclaré que la conformité avec le présent document peut impliquer l'utilisation de brevets présentés dans plusieurs paragraphes (voir tableau ci-dessous). Ces brevets sont détenus par leurs inventeurs respectifs sous licence ODVA, Inc:

| | | | |
|---|---|---|---|
| US 5,400,331 | [ODVA] | Communication network interface with screeners for incoming messages | Paragraphe 3.4, Articles 4 à 9 |
| US 5,471,461 | [ODVA] | Digital communication network with a moderator station election process | |
| US 5,491,531 | [ODVA] | Media access controller with a shared class message delivery capability | |
| US 5,493,571 | [ODVA] | Apparatus and method for digital communications with improved delimiter detection | |
| US 5,537,549 | [ODVA] | Communication network with time coordinated station activity by time slot and periodic interval number | |
| US 5,553,095 | [ODVA] | Method and apparatus for exchanging different classes of data during different time intervals | |
| US 8,244,838 | [ODVA] | Industrial controller employing the network ring topology | Article 10 |

La CEI ne prend pas position quant à la preuve, à la validité et à la portée de ces droits de propriété.

L'ODVA et les détenteurs de ces droits de propriété ont donné l'assurance à la CEI que l'ODVA consent à négocier des licences avec des demandeurs du monde entier, soit sans frais soit à des termes et conditions raisonnables et non discriminatoires. A ce propos, la déclaration de l'ODVA et des détenteurs de ces droits de propriété est enregistrée à la CEI. Des informations peuvent être être demandées à:

[ODVA] ODVA, Inc.
2370 East Stadium Boulevard #1000
Ann Arbor, Michigan 48104
USA
Attention: Office of the Executive Director

courrier électronique: odva@odva.org

L'attention est d'autre part attirée sur le fait que certains des éléments du présent document peuvent faire l'objet de droits de propriété autres que ceux qui ont été mentionnés ci-dessus. La CEI ne saurait être tenue pour responsable de l'identification de ces droits de propriété en tout ou partie.

L'ISO (www.iso.org/patents) et la CEI (http://patents.iec.ch) maintiennent des bases de données, consultables en ligne, des droits de propriété pertinents à leurs normes. Les utilisateurs sont encouragés à consulter ces bases de données pour obtenir l'information la plus récente concernant les droits de propriété.

# RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

## Partie 4-2: Spécification du protocole de la couche liaison de données – Eléments de type 2

## 1 Domaine d'application

### 1.1 Généralités

La couche de liaison de données assure les communications de messagerie à contrainte de temps de base entre les appareils d'un environnement d'automatisation.

Ce protocole offre des opportunités de communication séquentielle et synchrone cyclique à toutes les entités de liaison de données participantes. Un accès planifié de premier plan est proposé à toutes les activités à contrainte de temps, un accès non planifié l'étant aux activités moins critiques.

Des transferts déterministes et synchronisés peuvent être assurés à des intervalles cycliques jusqu'à 1 ms pour des appareils distants de 25 km. Cette performance peut être adaptée de manière dynamique et en ligne en reconfigurant les paramètres de la liaison locale sans interrompre le fonctionnement normal. De la même manière, des connexions DL et de nouveaux appareils peuvent être ajoutés ou retirés pendant le fonctionnement normal.

Ce protocole offre la possibilité de maintenir la synchronisation d'horloge d'une liaison étendue supérieure à 10 μs.

Ce protocole permet d'optimiser chaque opportunité d'accès en concaténant plusieurs DLSDU et DLPCI associés en une seule DLPDU, améliorant le transfert de données des entités de liaison de données qui émettent activement plusieurs flux de données.

La taille maximale du système est un nombre illimité de liaisons de 99 nœuds, comportant chacune 255 adresses DLSAP. Chaque liaison comporte un maximum de $2^{24}$ homologues connexes et DLCEP d'éditeur.

### 1.2 Spécifications

La présente norme spécifie

a) les modes opératoires de transfert opportun des données et des informations de commande entre une entité utilisateur de liaison de données et une entité utilisateur homologue, et parmi les entités de liaison de données formant le fournisseur de service de liaison de données distribué;

b) la structure des DLPDU de bus de terrain utilisée par le protocole de la présente norme pour le transfert des données et des informations de commande, et leur représentation sous forme d'unités de données d'interface physique.

### 1.3 Procédures

Les procédures sont définies en termes

a) d'interactions entre les entités DL (DLE) homologues par l'échange de DLPDU de bus de terrain;

b) d'interactions entre un fournisseur de service DL (DLS) et un utilisateur DLS au sein du même système par l'échange de primitives DLS;

c) d'interactions entre un fournisseur DLS et un fournisseur de service Ph au sein du même système par l'échange de primitives de service Ph.

### 1.4 Applicabilité

Ces modes opératoires s'appliquent aux instances de communication entre des systèmes qui prennent en charge des services de communications à contrainte de temps dans la couche de liaison de données des modèles de référence OSI ou de bus de terrain, et qui peuvent être connectés dans un environnement d'interconnexion de systèmes ouverts.

Les profils sont un moyen simple à plusieurs attributs de récapituler les capacités d'une mise en œuvre et donc son applicabilité en fonction des différents besoins de communications à contrainte de temps.

### 1.5 Conformité

La présente norme spécifie également les exigences relatives aux systèmes mettant en œuvre ces modes opératoires. La présente norme ne comporte aucun essai visant à démontrer la conformité à ces exigences.

## 2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

NOTE Toutes les parties de la série CEI 61158, ainsi que la CEI 61784-1 et la CEI 61784-2 font l'objet d'une maintenance simultanée. Les références croisées à ces documents dans le texte se rapportent par conséquent aux éditions datées dans la présente liste de références normatives.

CEI 61131-3, *Automates programmables – Partie 3: Langages de programmation*

CEI 61158-3-2:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 3-2: Définition des services de la couche liaison de données – Eléments de type 2*

CEI 61158-5-2:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-2: Définition des services de la couche application – Eléments de type 2*

CEI 61158-6-2:2014, *Réseaux de communication industriels – Spécifications des bus de terrain- Partie 6-2: Spécification du protocole de la couche application – Eléments de type 2*

IEC 61588:2009, *Precision clock synchronization protocol for networked measurement and control systems* (disponible en anglais seulement)

CEI 61784-3-2, *Réseaux de communication industriels – Profils – Partie 3-2: Bus de terrain de sécurité fonctionnelle – Spécifications supplémentaires pour CPF 2*

CEI 62026-3:2008, *Appareillage à basse tension – Interfaces appareil de commande-appareil (CDI) – Partie 3: DeviceNet*

ISO/CEI 3309[1], *Technologies de l'information – Télécommunications et échange d'informations entre systèmes – Procédures de commande de liaison de données à haut niveau (HDLC) – Structure de trame*

ISO/CEI 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Le modèle de base*

ISO/CEI 7498-3, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Dénomination et adressage*

ISO/IEC 8802-3, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications* (disponible en anglais seulement)

ISO 11898:1993[2], *Véhicules routiers – Echange d'information numérique – Gestionnaire de réseau de communication à vitesse élevée (CAN)*

IEEE 802.1D-2004, *IEEE standard for local and metropolitan area networks – Media Access Control (MAC) bridges,* disponible à l'adresse <http://www.ieee.org>

IEEE 802.1Q-2005[2], *IEEE standard for local and metropolitan area networks – Virtual bridged local area networks,* disponible à l'adresse <http://www.ieee.org>

IEEE 802.3-2008, *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications,* disponible à l'adresse <http://www.ieee.org>

IETF RFC 951, *Bootstrap Protocol (BOOTP)*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1542, *Clarifications and Extensions for the Bootstrap Protocol*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 2131, *Dynamic Host Configuration Protocol*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 2132, *DHCP Options and BOOTP Vendor Extensions*, disponible à l'adresse <http://www.ietf.org>

_____

1 Cette norme a été supprimée.

2 Une édition plus récente de cette norme a été publiée, mais seule l'édition citée s'applique.

IETF RFC 4541, *Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 5227:2008, *IPv4 Address Conflict Detection*, disponible à l'adresse <http://www.ietf.org>

## 3 Termes, définitions, symboles, abréviations et conventions

Pour les besoins du présent document, les termes, définitions, symboles et abréviations suivants s'appliquent.

### 3.1 Termes et définitions relatifs au modèle de référence

La présente norme repose en partie sur les concepts développés dans l'ISO/CEI 7498-1 et l'ISO/CEI 7498-3, et utilise les termes suivants.

**3.1.1 called-DL-address** [ISO/CEI 7498-3]

**3.1.2 calling-DL-address** [ISO/CEI 7498-3]

**3.1.3 centralized multi-end-point-connection** [ISO/CEI 7498-1]

**3.1.4 correspondent (N)-entities**
**correspondent DL-entities (N=2)**
**correspondent Ph-entities (N=1)** [ISO/CEI 7498-1]

**3.1.5 demultiplexing** [ISO/CEI 7498-1]

**3.1.6 DL-address** [ISO/CEI 7498-3]

**3.1.7 DL-address-mapping** [ISO/CEI 7498-1]

**3.1.8 DL-connection** [ISO/CEI 7498-1]

**3.1.9 DL-connection-end-point** [ISO/CEI 7498-1]

**3.1.10 DL-connection-end-point-identifier** [ISO/CEI 7498-1]

**3.1.11 DL-connection-mode transmission** [ISO/CEI 7498-1]

**3.1.12 DL-connectionless-mode transmission** [ISO/CEI 7498-1]

**3.1.13 DL-data-sink** [ISO/CEI 7498-1]

**3.1.14 DL-data-source** [ISO/CEI 7498-1]

**3.1.15 DL-duplex-transmission** [ISO/CEI 7498-1]

**3.1.16 DL-facility** [ISO/CEI 7498-1]

**3.1.17 DL-local-view** [ISO/CEI 7498-3]

**3.1.18 DL-name** [ISO/CEI 7498-3]

**3.1.19 DL-protocol** [ISO/CEI 7498-1]

**3.1.20 DL-protocol-connection-identifier** [ISO/CEI 7498-1]

| | | |
|---|---|---|
| **3.1.21** | **DL-protocol-control-information** | [ISO/CEI 7498-1] |
| **3.1.22** | **DL-protocol-data-unit** | [ISO/CEI 7498-1] |
| **3.1.23** | **DL-protocol-version-identifier** | [ISO/CEI 7498-1] |
| **3.1.24** | **DL-relay** | [ISO/CEI 7498-1] |
| **3.1.25** | **DL-service-connection-identifier** | [ISO/CEI 7498-1] |
| **3.1.26** | **DL-service-data-unit** | [ISO/CEI 7498-1] |
| **3.1.27** | **DL-simplex-transmission** | [ISO/CEI 7498-1] |
| **3.1.28** | **DL-subsystem** | [ISO/CEI 7498-1] |
| **3.1.29** | **DL-user-data** | [ISO/CEI 7498-1] |
| **3.1.30** | **flow control** | [ISO/CEI 7498-1] |
| **3.1.31** | **layer-management** | [ISO/CEI 7498-1] |
| **3.1.32** | **multiplexing** | [ISO/CEI 7498-3] |
| **3.1.33** | **naming-(addressing)-authority** | [ISO/CEI 7498-3] |
| **3.1.34** | **naming-(addressing)-domain** | [ISO/CEI 7498-3] |
| **3.1.35** | **naming-(addressing)-subdomain** | [ISO/CEI 7498-3] |
| **3.1.36** | **(N)-entity**<br>**DL-entity**<br>**Ph-entity** | [ISO/CEI 7498-1] |
| **3.1.37** | **(N)-interface-data-unit**<br>**DL-service-data-unit (N=2)**<br>**Ph-interface-data-unit (N=1)** | [ISO/CEI 7498-1] |
| **3.1.38** | **(N)-layer**<br>**DL-layer (N=2)**<br>**Ph-layer (N=1)** | [ISO/CEI 7498-1] |
| **3.1.39** | **(N)-service**<br>**DL-service (N=2)**<br>**Ph-service (N=1)** | [ISO/CEI 7498-1] |
| **3.1.40** | **(N)-service-access-point**<br>**DL-service-access-point (N=2)**<br>**Ph-service-access-point (N=1)** | [ISO/CEI 7498-1] |
| **3.1.41** | **(N)-service-access-point-address**<br>**DL-service-access-point-address (N=2)**<br>**Ph-service-access-point-address (N=1)** | [ISO/CEI 7498-1] |
| **3.1.42** | **peer-entities** | [ISO/CEI 7498-1] |
| **3.1.43** | **Ph-interface-control-information** | [ISO/CEI 7498-1] |

**3.1.44 Ph-interface-data** [ISO/CEI 7498-1]

**3.1.45 primitive name** [ISO/CEI 7498-3]

**3.1.46 reassembling** [ISO/CEI 7498-1]

**3.1.47 recombining** [ISO/CEI 7498-1]

**3.1.48 reset** [ISO/CEI 7498-1]

**3.1.49 responding-DL-address** [ISO/CEI 7498-3]

**3.1.50 routing** [ISO/CEI 7498-1]

**3.1.51 segmenting** [ISO/CEI 7498-1]

**3.1.52 sequencing** [ISO/CEI 7498-1]

**3.1.53 splitting** [ISO/CEI 7498-1]

**3.1.54 synonymous name** [ISO/CEI 7498-3]

**3.1.55 systems-management** [ISO/CEI 7498-1]

## 3.2 Termes et définitions relatifs à la convention de service

Pour les besoins du présent document, les termes suivants, définis dans l'ISO/CEI 10731 et relatifs à la couche liaison de données, s'appliquent:

**3.2.1 acceptor**

**3.2.2 asymmetrical service**

**3.2.3 confirm (primitive);**
**requestor.deliver (primitive)**

**3.2.4 deliver (primitive)**

**3.2.5 DL-confirmed-facility**

**3.2.6 DL-facility**

**3.2.7 DL-local-view**

**3.2.8 DL-mandatory-facility**

**3.2.9 DL-non-confirmed-facility**

**3.2.10 DL-provider-initiated-facility**

**3.2.11 DL-provider-optional-facility**

**3.2.12 DL-service-primitive;**
**primitive**

**3.2.13 DL-service-provider**

**3.2.14 DL-service-user**

**3.2.15 DL-user-optional-facility**

**3.2.16 indication (primitive)**
**acceptor.deliver (primitive)**

**3.2.17 multi-peer**

**3.2.18 request (primitive);**
**requestor.submit (primitive)**

**3.2.19 requestor**

**3.2.20 response (primitive);**
**acceptor.submit (primitive)**

**3.2.21 submit (primitive)**

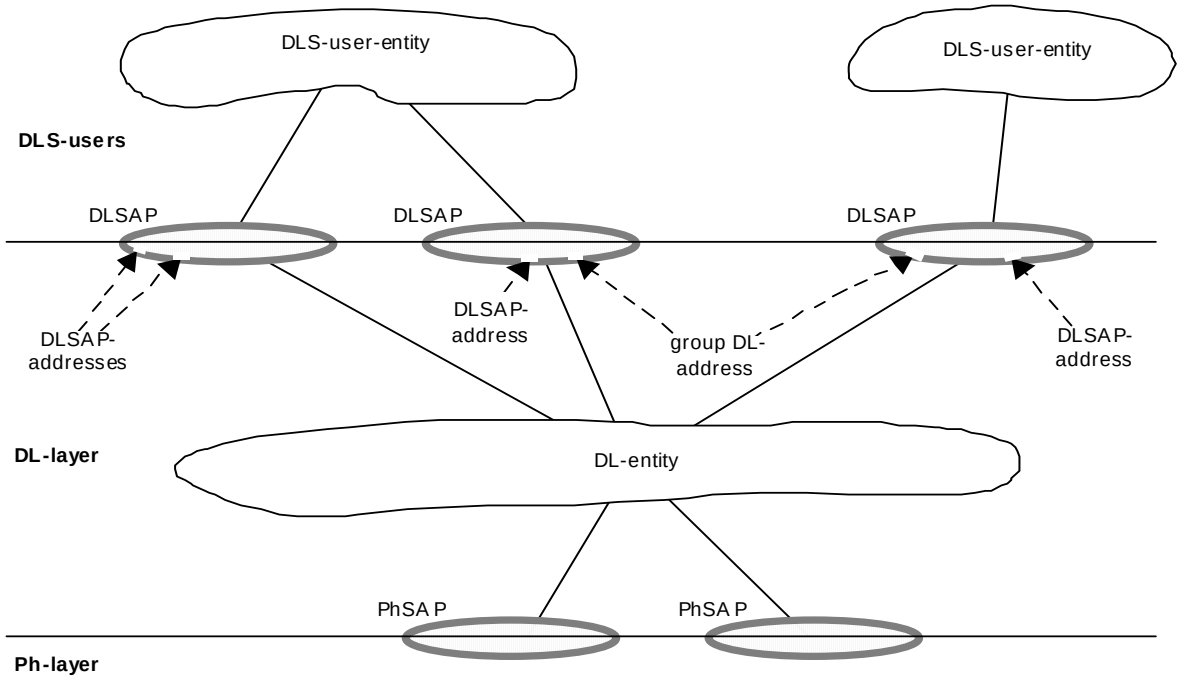
**3.2.22 symmetrical service**

### 3.3 Termes et définitions communs

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

NOTE La plupart des définitions sont communes à plusieurs types de protocole et ne sont pas nécessairement utilisées par chacun d'eux.

**3.3.1**
**segment DL**
**liaison**
**liaison locale**
sous-réseau DL dans lequel l'une des DLE connectées peut communiquer directement sans l'intervention d'un relais DL, à chaque fois que toutes les DLE participant à une instance de communication sont simultanément attentives au sous-réseau DL pendant la/les période(s) de tentative de communication




NOTE 1 Les DLSAP et PhSAP sont décrits comme des éléments ovales à cheval sur la limite entre deux couches adjacentes.

NOTE 2 Les adresses DL sont décrites comme désignant de petits intervalles (points d'accès) dans la partie DLL d'un DLSAP.

NOTE 3 Une seule entité DL peut comporter plusieurs adresses DLSAP et adresses DL de groupe associées à un seul DLSAP.

**Légende**

| Anglais | Français |
|---|---|
| DLS user entity | Entité utilisateur DLS |
| DLS users | Utilisateurs DLS |
| DLSAP addresses | Adresses DLSAP |
| Group DL address | Adresse de groupe DL |
| DL layer | Couche DL |
| DL entity | Entité DL |
| Ph layer | Couche Ph |

**Figure 1 – Relations des DLSAP, des adresses DLSAP et des adresses DL de groupe**

### 3.3.2
### DLSAP
point distinctif au niveau duquel des services DL sont fournis par une seule entité DL à une seule entité de couche supérieure

Note 1 à l'article: Cette définition, déduite de l'ISO/CEI 7498-1, est reprise ici pour faciliter la compréhension de la distinction critique entre les DLSAP et leurs adresses DL. (Voir Figure 1).

**3.3.3**
**adresse DL (SAP)**
adresse DLSAP individuelle désignant le DLSAP d'un utilisateur DLS unique ou adresse DL de groupe désignant plusieurs DLSAP, chacun d'eux appartenant à un seul utilisateur DLS

Note 1 à l'article: Cette terminologie a été choisie car l'ISO/CEI 7498-3 ne permet pas d'utiliser le terme adresse DLSAP pour désigner plusieurs DLSAP uniques d'un seul utilisateur DLS.

**3.3.4**
**adresse DLSAP (individuelle)**
adresse DL qui désigne un seul DLSAP dans la liaison étendue

Note 1 à l'article: Une seule entité DL peut comporter plusieurs adresses DLSAP associées à un seul DLSAP.

**3.3.5**
**liaison étendue**
sous-réseau DL composé d'un nombre maximal de liaisons interconnectées par des relais DL et partageant un espace de nom DL unique (adresse DL) dans lequel toutes les entités DL connectées peuvent communiquer, soit directement, soit à l'aide d'une ou de plusieurs de ces entités relais DL

Note 1 à l'article: Une liaison étendue peut être composée d'une seule liaison.

**3.3.6**
**trame**
synonyme discrédité de DLPDU

**3.3.7**
**adresse DL de groupe**
adresse DL qui désigne potentiellement plusieurs DLSAP dans la liaison étendue. Une entité DL unique peut comporter plusieurs adresses DL de groupe avec un seul DLSAP. Une entité DL unique peut également comporter une seule adresse DL de groupe avec plusieurs DLSAP

**3.3.8**
**nœud**
entité DL unique telle qu'elle se présente sur une liaison locale

**3.3.9**
**utilisateur DLS destinataire**
utilisateur du service DL auquel sont destinées les données utilisateur DL

Note 1 à l'article: Un utilisateur de service DL peut être à la fois un utilisateur DLS expéditeur et destinataire.

**3.3.10**
**utilisateur DLS expéditeur**
utilisateur du service DL à la source des données utilisateur DL

### 3.4 Définitions de type 2 supplémentaires

**3.4.1**
**intervalle réel entre paquets**
**API (Actual Packet Interval)**
mesure de la fréquence de génération de données Lpacket par une connexion particulière

**3.4.2**
**allouer**
prendre une ressource dans une zone commune et attribuer la ressource en question à l'usage exclusif d'une entité spécifique

**3.4.3**
**application**
fonction ou structure de données pour laquelle des données sont consommées ou produites

**3.4.4**
**objets applicatifs**
classes contenant plusieurs objets permettant de gérer et d'assurer l'échange dynamique des messages sur le réseau et au sein de l'appareil

**3.4.5**
**attribut**
description d'une caractéristique ou fonction, visible par un observateur externe, d'un objet

Note 1 à l'article: Les attributs d'un objet contiennent des informations sur les portions variables d'un objet. En règle générale, ils fournissent des informations de statut ou régissent l'action d'un objet. Les attributs peuvent également influer sur le comportement d'un objet. Les attributs se répartissent en deux catégories: les attributs de classe et les attributs d'instance.

**3.4.6**
**comportement**
indication de la manière dont l'objet réagit à des événements particuliers

Note 1 à l'article: Sa description inclut la relation entre les valeurs des attributs et les services.

**3.4.7**
**bit**
unité d'information consistant en un 1 ou un 0

Note 1 à l'article: Il s'agit de la plus petite unité de données qui puisse être transmise.

**3.4.8**
**blocage ou temps de blocage**
durée nécessaire après la transmission avant que le nœud ne soit autorisé à recevoir des données

**3.4.9**
**client**
1) Objet qui utilise les services d'un autre objet (serveur) pour exécuter une tâche
2) Emetteur d'un message auquel un serveur réagit

**3.4.10**
**objets de communication**
composants permettant de gérer et d'assurer l'échange dynamique des messages sur le réseau (objet ControlNet ou UCMM (Unconnected Message Manager), par exemple)

**3.4.11**
**connexion**
liaison logique entre deux objets applicatifs au sein du même appareil ou d'appareils différents

**3.4.12**
**ID de connexion**
**CID (Connection ID)**
identifiant attribué à une transmission et associé à une connexion particulière entre des producteurs et des consommateurs, permettant d'identifier une partie spécifique des informations d'application

Note 1 à l'article: Dans la liaison de données, il s'agit d'une balise générique.

**3.4.13**
**contrôle de redondance cyclique**
**CRC**
valeur résiduelle calculée à partir d'une matrice de données et utilisée comme signature représentative de la matrice

**3.4.14**
**cyclique**
terme utilisé pour décrire les événements qui se reproduisent de manière régulière et répétitive

**3.4.15**
**surdité**
situation dans laquelle le nœud ne peut pas entendre la DLPDU modérateur, mais peut entendre le trafic de liaison

**3.4.16**
**appareil**
connexion matérielle physique à la liaison

Note 1 à l'article: Un appareil peut contenir plusieurs nœuds.

**3.4.17**
**DLPDU**
unité de données de protocole de liaison de données (Data-Link Protocol Data Unit)

Note 1 à l'article: Une DLPDU est composée d'un MAC ID source, d'aucun ou de plusieurs Lpackets et d'un FCS tel que transmis ou reçu par un PhE associé.

**3.4.18**
**délimiteur de fin**
ensemble unique de M_symbols identifiant la fin d'une DLPDU

**3.4.19**
**erreur**
divergence entre une valeur ou condition calculée, observée ou mesurée et la valeur ou condition spécifiée ou théoriquement correcte

**3.4.20**
**liaison étendue**
liaisons connectées par des routeurs DL (parfois appelées réseau local)

**3.4.21**
**erreur FCS**
erreur qui se produit lorsque la valeur de la séquence de contrôle de trame calculée après réception de tous les octets d'une DLPDU ne correspond pas à la valeur résiduelle prévue

**3.4.22**
**balise fixe**
identifiant (balise) à deux octets qui identifie un service spécifique que doit réaliser soit

a) le nœud destinataire sur la liaison locale qui comporte un MAC ID spécifié, ou

b) tous les nœuds destinataires de la liaison locale

Note 1 à l'article: L'identification du/des nœud(s) cible est comprise dans la balise à deux octets.

**3.4.23**
**trame**
synonyme discrédité de DLPDU

**3.4.24**
**séquence de contrôle de trame**
**FCS (Frame Check Sequence)**
données redondantes issues d'un bloc de données d'une DLPDU (trame), utilisant une fonction de hachage, et enregistrées ou transmises avec le bloc de données, afin de déterminer l'altération des données

**3.4.25**
**balise générique**
identifiant (balise) à trois octets qui identifie une partie spécifique des informations d'application

**3.4.26**
**bande de garde**
intervalle attribué pour la transmission de la DLPDU modérateur

**3.4.27**
**jeton implicite**
mécanisme régissant les droits de transmission

Note 1 à l'article: Aucun message de jeton réel n'est transmis sur le support. Chaque nœud garde une trace du MAC ID du nœud qu'il suppose détenir les droits de transmission. Les droits de transmission sont transmis d'un nœud à l'autre en conservant un enregistrement du dernier nœud qui les a transmis. Un intervalle est utilisé afin d'ignorer un nœud manquant dans la rotation.

**3.4.28**
**registre de jeton implicite**
registre contenant le MAC ID du nœud détenant les droits de transmission

**3.4.29**
**instance**
occurrence physique réelle d'un objet dans une classe, identifiant l'un des objets dans la même classe d'objets

EXEMPLE Californie est une instance de l'état de la classe d'objets.

Note 1 à l'article: Les termes objet, instance et instance d'objet font référence à une instance particulière.

**3.4.30**
**attribut d'instance**
attribut qui est réservé à l'usage exclusif d'une instance d'objet et n'est pas commun à la classe d'objets

**3.4.31**
**instancié**
se dit d'un objet créé dans un appareil

**3.4.32**
**Keeper**
objet chargé de distribuer les données de configuration d'une liaison à tous les nœuds de la liaison

**3.4.33**
**liaison**
ensemble de nœuds dotés de MAC ID uniques, associés à des segments Ph connectés par des répéteurs Ph

**3.4.34**
**little endian**
modèle d'organisation de mémoire et d'organisation de la transmission permettant respectivement de stocker l'octet de poids faible à l'adresse la plus basse et de le placer en premier sur le support

**3.4.35**
**Lpacket**
sous-partie bien définie d'une DLPDU composée

a) d'informations de taille et de commande

b) d'une balise fixe ou générique, et

c) de données utilisateur DLS ou, si la balise porte une signification DL, de données DL

**3.4.36**
**M_symbol(s)**
symboles représentant les bits de données et les informations connexes codés et transmis par la PhL

**3.4.37**
**nœud planifié maximal**
nœud doté du MAC ID le plus élevé pouvant utiliser un temps de référence sur une liaison

**3.4.38**
**nœud non planifié maximal**
nœud doté du MAC ID le plus élevé pouvant utiliser un temps non planifié sur une liaison

**3.4.39**
**Message Router**
objet d'un nœud qui distribue les demandes de message aux objets d'application concernés

**3.4.40**
**modérateur**
nœud doté du MAC ID le plus bas chargé de transmettre la DLPDU modératrice

**3.4.41**
**DLPDU modérateur**
DLPDU transmis par le nœud doté du MAC ID le plus bas afin de synchroniser les nœuds et de distribuer les paramètres de configuration de liaison

**3.4.42**
**connexion multipoint**
connexion d'un nœud à plusieurs autres nœuds

Note 1 à l'article: Les connexions multipoint permettent à plusieurs nœuds de consommateur (abonné) de recevoir des messages provenant d'un seul producteur (éditeur).

**3.4.43**
**réseau**
série de nœuds connectés par un certain type de support de communication

Note 1 à l'article: Les chemins de connexion entre des paires de nœuds peuvent inclure des répéteurs, des routeurs et des passerelles.

**3.4.44**
**moniteur d'accès au réseau**
**NAM (Network Access Monitor)**
composant d'un nœud qui gère la communication avec un nœud temporaire connecté au NAP local des nœuds

**3.4.45**
**port d'accès au réseau**
**NAP (Network Access Port)**
variante de PhL permettant de connecter un nœud temporaire à la liaison par l'intermédiaire du NAP d'un nœud permanent

**3.4.46**
**adresse réseau ou adresse de nœud**
adresse du nœud sur la liaison

Note 1 à l'article: Cette adresse est également appelée MAC ID.

**3.4.47**
**voyants d'état du réseau**
voyants d'un nœud signalant l'état des couches de liaison physiques et des couches de liaison de données

**3.4.48**
**durée de mise à jour du réseau**
**NUT**
intervalle répétitif au cours duquel les données peuvent être envoyées sur la liaison

**3.4.49**
**nœud**
connexion logique à une liaison locale, nécessitant un MAC ID unique

Note 1 à l'article: Un seul appareil physique peut apparaître sous la forme de plusieurs nœuds sur la même liaison locale. Pour les besoins de ce protocole, chaque nœud est considéré comme une DLE distincte.

**3.4.50**
**non concurrence**
situation dans laquelle une transmission est reçue d'un MAC ID imprévu semblant enfreindre le protocole d'accès temporel

Note 1 à l'article: Cela peut se produire lorsqu'une connexion est établie entre deux liaisons de fonctionnement qui ne sont pas synchronisées, mais qui détiennent les mêmes informations de configuration

**3.4.51**
**symbole sans données**
symbole PhL qui enfreint les exigences du codage Manchester biphasé L

**3.4.52**
**objet**
représentation abstraite d'un composant particulier dans un appareil

Note 1 à l'article: Un objet peut être

1) une représentation abstraite des capacités d'un ordinateur. Les objets peuvent être formés de tout ou partie des composants suivants:
   a) données (informations évoluant dans le temps);
   b) configuration (paramètres de comportement);
   c) méthodes (actions qui peuvent être réalisées à l'aide de données et d'une configuration);
2) un ensemble de données (se présentant sous la forme de variables) et de méthodes (modes opératoires) connexes d'utilisation de ces données qui définit clairement l'interface et le comportement.

**3.4.53**
**service spécifique à l'objet**
service défini par une classe d'objets particulière permettant d'exécuter une fonction requise qu'un service classique ne peut assurer

Note 1 à l'article: Un service spécifique à l'objet est unique à la classe d'objets qu'il définit.

**3.4.54**
**émetteur**
client chargé d'établir un chemin de connexion vers la cible

**3.4.55**
**nœud permanent**
nœud dont la connexion au réseau n'utilise par la variante PhL du port d'accès au réseau (NAP)

Note 1 à l'article: Ce nœud peut éventuellement prendre en charge une variante PhL du NAP afin de permettre aux nœuds temporaires de se connecter au réseau.

**3.4.56**
**produire**
acte d'envoyer des données destinées à être reçues par un consommateur

**3.4.57**
**producteur**
nœud chargé de l'envoi des données

**3.4.58**
**support redondant**
système s'appuyant sur plusieurs supports pour prévenir les défaillances de communication

**3.4.59**
**intervalle de paquet demandé**
**RPI (Requested Packet Interval)**
mesure de la fréquence de demande de transmission de Lpackets par l'application d'origine ou de paquets de données provenant de l'application cible

**3.4.60**
**suspect**
nœud ayant reçu une DLPDU modérateur opposé à la configuration de liaison qu'il utilise

**3.4.61**
**planifié**
transfert de données déterministe et répétable sur des durées de mise à jour du réseau prédéfinies

**3.4.62**
**serveur**
objet qui offre des services à un autre objet (client)

**3.4.63**
**service**
opération ou fonction qu'un objet et/ou une classe d'objets exécute à la demande d'un autre objet et/ou une autre classe d'objets

Note 1 à l'article: un ensemble de services communs est défini et des dispositions permettant la définition de services propres à un objet sont fournies. Les services propres à un objet sont des services définis par une classe d'objets particulière afin de remplir une fonction nécessaire qui n'est assurée par aucun service commun.

**3.4.64**
**intervalle**
durée maximale requise pour détecter une transmission prévue

Note 1 à l'article: chaque nœud attend un intervalle pour chaque nœud manquant pendant la transmission du jeton impliqué.

**3.4.65**
**délimiteur de début**
ensemble unique de M_symbols identifiant le début d'une DLPDU

**3.4.66**
**supernœud**
DLE dont le MAC ID est nul

Note 1 à l'article: Cette adresse DL de la DLE est réservée à des fonctions DLL particulières.

**3.4.67**
**identifiant unique de table**
**TUI (Table Unique Identifier)**
code de référence unique utilisé par les objets ControlNet et Keeper pour faire référence à un ensemble cohérent d'attributs de configuration de liaison

**3.4.68**
**balise**
nom abrégé d'une partie spécifique des informations d'application, de deux ou trois octets

**3.4.69**
**cible**
nœud d'extrémité avec lequel une connexion est établie

**3.4.70**
**nœud temporaire**
synonyme de nœud transitoire

**3.4.71**
**tMinus**
nombre de NUT avant qu'un nouvel ensemble de paramètres de configuration de liaison doive être utilisé

**3.4.72**
**tonalité**
instant marquant la limite entre deux NUT

**3.4.73**
**outil**
logiciel exécutable qui interagit avec l'utilisateur pour exécuter une fonction donnée

EXEMPLE Logiciel de planification de liaison (LSS).

**3.4.74**
**id transaction**
zone d'un en-tête UCMM mettant une réponse en correspondance avec la demande associée, qu'un serveur répercute dans le message de réponse

**3.4.75**
**nœud transitoire**
nœud ayant pour seul objet d'être connecté au réseau de manière temporaire à l'aide du support PhL NAP connecté au NAP d'un nœud permanent

**3.4.76**
**gestionnaire de message non connecté**
**UCMM (UnConnected Message Manager)**
composant au sein d'un nœud qui transmet et reçoit des messages explicites non connectés et les envoie directement à l'objet Routeur de message

**3.4.77**
**service non connecté**
service de messagerie qui ne dépend pas de la configuration d'une connexion entre des appareils pour autoriser les échanges d'informations

**3.4.78**
**non planifié**
transfert de données qui utilise la durée attribuée restante dans le NUT à l'issue des transferts planifiés

**3.4.79**
**ID fournisseur**
identification de chaque fabricant/fournisseur par un numéro unique

Note 1 à l'article: les ID fournisseur sont attribués par l'ODVA, Inc.

## 3.5 Symboles et abréviations de type 2

| | | |
|---|---|---|
| **ACD** | Détection de conflit d'adresses (Address Conflict Detection) | |
| **ACM** | Machine de contrôle d'accès (Access control machine) | |
| **BOOTP** | Bootstrap Protocol | [IETF RFC 951] |
| **CA** | Précision de l'horloge | |
| **CAN** | Réseau local du contrôleur | [ISO 11898:1993] |
| **COP** | Mot de passe de l'auteur de la connexion | |
| **DHCP** | Dynamic Host Configuration Protocol | [IETF RFC 2131] |
| **DLR** | Device Level Ring | |
| **DSCP** | Point de code DiffServ | |
| **FCS** | Séquence de contrôle de trame | |
| **IP** | Internet Protocol | [IETF RFC 791] |
| **DEL** | Diode ElectroLuminescente | |
| **LSS** | Logiciel de planification de liaison | |
| **MAC ID** | Adresse MAC d'un nœud | |
| **MSTP** | Multiple Spanning Tree Protocol | [IEEE 802.1Q] |
| **NAM** | Moniteur de connexion au réseau | |
| **NAP** | Port d'accès au réseau | |
| **ND** | Symbole sans données | |
| **NUT** | Durée de mise à jour du réseau | |
| **PT** | Terminal de programmation (connexion temporaire au réseau) | |
| **QoS** | Qualité de service | |
| **Rcv** | Recevoir | |
| **RPI** | Intervalle demandé entre les paquets | |
| **RSTP** | Rapid Spanning Tree Protocol | [IEEE 802.1D] |
| **Rx** | Recevoir | |
| **RxLLC** | Contrôle de liaison logique de réception | |
| **RxM** | Machine destinataire | |
| **SEM** | Matrice d'événement d'état | |
| **SMAX** | MAC ID du nœud planifié maximal | |
| **STD** | Diagramme de transition d'états, utilisé pour décrire le comportement de l'objet | |
| **STP** | Spanning Tree Protocol | [IEEE 802.1D] |
| **TCP** | Terminal Control Protocol | [IETF RFC 793] |
| **TFTP** | Trivial File Transfer Protocol | [IETF RFC 1350] |

| | |
|---|---|
| **TUI** | Identifiant unique de table |
| **Tx** | Transmettre |
| **TxLLC** | Contrôle de liaison logique de transmission |
| **TxM** | Machine de transmission |
| **UCCM** | Gestionnaire de message non connecté |
| **UMAX** | MAC ID du nœud non planifié maximal |
| **USR** | Registre de démarrage non planifié |

### 3.6 Conventions relatives aux objets de gestion de la station

Les objets de gestion de la station sont spécifiés à l'Article 7, au moyen d'attributs, de services et d'un comportement.

Les attributs d'objet sont spécifiés dans des tableaux formatés selon le Tableau 1.

**Tableau 1 – Format des tableaux d'attributs**

| ID de l'attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| Voir a) | Voir b) | Voir c) | Voir d) | Voir e) | Voir 6.1.3 | Voir f) | Voir g) |

a) « ID de l'attribut » désigne la valeur d'identification affectée à un attribut.

b) « Nécessité dans la mise en œuvre » indique si l'attribut est nécessaire ou non dans la mise en œuvre. Un attribut peut être facultatif, exigé ou conditionnel.

Un attribut conditionnel est exigé si certains comportements d'objet et/ou attributs sont mis en œuvre tel que cela est défini par la classe.

Si un attribut est facultatif, une valeur par défaut doit être définie dans la sémantique. La valeur par défaut d'un attribut facultatif doit avoir le même comportement que si l'attribut n'était pas mis en œuvre. Si la valeur par défaut d'un attribut facultatif ne correspond pas au comportement de l'objet lorsque l'objet ne met pas en œuvre l'attribut, la définition d'objet doit déclarer le comportement.

Dans tous les cas, l'adjectif « par défaut » indique une « valeur d'usine par défaut » tel que délivrée par le fournisseur.

c) « Règle d'accès » désigne la manière dont un demandeur peut accéder à un attribut. Les définitions relatives aux règles d'accès sont les suivantes:

- Settable (Set) <Définissable (Définir)> – Au moins l'un des services SET doit accéder à l'attribut. Si le comportement de l'appareil n'exige pas de service SET, une mise en œuvre d'appareil de cet objet n'est pas exigée pour mettre en œuvre l'attribut sous une forme définissable.

  Sauf spécification contraire dans la définition d'objet, les services GET doivent également accéder aux attributs définissables.

- Gettable (Get) <Récupérable (Récupérer)> – Au moins l'un des services GET doit accéder à l'attribut.

d) « NV » indique lorsqu'une valeur d'attribut est maintenue tout au long des cycles d'alimentation. Cette colonne est utilisée dans les définitions d'objet lorsqu'une mémoire non volatile des valeurs d'attribut est exigée. Une entrée « NV » signale que la valeur doit être sauvegardée tandis qu'une entrée « V » signale que la valeur n'est pas sauvegardée.

e) « Nom » fait référence à l'attribut.

f) « Description d'attribut » fournit des informations générales concernant l'attribut.

g) « Sémantique des valeurs » désigne la signification de la valeur de l'attribut.

# 4 Présentation du protocole de liaison de données

## 4.1 Généralités

### 4.1.1 Architecture DLL

La DLL de Type 2 est modélisée sous la forme d'une machine de contrôle d'accès (ACM) intégrée, dotée de fonctions de support de planification, et conçue pour assurer une prise en charge fiable et efficace des services de transfert de données en mode connexion de niveau supérieur et en mode sans connexion. L'interaction avec ces fonctions de niveau supérieur sont des fonctions de gestion de DLL.

La structure PhL et les délimiteurs sont gérés par les fonctions DLL pour la conversion parallèle-série et série-parallèle des demandes M_symbol et des indications.

Dans la couche de liaison de données, la machine de contrôle d'accès est chargée de détecter et de récupérer les interruptions de réseau. L'ACM a pour principaux objectifs

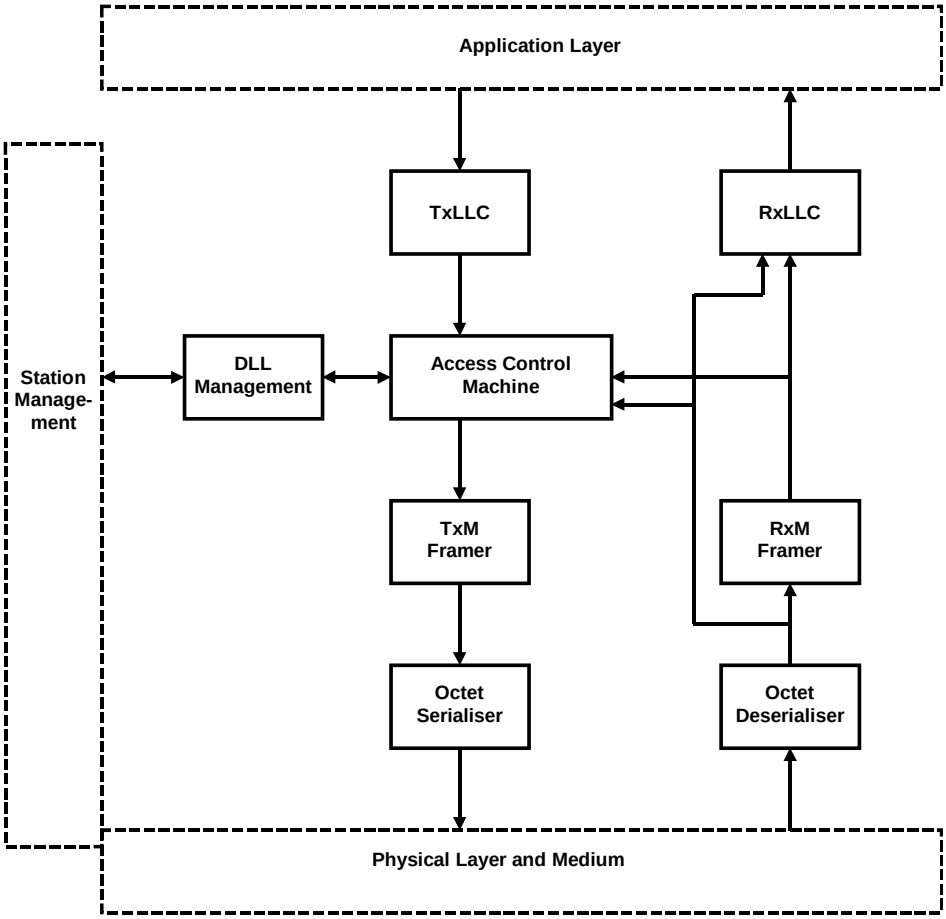
– de s'assurer que le nœud local détecte et utilise intégralement ses emplacements d'accès attribués dans la planification;
– de s'assurer que le nœud local ne gêne pas les transmissions des autres nœuds, particulièrement le nœud modérateur;
– de démarrer le NUT suivant (voir 4.1.2) planifié, que la DLPDU modérateur soit entendue ou pas;
– si le nœud local est le modérateur, de transmettre chaque DLPDU modérateur strictement selon la planification.

La couche de liaison de données contient les composants figurant dans le Tableau 2:

**Tableau 2 – Composants de la couche de liaison de données**

| Composants | Description |
|---|---|
| **Machine de contrôle d'accès (ACM)** | reçoit et transmet les DLPDU de contrôle et les informations d'en-tête, et détermine le déroulement et la durée des transmissions. |
| **Transmit LLC (TxLLC)** | place en mémoire tampon les DLSDU reçues du gestionnaire de station et de l'utilisateur DLS, et détermine celle qu'il convient de transmettre ensuite. |
| **Receive LLC (RxLLC)** | procède à la mise en quarantaine des unités de liaison reçues tant qu'elles n'ont pas été validées par un FCS correct. |
| **Transmit Machine (TxM)** | reçoit les demandes à envoyer aux en-têtes et amorces de fin de bande du DLPDU et les Lpackets de l'ACM, et les découpe en demandes de symbole d'octet qui sont envoyés au convertisseur parallèle-série. |
| **Receive Machine (RxM)** | assemble les Lpackets reçus des symboles d'octet provenant du convertisseur série-parallèle et les soumet au RxLLC. |
| **Convertisseur parallèle-série** | reçoit les symboles d'octet, les code et les convertit, puis les envoie en tant que M_symbols à la PhL. Il est également chargé de générer le FCS. |
| **Convertisseur série-parallèle** | reçoit les M_symbols de la PhL, les convertit en octets et les envoie à la machine destinataire. Il est également chargé de vérifier le FCS. |
| **Interface de gestion DLL** | conserve les variables de gestion de station appartenant à la DLL, et permet de gérer les modifications synchronisées apportées aux paramètres de liaison. |

La disposition interne de ces composants et leurs interfaces sont présentées à la Figure 2. Les pointes de flèche illustrent la direction principale du flux de données et de contrôle.




**Légende**

| Anglais | Français |
|---|---|
| application layer | couche d'application |
| station management | gestion de station |
| DLL management | gestion DLL |
| access control machine | machine de contrôle d'accès |
| TxM framer | structure TxM |
| RxM framer | structure RxM |
| octet serialiser | convertisseur parallèle-série d'octet |
| octet deserialiser | convertisseur série-parallèle d'octet |
| physical layer and medium | couche physique et support |

**Figure 2 – Architecture interne de la couche de liaison de données**

### 4.1.2 Machine de contrôle d'accès (ACM) et fonctions de support de planification

Les fonctions ACM planifient toute la communication entre deux DLE. Le déroulement de cette communication est réglé sur deux niveaux,

1) pour fournir une distribution temporelle précise des opportunités planifiées des communications conçues conformément à la liaison locale et à un réseau étendu de liaisons, et avec une planification préalablement établie,
2) pour assurer une distribution démocratique des opportunités non planifiées des communications arbitraires, en général de manière cycle, mais asynchrone.

Le déroulement précis de la planification, tel qu'indiqué en 1), est essentiel à la prise en charge de nombreuses tâches de contrôle et de collecte des données dans le domaine des

applications de ce protocole. La planification repose sur un cycle de durée fixe et répétitif appelé durée de mise à jour du réseau (NUT). La NUT est maintenue dans un synchronisme proche de tous les nœuds participant de la DLL et est utilisée pour fournir deux types d'opportunité d'accès;

1) Une opportunité pour chaque station active de transférer une DLPDU planifiée contenant plusieurs DLSDU planifiées.

   Seules les DLSDU planifiées comme leurs paramètres QoS peuvent être incluses dans une DLPDU planifiée.

   NOTE Les DLSDU planifiées comme leurs paramètres QoS sont en général des transferts en mode connexion comme leurs paramètres de balise.

2) Une opportunité, pour au moins une station active, d'envoyer une DLPDU (non planifiée) d'arrière-plan contenant plusieurs DLSDU de type QoS. Si le temps est disponible dans la NUT, une opportunité MAC d'arrière-plan est offerte à chaque station active pour leur adresse MAC ID. Le MAC ID de la station de la première opportunité d'arrière-plan de chaque NUT est incrémenté (modulo +1 pour l'ensemble d'adresses d'arrière-plan) pour chaque NUT successive afin de partager démocratiquement le temps disponible entre les stations actives.

Les modes opératoires de prise en charge sont inclus pour le transfert automatique vers les services de planification de sauvegarde et pour gérer l'utilisation du support Ph redondant.

#### 4.1.3 Transfert de données en mode connexion, en mode sans connexion et service DL

Le mode connexion, le mode sans connexion et les services DL offrent les fonctionnalités nécessaires à la

– gestion des interactions entre le fournisseur DL et l'utilisateur DL en convertissant les primitives de demande et de réponse en opérations DLE nécessaires, et en générant des primitives d'indications et de confirmation, le cas échéant,

– détermination de l'adresse DL et des informations de commande à ajouter à chaque DLSDU,

– confirmation locale de l'état de chaque DLSDU sortante,

– formation de DLPDU par concaténation de plusieurs DLSDU d'utilisateur DL pour le transfert efficace sous la forme de PhPDU uniques, soumis à des limitations de taille de DLPDU, aux paramètres QoS et au type d'opportunité d'accès.

### 4.2 Services fournis par la DL

#### 4.2.1 Présentation

La DLL offre des services de transfert de données sans connexion et en mode connexion pour les DLSDU de taille limitée, un service synchronisé en interne, des services de planification du contrôle de l'attribution du temps du service PhL partagé sous-jacent, un service de gestion d'adresse DL (SAP) et de file d'attente, et un service de compression/décompression combinant plusieurs DLSDU en une seule DLPDU de transfert de données pour une utilisation efficace de chaque opportunité d'accès.

#### 4.2.2 QoS

##### 4.2.2.1 Définition

Le paramètre QoS spécifie les options de priorité des DLSDU et les opportunités d'accès. Les valeurs disponibles sont les suivantes:

#### 4.2.2.2 Priorité planifiée par DLL

Le QoS de priorité planifiée assure l'envoi cyclique et acyclique temporel des DLSDU. Le déroulement d'exécution de ce service planifié de bus de terrain peut être précis et répété à 1 ms près. En général, la priorité planifiée est utilisée avec les services en mode connecté.

NOTE Le maître Keeper est chargé de la publication régulière d'un identifiant unique de table (TUI) permettant à chaque nœud de détenir une référence pour l'ensemble en cours de paramètres de fonctionnement de liaison. Cette publication du TUI utilise une balise fixe, et est envoyée avec l'ensemble de priorité QoS à la planification.

#### 4.2.2.3 Priorité élevée DLL (non planifiée)

Le QoS de priorité élevée assure l'envoi acyclique des DLSDU avec un temps supérieur limité pour le délai d'envoi. Les données relatives à cette priorité sont envoyées uniquement si toutes les données planifiées ont été envoyées et si une opportunité d'envoi non planifiée est disponible.

#### 4.2.2.4 Priorité basse DLL (non planifiée)

Le QoS de priorité basse assure l'envoi des DLSDU uniquement en fonction des disponibilités. Les données relatives à cette priorité sont envoyées uniquement si toutes les autres priorités de données ont été envoyées et si une opportunité d'envoi non planifiée est disponible.

NOTE Les priorités élevée et basse sont uniquement utilisées dans un sens local pour définir l'ordre de service des données utilisateur DLS soumises non planifiées.

### 4.3 Structure et définition des adresses DL

#### 4.3.1 Généralités

Les adresses DL sont utilisées comme des adresses MAC ID (indicateurs de nœud de liaison), des adresses de balise fixe (adresses DLSAP et adresses DL de groupe) et des adresses de balise générique (adresses DLCEP).

Le Paragraphe 4.3 définit la forme et la structure des adresses DL et inclut des définitions spécifiques pour certaines adresses DL standard.

Toutes les adresses sont composées d'octets. Chaque octet doit être en premier lieu transféré au bit inférieur de la couche Ph et plusieurs octets doivent être transférés en commençant par l'octet de poids faible (format petit-boutiste).

Trois types d'adresses DL sont utilisés.

#### 4.3.2 Adresse MAC ID

Les adresses MAC ID identifient les nœuds physiques sur la liaison locale (voir Figure 3).

Les valeurs admises sont comprises entre 0 et 99, les valeurs comprises entre 100 et 254 étant réservées.

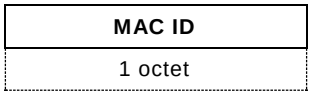



**Figure 3 – Structure de base d'une adresse MAC ID**

Le Tableau 3 spécifie l'utilisation des adresses MAC ID.

**Tableau 3 – Allocation d'adresses MAC ID**

| MAC ID | Signification et objectif |
|---|---|
| 0x0 | adresse temporaire utilisée pour la maintenance |
| SMAX | valeur maximale en cours de l'adresse pour les opportunités d'accès planifiées. SMAX =< UMAX |
| UMAX | valeur maximale en cours de l'adresse pour les opportunités d'accès non planifiées SMAX. =< UMAX =< 0x63 |
| 0x64 à 0xFE | réservé |
| 0xFF | adresse de diffusion à tous les nœuds MAC ID de cette liaison |

### 4.3.3 Adresse de balise générique

Les types d'adresse de balise générique identifient tous les transferts de données des services en mode connecté. La capacité disponible dans la DLL est de 3 octets (voir Figure 4).

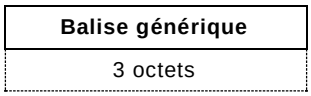



**Figure 4 – Structure de base d'une adresse de balise générique**

Les balises génériques permettent d'envoyer des DLSDU connectées et peuvent être associées à l'une des priorités QoS disponibles (planifiée, élevée, basse).

NOTE La négociation et la gestion des balises génériques et des services en mode connecté relèvent de la responsabilité de l'utilisateur DL.

### 4.3.4 Adresse de balise fixe

Les types d'adresse de balise fixe identifient les DLSAP à l'intérieur du MAC ID de la station désignée. Ils sont utilisés avec tous les transferts de données pour les services en mode sans connexion.

NOTE Les balises fixes permettent d'envoyer des DLSDU sans connexion et sont en principe associées à des priorités QoS non planifiées (élevées, basses). En général, les balises fixes sont utilisées pour gérer la station et établir des connexions.

Chaque adresse de balise fixe doit être composée de deux octets (voir Figure 5).

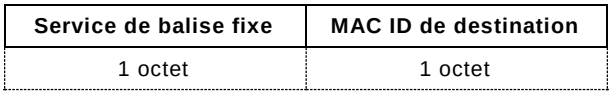



**Figure 5 – Structure de base d'une adresse de balise fixe**

Le premier octet doit spécifier le point d'accès au service dans le nœud de destination (voir Tableau 4). Le deuxième octet doit être l'adresse du nœud de destination. L'adresse de destination doit être un MAC ID ou 0xFF indiquant l'adresse de diffusion de tous les MAC ID sur la liaison locale.

**Tableau 4 – Définitions de service de balise fixe**

| Service de balise fixe | Signification |
|---|---|
| 0x00 | modérateur |
| 0x01 – 0x08 | spécifique au fournisseur |
| 0x09 | demande ping |
| 0x0A – 0x14 | spécifique au fournisseur |

| Service de balise fixe | Signification |
|---|---|
| 0x15 | tMinus |
| 0x16 – 0x28 | spécifique au fournisseur |
| 0x29 | réponse ping |
| 0x2A – 0x3F | spécifique au fournisseur |
| 0x40 – 0x6F | réservé |
| 0x70 – 0x7F | spécifique au fournisseur |
| 0x80 | I'm alive |
| 0x81 | paramètres de liaison |
| 0x82 | réservé |
| 0x83 | UCMM |
| 0x84 | TUI |
| 0x85 | IP (réservé au protocole Internet) |
| 0x86 | WAMI |
| 0x87 | réservé |
| 0x88 | Keeper UCMM |
| 0x89 | Ethernet (utilisé pour le protocole de résolution d'adresse) |
| 0x8A – 0x8B | réservé |
| 0x8C | Répartition temporelle |
| 0x8D – 0x8F | réservé à la répartition temporelle |
| 0x90 | débogage |
| 0x91 – 0xCF | réservé |
| 0xD0 – 0xEF | adresses de groupe |
| 0xF0 – 0xFF | spécifique au fournisseur |

Les balises fixes comprises entre 0xD0 et 0xEF doivent être réservées afin de permettre le tramage de groupe des identifiants de connexion.

## 4.4 Service supposé provenir de PhL

### 4.4.1 Exigences générales

Les Paragraphes 4.4.2 à 4.4.3 contiennent les exigences relatives à l'interface d'une PhL de type 2 qui ne sont pas requises pour les autres types.

– Machine de transmission.
– Machine destinataire.
– Convertisseur parallèle-série.
– Convertisseur série-parallèle.

Cela est nécessaire car les unités de données d'interface Ph (PhIDU) que le type 2 transmet par l'interface DLL-PhL sont des données en série individuelles et des symboles sans données, conçues du côté DLE en tant que M_Symbols (voir Tableau 5), alors que les PhIDU des autres types ne sont pas conceptualisées de cette manière. En conséquence, une couche de liaison de données de type 2 requiert une PhL de type 2.

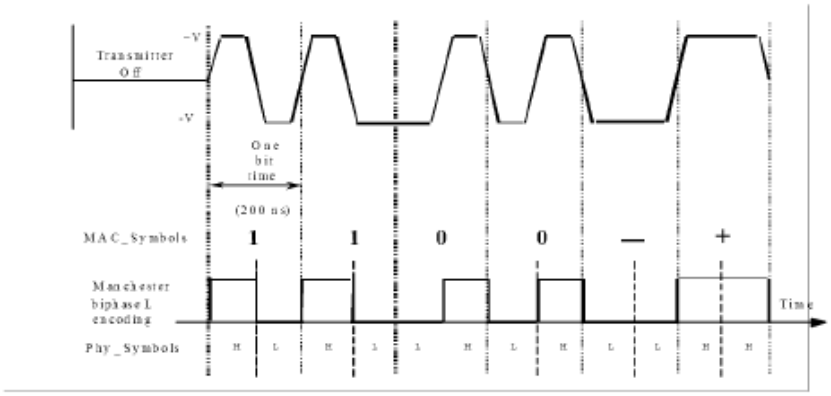
### 4.4.2 Règles de codage des données

Les M_Symbols transférés au niveau de l'interface DLL-PhL sont de quatre types, appelés 0, 1, +, -. Ils seront codés à l'intérieur de la PhL dans les Ph_Symbols appropriés (voir Tableau 5). M_0 et M_1 seront codés en Ph_Symbols représentant les règles de codage biphasé L Manchester. Les symboles M_ND sont des symboles sans données afin de pouvoir créer des schémas de données uniques utilisés pour les délimiteurs de début et de fin.

L'exemple de la forme d'onde de la tension de signal telle qu'elle peut être appliquée à un support conducteur est présenté sous une forme idéalisée dans la Figure 6 pour illustrer les règles de codage des données présentées dans le Tableau 5.

**Tableau 5 – Règles de codage des données**

| Bits de données (nom commun) | Représentation M_Symbol | Codage Ph_Symbol | Codé Manchester |
|---|---|---|---|
| **données « zéro »** | M_0 ou {0} | {L,H} | 0 |
| **données « un »** | M_1 ou {1} | {H,L} | 1 |
| **« non_data+ »** | M_ND+ ou {+} | {H,H} | pas de données |
| **« non_data- »** | M_ND- ou {-} | {L,L} | pas de données |




**Légende**

| Anglais | Français |
|---|---|
| transmitter off | émetteur désactivé |
| one bit time | un temps binaire |
| Manchester biphase L encoding | codage biphasé L Manchester |
| time | temps |

**Figure 6 – M_symbols et codage Manchester à 5 MHz**

### 4.4.3 Interface DLL-PhL

#### 4.4.3.1 Généralités

L'interface DLL-PhL peut ne pas être exposée dans la mise en œuvre d'une variante PhL. Cette interface peut être propre au nœud. Si la conformité à l'interface DLL-PhL est demandée, elle doit être conforme aux exigences de 4.4.3.

#### 4.4.3.2 Ph_lock_indication

ph_lock_indication doit fournir une indication du verrouillage de données ou de la synchronisation Ph_Symbol réalisée par PhL. Les états valides de ph_lock_indication doivent être « true » et « false ». La valeur de ph_lock_indication doit être « true » en présence de Ph_Symbols valides dans l'interface PhL et lorsque la temporisation de l'interface PhL-DLL de M_Symbols est conforme aux exigences de PhL. Elle doit être « false » entre les trames (lorsque le support ne contient pas de Ph_Symbols), en cas de perte de la synchronisation des données ou lorsque la temporisation n'est pas conforme aux exigences de PhL. La valeur de ph_lock_indication doit être « true » avant de lancer le délimiteur de début.

#### 4.4.3.3 Ph_frame_indication

ph_frame_indication doit fournir une indication d'une trame de données valide provenant de PhL. Les valeurs d'états valides de ph_frame_indication doivent être « true » et « false ». La valeur de ph_frame_indication doit être « true » lorsque ph_lock_indication = true et lors de la réception du premier délimiteur de début valide. La valeur de ph_frame_indication doit être « false » à la réception du symbole M_ND suivant (après le délimiteur de début) ou lorsque ph_lock_indication = false.

NOTE Ce signal fournit une synchronisation d'octet à la DLL.

#### 4.4.3.4 Ph_carrier_indication

ph_carrier_indication doit indiquer la présence d'une porteuse de signal sur le support. La valeur de ph_carrier_indication doit être « true » si la valeur d'identification de PhL interne de la porteuse de signal était « true » au cours de l'une des 4 dernières temporisations de M_symbol. Sa valeur doit être « false » dans le cas contraire.

#### 4.4.3.5 Ph_data_indication

ph_data_indication doit représenter les M_Symbols décodés à partir des représentations internes de PhL (voir Tableau 5, par exemple). Les M_symbols valides doivent être M_0 {0}, M_1 {1}, M_ND+ {+} et M_ND- {-} (voir Tableau 6).

**Tableau 6 – Symboles de données M**

| Bits de données (nom commun) | Représentation M_Symbol |
|---|---|
| **données « zéro »** | M_0 ou {0} |
| **données « un »** | M_1 ou {1} |
| **« non_data+ »** | M_ND+ ou {+} |
| **« non_data- »** | M_ND- ou {-} |

ph_data_indication doit représenter les M_Symbols décodés dans la PhL à chaque fois que la valeur de ph_lock_indication est « true ».

#### 4.4.3.6 Ph_status_indication

ph_status_indication doit représenter l'état du délimiteur de la trame reçue de PhL (voir Tableau 7). Les symboles valides doivent être Normal, Abort et Invalid. ph_status_indication doit indiquer Normal suite à la réception d'une trame (ph_frame_indication = true) composée d'un délimiteur de début, de données valides (pas de symbole M_ND) et d'un délimiteur de fin. ph_status_indication doit indiquer Abort suite à la réception d'une trame (ph_frame_indication = true) composée d'un délimiteur de début, de données valides et d'un deuxième délimiteur de début. ph_status_indication doit indiquer Invalid suite à la réception d'une trame (ph_frame_indication = true) composée d'un délimiteur de début et de la détection d'un symbole M_ND ne faisant pas partie d'un délimiteur de début ou de fin.

**Tableau 7 – Tableau de vérité de ph_status_indication**

| Ph_status_indication | Ph_frame_indication | Délimiteurs de début d'une seule trame | Détection de délimiteur de fin | Violations de code Manchester sans délimiteur |
|---|---|---|---|---|
| **Normal** | vrai | 1 | vrai | faux |
| **Abort** | vrai | 2 | indifférent | faux |
| **Invalid** | vrai | 1 | indifférent | vrai |

#### 4.4.3.7 Ph_data_request

ph_data_request doit présenter le M_Symbols à transmettre. Les symboles valides doivent être M_0, M_1, M_ND+ ou M_ND- (voir Tableau 6). ph_data_request doit indiquer M_0 si aucune donnée ne doit être transmise (et si ph_frame_request = false).

#### 4.4.3.8 Ph_frame_request

La valeur de ph_frame_request doit être « true » si ph_data_request présente les M_Symbols à coder dans les Ph_Symbols appropriés par la PhL. Elle doit être « false » si aucun M_Symbols valide ne doit être transféré vers PhL.

### 4.5 Classes fonctionnelles

Les mises en œuvre de ce protocole appartiennent à deux grandes classes:

– auteurs de la connexion (parfois appelés lecteurs laser);
– répondeurs de connexion ou cibles de connexion (parfois appelés Adaptateurs).

Les appareils dotés de fonctions d'émission de connexion contiendront des fonctions telles que les objets de planification, le support des outils de configuration externes et l'objet Keeper.

En général, les répondeurs de connexion sont des appareils plus simples qui n'ont pas besoin de toutes ces capacités.

La présente spécification ne préconise pas un groupement particulier de fonctions pour les classes d'appareils de mise en œuvre.

NOTE Le groupement de services et de fonctions en classes de mise en œuvre peut être assuré par des spécifications de profils distinctes.

## 5 Structure générale et codage des PhIDU et DLPDU et éléments de mode opératoire connexes

### 5.1 Présentation

La DLL et ses modes opératoires sont nécessaires pour fournir les services proposés à l'utilisateur DLS à l'aide des services disponibles auprès de PhL. Les principaux services de ce protocole sont

a) disposition d'opportunités d'accès opportun aux services DL de transfert de données planifiés du support sur un lien et sur un réseau étendu à liaisons multiples;
b) la compression de plusieurs transferts d'utilisateur DLS en une seule unité de transfert PhL (DLPDU) afin d'utiliser efficacement chaque opportunité de transfert;
c) la répartition efficace des données d'utilisateur DLS entre un utilisateur DLS de publication et un ensemble d'utilisateurs DLS d'abonnement;
d) la disposition d'un sens synchronisé d'horloge interne parmi les DLE, disponible aux utilisateurs DLS de la liaison étendue;
e) la disponibilité normalisée des capacités de gestion de l'adresse DL locale, de la mémoire tampon et de la file d'attente.

### 5.2 Mode opératoire d'accès au support

La couche de liaison de données a pour principal objet de déterminer, conjointement avec d'autres couches de la même liaison, les autorisations de transmission sur le support. Au niveau de son interface, la DLL offre des services de réception et de fourniture d'unités de données pour l'utilisateur DLS et la gestion de station.

Le protocole DLL repose sur un cycle de durée fixe et répétitif appelé durée de mise à jour du réseau (NUT). La NUT est maintenue dans un synchronisme proche de tous les nœuds de la liaison. Un nœud n'est pas autorisé à transmettre sur le support si la NUT n'en convient pas avec celle utilisée au niveau d'une liaison. Plusieurs liaisons peuvent présenter des durées de mise à jour du réseau différentes.

Chaque nœud contient son temporisateur synchronisé avec la NUT de la liaison locale. L'accès au support est déterminé par une subdivision locale de la NUT en emplacements d'accès. L'accès au support s'effectue dans l'ordre séquentiel en fonction du MAC ID du nœud. Des comportements particuliers ont été intégrés dans le protocole d'accès, permettant à un nœud prenant provisoirement en compte un MAC ID nul de procéder à des opérations de maintenance sur la liaison. Les numéros MAC ID de tous les nœuds d'une liaison sont uniques. Une DLL qui détecte la présence d'un MAC ID en double doit immédiatement arrêter la transmission.

Un mécanisme de transmission de jeton implicite est utilisé pour autoriser l'accès au support. Chaque nœud surveille le MAC ID source de chaque DLPDU reçue. A la fin d'une DLPDU, chaque DLL attribue un « registre de jeton implicite » au MAC ID source reçu, incrémenté de 1. Si le registre de jeton implicite est égal au MAC ID local, le nœud doit transmettre une DLPDU contenant un ou plusieurs Lpackets avec des données ou une DLPDU nulle. Dans tous les autres cas, le nœud recherche une autre DLPDU dans le nœud identifié par le « registre de jeton implicite » ou une valeur de délai d'attente, si le nœud identifié ne peut assurer la transmission. Dans chaque cas, le « jeton implicite » est automatiquement avancé au MAC ID suivant. Tous les nœuds portent la même valeur dans leur « registre de jeton implicite », empêchant les collisions sur le support.

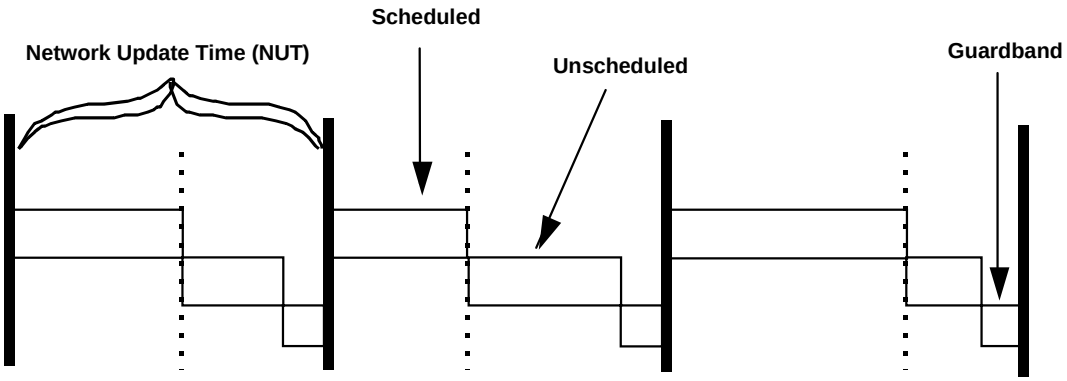
Le délai d'attente (appelé « intervalle ») est fonction de la durée requise pour que

– le nœud en cours entende la fin de la transmission du nœud précédent;
– le nœud en cours commence à transmettre;
– le nœud suivant entende le début de la transmission du nœud en cours.

L'intervalle est ajusté de manière à compenser la longueur totale du support, le délai de propagation du support ayant un impact sur le premier et le dernier élément de la liste précédente.

NOTE Le calcul de l'intervalle est spécifié en 8.2.

Chaque NUT est divisée en trois parties principales: planifiée, non planifiée et bande de garde (voir Figure 7). Cette séquence est répétée dans chaque NUT. Le mécanisme de transmission de jeton implicite permet d'autoriser l'accès au support pendant les intervalles planifiés et non planifiés.

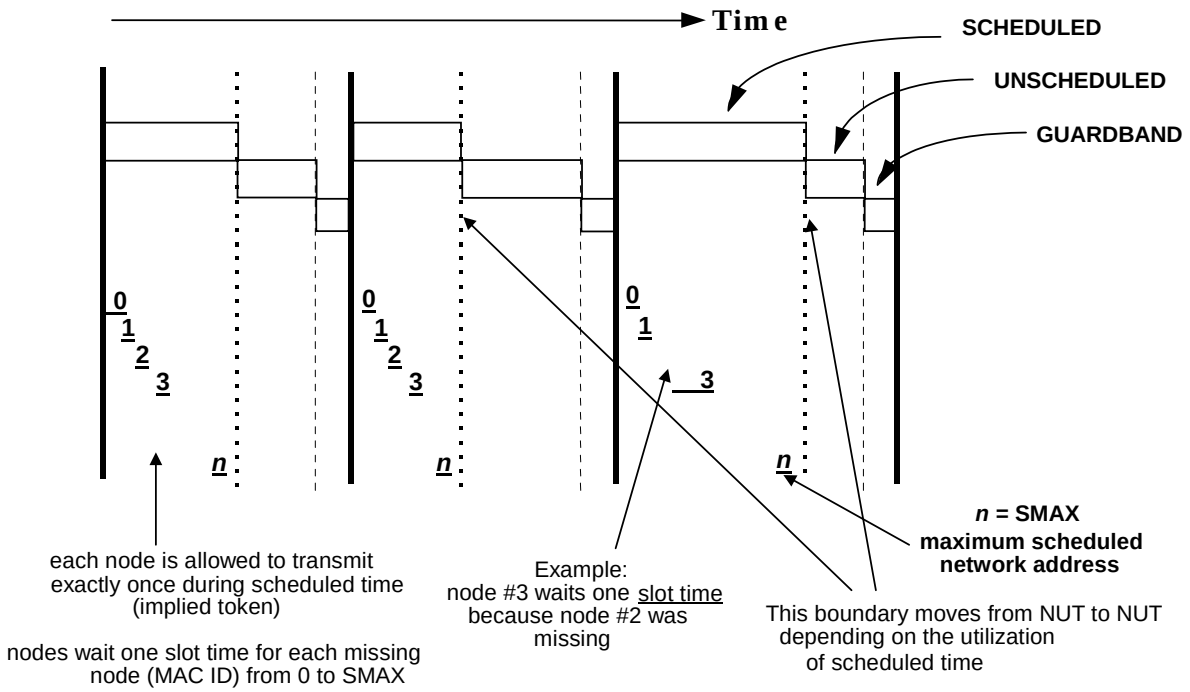



**Légende**

| Anglais | Français |
|---|---|
| Network update time (NUT) | durée de mise à jour du réseau (NUT) |
| scheduled | planifiée |
| unscheduled | non planifiée |
| guardband | bande de garde |

**Figure 7 – Structure de la NUT**

Au cours de la partie planifiée de la NUT, chaque nœud, en partant du nœud 0 et en finissant par le nœud SMAX, a une occasion de transmettre des données (planifiées) critiques du point de vue temporel. SMAX est le MAC ID du nœud le plus élevé ayant accès au support pendant la partie planifiée de la NUT. Chaque nœud entre 0 et SMAX ne dispose que d'une seule opportunité d'envoi d'une DLPDU de données planifiées dans chaque NUT. L'opportunité d'accès au support pendant la durée planifiée est la même pour tous les nœuds de chaque NUT. Cela permet d'envoyer les données transmises pendant la partie planifiée de la NUT de manière prévisible et déterministe. La Figure 8 illustre l'autorisation de transmission pendant la durée planifiée. L'utilisateur DLS ajuste la quantité de données que chaque nœud peut transmettre au cours de la transmission planifiée de jeton.

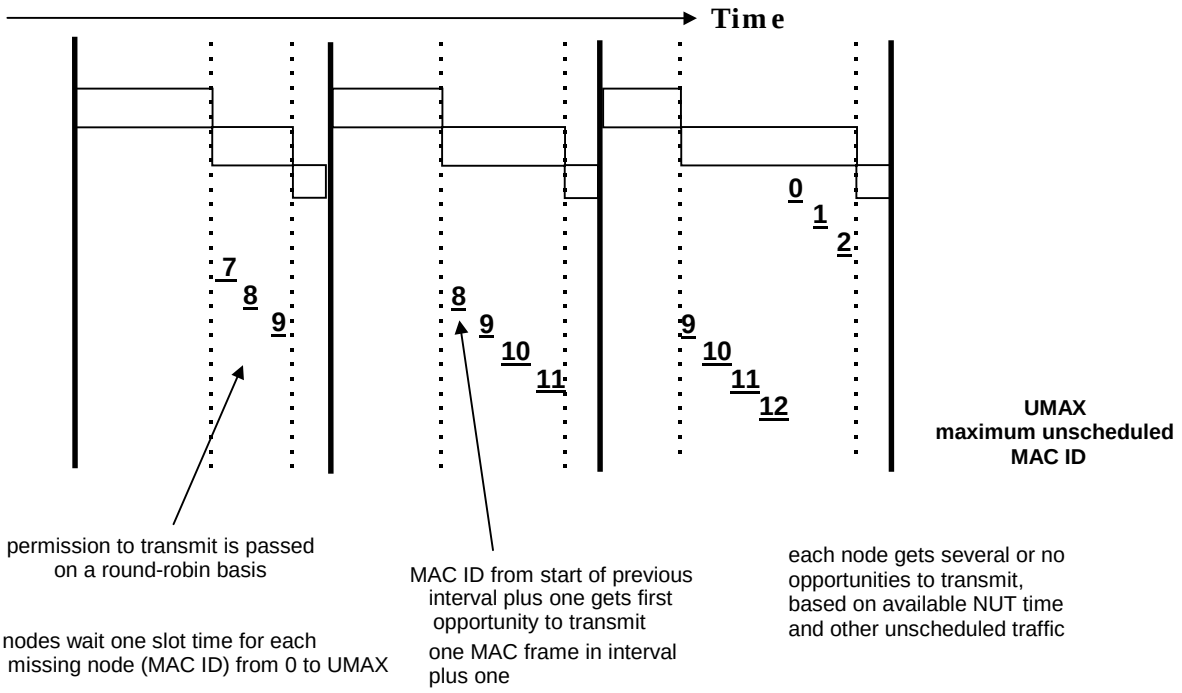



**Légende**

| Anglais | Français |
|---|---|
| Time | durée |
| scheduled | planifiée |
| unscheduled | non planifiée |
| guardband | bande de garde |
| maximum scheduled network address | adresse de réseau planifiée maximum |
| each node is allowed to transmit exactly once during scheduled time (implied token) | chaque nœud est autorisé à transmettre exactement une fois pendant la durée planifiée (jeton impliqué) |
| Example: node #3 waits one slot time because node #2 was missing | Exemple: le nœud 3 attend un intervalle de temps car le nœud 2 était manquant |
| this boundary moves from NUT to NUT depending on the utilization of scheduled time | cette limite se déplace d'une NUT à l'autre en fonction de l'utilisation de la durée planifiée |
| nodes wait one slot time for each missing node (MAC ID) from 0 to SMAX | les nœuds attendent un intervalle de temps pour chaque nœud manquant (MAC ID) entre 0 et SMAX |

**Figure 8 – Accès au support pendant la durée planifiée**

Au cours de la partie non planifiée de la NUT, tous les nœuds entre 0 et UMAX partagent l'opportunité de transmettre une DLPDU de données non critiques prioritaires de manière circulaire, tant que la durée NUT attribuée n'est pas épuisée. UMAX est le MAC ID du nœud le plus élevé ayant accès au support pendant la partie non planifiée de la NUT. La méthode circulaire d'opportunité d'accès permet à chaque nœud entre 0 et UMAX d'avoir zéro, une ou plusieurs opportunités d'envoyer des données non planifiées, selon la quantité de NUT restante à l'issue de la durée planifiée. Les variations du trafic planifié indiquent que l'opportunité d'accès au support pendant la durée non planifiée peut être différente pour chaque nœud d'une NUT. La Figure 9 illustre comment l'autorisation de transmission est accordée pendant la durée non planifiée. Le MAC ID du premier nœud de la partie non planifiée de la NUT est incrémenté de 1 pour chaque NUT. Le jeton non planifié commence par le MAC ID spécifié dans le registre de démarrage non planifié (USR) de la précédente DLPDU modérateur. L'USR incrémente chaque NUT de un modulo (UMAX+1). Si l'USR atteint UMAX avant la bande de garde, il revient à zéro et la transmission de jetons se poursuit.




**Légende**

| Anglais | Français |
|---|---|
| Time | durée |
| maximum unscheduled MAC ID | MAC ID non planifié maximum |
| permission to transmit is passed on a round-robin basis | l'autorisation de transmettre est donnée de manière circulaire |
| MAC ID from start of previous interval plus one gets first opportunity to transmit one MAC frame in interval plus one | Le MAC ID du début de l'intervalle précédent incrémenté de un obtient la première opportunité de transmettre une trame MAC dans l'intervalle plus un |
| each node gets several or no opportunities to transmit, based on available NUT time and other unscheduled traffic | chaque nœud obtient plusieurs ou aucune autorisation(s) de transmettre, sur la base de la durée NUT disponible et d'un autre trafic non planifié |
| nodes wait one slot time for each missing node (MAC ID) from 0 to UMAX | les nœuds attendent un intervalle de temps pour chaque nœud manquant (MAC ID) entre 0 et UMAX |

**Figure 9 – Accès au support pendant la durée non planifiée**

Lorsque la bande de garde est atteinte, tous les nœuds arrêtent de transmettre. Un nœud n'est pas autorisé à lancer une transmission à moins qu'elle puisse se terminer avant le début de la bande de garde. Au cours de la bande de garde, le nœud comportant le MAC ID le plus bas (appelé « modérateur ») transmet un message de maintenance (appelé « DLPDU modérateur ») qui

– conserve les temporisateurs NUT de tous les nœuds synchronisés;

– publie les paramètres de liaison critiques permettant à tous les membres du groupe DLL local de partager une version commune des valeurs DLL importantes (NUT, intervalle, SMAX, UMAX, par exemple).

Le modérateur transmet la DLPDU modérateur, qui resynchronise tous les nœuds et redémarre la NUT. Suite à la réception d'une DLPDU modérateur valide, chaque nœud compare ses valeurs internes à celles transmises dans la DLPDU modérateur. Un nœud utilisant des paramètres de liaison qui ne correspondent pas se désactive lui-même. Si la DLPDU modérateur n'est pas entendue pendant 2 NUT consécutives, le nœud comportant le MAC ID le plus bas joue le rôle de modérateur et commence à transmettre la DLPDU modérateur dans la bande de garde de la troisième NUT. Un nœud modérateur qui remarque la présence d'un autre nœud en ligne transmettant avec un MAC ID inférieur au sien annule immédiatement son rôle de modérateur.

Les situations classiques susceptibles d'engendrer une interruption du protocole d'accès DLL sont les suivantes:

– bruit induit sur la liaison;
– faible qualité du câble ou du raccordement;
– connexion physique de deux liaisons alors que le réseau est en cours de fonctionnement.

En général, ce type d'interruption a pour conséquence d'amener les nœuds à contester le nœud qu'il convient de transmettre. Il s'agit alors d'une « non-occurrence » de réseau. Un autre problème se produit lorsque les nœuds n'admettent pas les mêmes paramètres de configuration de liaison. Un nœud qui conteste les paramètres de liaison transmis par le modérateur est dit « suspect » et arrête immédiatement la transmission. Le protocole d'accès DLL est conçu pour récupérer un nœud suspect et le ramener en ligne.

## 5.3 Structure et codage de DLPDU

### 5.3.1 Généralités

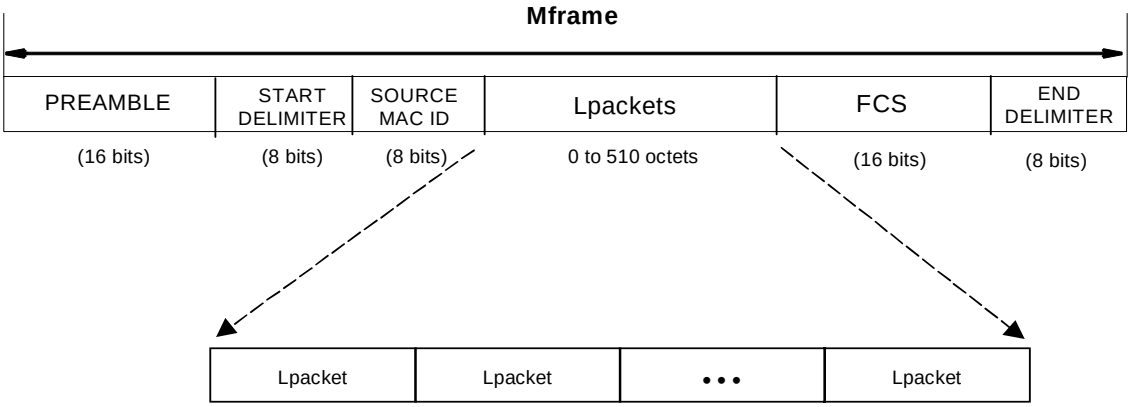
Les DLPDU sont des DLPDU transmises sur la couche Ph pour l'envoi sur le support Ph. Les DLPDU sont formées par le protocole TxLLC (contrôle de liaison logique de transmission) en deux étapes:

– Les DLSDU sont formées en unités de liaison (Lpackets);
– Plusieurs Lpackets sont compressés dans une DLPDU faisant l'objet d'un ensemble de contraintes liées au QoS requis, au type d'opportunité d'accès disponible et aux limites de longueur globale de DLPDU.

### 5.3.2 Composants DLPDU

La structure générale de la DLPDU est présentée dans la Figure 10 et gérée par les unités de support DLL TxFramer, RxFramer, Octet Serializer et Octet Deserializer (voir Figure 2).

Les composants de la DLPDU doivent être transmis dans l'ordre suivant: préambule, délimiteur de début, MAC ID source, zéro ou plusieurs Lpackets, FCS et délimiteur de fin.




**Légende**

| Anglais | Français |
|---|---|
| preamble | préambule |
| start delimiter | délimiteur de début |
| source MAC ID | MAC ID source |
| end delimiter | délimiteur de fin |
| ... to ... | ... à ... |

**Figure 10 – Format de DLPDU**

### 5.3.3 Préambule

Le préambule doit être composé de 16 {1} M_Symbols consécutifs.

### 5.3.4 Délimiteurs de début et de fin

Le délimiteur de début doit être composé des M_Symbols {+, 0, -, +, -, 1, 0, 1} transmis de gauche à droite.

Le délimiteur de fin doit être composé des M_Symbols {1, 0, 0, 1, +, -, +, -} transmis de gauche à droite.

L'utilisation des M_Symbols sans donnée doit être réservée pour les délimiteurs de début et de fin.

### 5.3.5 Octets DLPDU et ordre

Le MAC ID source, les Lpackets et la FCS doivent être composés d'octets. Un octet doit être composé d'exactement huit M_Symbols, chacun d'eux doit être {0} ou {1}. Un octet doit être d'abord transféré vers le bit inférieur PhL. Les champs multi-octets doivent être transmis, l'octet de poids faible l'étant en premier (format petit-boutiste).

### 5.3.6 MAC ID source

L'adresse du MAC ID source doit être composée d'un seul octet (voir 4.3.2). Un nœud peut également considérer provisoirement un MAC ID nul pour procéder à une maintenance de liaison.

### 5.3.7 Champ Lpackets

Aucun ou plusieurs Lpackets doivent être concaténés en une seule DLPDU faisant l'objet d'une contrainte liée au nombre maximal de 510 octets composant tous les Lpackets d'une DLPDU.

Une contrainte supplémentaire impose que les DLPDU envoyées au cours d'une opportunité d'accès planifiée ne doivent contenir que des Lpackets avec un paramètre QoS planifié.

### 5.3.8 Séquence de contrôle de trame (FCS)

#### 5.3.8.1 Exigences de séquence de contrôle de trame

La FCS doit être une somme de contrôle CCITT modifiée utilisant le polynôme de 16 bits: $G(X) = X^{16} + X^{12} + X^5 + 1$. Une FCS deux octets doit être incluse dans chaque DLPDU.

Les concepts formels de génération et de contrôle à la réception sont présentés en 5.3.8.2. La génération et le contrôle de la FCS sont présentés plus en détails en 9.7 et 9.8.

#### 5.3.8.2 Concepts formels de séquence de contrôle de trame

##### 5.3.8.2.1 Champ de séquence de contrôle de trame

Dans le Paragraphe 5.3.8.2, toute référence au bit *K* d'un octet fait référence au bit dont le poids dans un entier non signé d'un octet est $2^K$.

NOTE Parfois appelé numérotation de bit « petit-boutiste ».

Dans le cadre de la présente norme, comme dans les autres normes internationales (l'ISO/CEI 3309, l'ISO/CEI 8802 et l'ISO/CEI 9314-2, par exemple), la détection d'erreur au niveau de la DLPDU est assurée en calculant et ajoutant une séquence de contrôle de trame (FCS) à plusieurs bits aux autres champs DLPDU pendant la transmission. Il s'agit de former un « mot de code systématique »[3] de longueur ***n***, composé de ***k*** bits de message DLPDU suivis de ***n – k*** (égal à 16) bits redondants, puis de déterminer pendant la réception que le message et la FCS concaténés forment un mot de code (*n*,*k*) légal. Le mécanisme de contrôle est présenté ci-dessous, où la forme générique du polynôme générateur de cette construction FCS et le polynôme des données résiduelles du récepteur sont respectivement spécifiés dans les équations (4) et (9). Les polynômes spécifiques de la présente norme sont spécifiés dans le Tableau 8.

**Tableau 8 – Longueur, polynômes et constantes FCS**

| Elément | Valeur |
|---|---|
| *n-k* | 16 |
| G(X) | $X^{16} + X^{12} + X^5 + 1$ (Notes 1, |
| R(X) | $X^{12} + X^{11} + X^{10} + X^8 + X^3 + X^2 + X + 1$ (Notes 2, |

NOTE 1 Les mots de code D(X) construits à partir de ce polynôme G(X) comportent une distance 4 de Hamming pour les longueurs ≤ 4 095 octets et toutes les erreurs de poids paires sont détectées.

NOTE 2 Il s'agit des mêmes polynômes et méthodes que ceux spécifiés dans l'ISO/CEI 3309 (HDLC).

NOTE 3 Il convient que le R(x) restant soit 0001 1101 0000 1111 ($X^{15}$ à $X^0$, respectivement) en l'absence d'erreur.

[3] W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes* (2nd edition), MIT Press, Cambridge, 1972 (disponible en anglais uniquement).

#### 5.3.8.2.2 Au niveau du DLE émetteur

Le message d'origine (c'est-à-dire la DLPDU sans FCS), la FCS et le mot de code de message composite (la DLPDU et la FCS concaténées) doivent être considérés comme des vecteurs M(X), F(X) et D(X), de dimension *k*, *n* – *k* et *n*, respectivement, dans un champ d'extension sur GF(2). Si les bits de message sont $m_1$ ... $m_k$ et que les bits FCS sont $f_{n-k-1}$ ... $f_0$, où

| | |
|---|---|
| $m_1$ ... $m_8$ | forment le premier octet envoyé, |
| $m_{8N-7}$ ... $m_{8N}$ | forment le *N*ème octet envoyé, |
| $f_7$ ... $f_0$ | forment le dernier octet envoyé et |
| $m_1$ | est envoyé par le(s) premier(s) symbole(s) PhL du message et $f_0$ est envoyé par le(s) dernier(s) symbole(s) PhL du message (sans compter les informations de trame PhL), |

NOTE 1 Cet ordre « tel que transmis » est critique en fonction des propriétés de détection d'erreur de la FCS.

alors le vecteur de message M(X) doit être

$$M(X) = m_1X^{k-1} + m_2X^{k-2} + \ldots + m_{k-1}X^1 + m_k \quad (1)$$

et le vecteur FCS F(X) doit être

$$F(X) = f_{n-k-1}X^{n-k-1} + \ldots + f_0 = f_{15}X^{15} + \ldots + f_0 \quad (2)$$

Le vecteur composite D(X), de la DLPDU complète, doit être construit comme la concaténation du message et les vecteurs FCS

$$D(X) = M(X)\,X^{n-k} + F(X) = m_1X^{n-1} + m_2X^{n-2} + \ldots + m_kX^{n-k} + f_{n-k-1}X^{n-k-1} + \ldots + f_0 = m_1X^{n-1} + m_2X^{n-2} + \ldots + m_kX^{16} + f_{15}X^{15} + \ldots + f_0 \text{ (si k = 15)} \quad (3)$$

La DLPDU présentée à la PhL doit être composée d'une séquence d'octets dans l'ordre indiqué.

Les bits de contrôle redondants $f_{n-k-1}$ ... $f_0$ de la FCS doivent être les coefficients du reste F(X), après division par G(X), de L(X) $(X^k + 1)$ + M(X) $X^{n-k}$

où G(X) est le polynôme générateur de degré ***n-k*** des mots de code

$$G(X) = X^{n-k} + g_{n-k-1}X^{n-k-1} + \ldots + 1 \quad (4)$$

et L(X) est le polynôme de poids maximal (tous) de degré ***n-k-1***

$$L(X) = (X^{n-k} + 1)/(X + 1) = X^{n-k-1} + X^{n-k-2} + \ldots + X + 1 = X^{15} + X^{14} + X^{13} + X^{12} + \ldots + X^2 + X + 1 \text{ (si k = 15)} \quad (5)$$

En d'autres termes,

$$F(X) = L(X)\,(X^k + 1) + M(X)\,X^{n-k} \text{ (modulo G(X))} \quad (6)$$

NOTE 2 Les termes L(X) sont inclus dans le calcul afin de détecter la troncature ou l'extension initiale ou terminale de message en ajoutant un facteur dépendant de la longueur à la FCS.

NOTE 3 Comme mise en œuvre classique lorsque n-k = 16, le reste initial de la division est prédéfini sur chacun d'eux. Le train de bits de message transmis est multiplié par $X^{n-k}$ et divisé (modulo 2) par le polynôme générateur G(X), spécifié dans l'équation (7). Les compléments du reste résultant sont transmis comme FCS de (n-k) bits, avec le coefficient de $X^{n-k-1}$ transmis en premier.

L'équation de D(X) ne s'applique pas au protocole de Type 8, car ce protocole envoie le message et les bits de contrôle dans des DLPDU distinctes. L'équation de F(X) telle que donnée ne s'applique pas au protocole de Type 8, car ce protocole ne complète pas les bits de contrôle avant la transmission. L'équation correspondant au F(X) du protocole de Type 8 est

$$F_{Type\ 8}(X) = L(X)\ X^k + M(X)\ X^{n-k}\ (\text{modulo } G(X)) \quad (7)$$

NOTE 4 Comme mise en œuvre classique du protocole de Type 8, le reste initial de la division est prédéfini sur chacun d'eux. Le train de bits de message transmis est multiplié par $X^{n-k}$ et divisé (modulo 2) par le polynôme générateur G(X), spécifié dans l'équation (7). Le reste obtenu sans complément de uns est transmis en tant que FCS à (n-k) bits, le coefficient de $X^0$ étant transmis en premier.

#### 5.3.8.2.3 Au niveau du DLE destinataire

La séquence d'octets indiquée par PhE doit être concaténée dans la DLPDU et la FCS reçues, puis considérée comme un vecteur V(X) de dimension ***u***

$$V(X) = v_1X^{u-1} + v_2X^{u-2} + \ldots + v_{u-1}X + v_u \quad (8)$$

NOTE 1 En raison d'erreurs, **u** peut être différent de **n**, la dimension du vecteur de code transmis.

Un reste R(X) doit être calculé pour V(X), la DLPDU et la FCS reçues, par une méthode analogue à celle utilisée par le DLE émetteur (voir 0) dans le calcul de F(X)

$$R(X) = L(X)\ X^u + V(X)\ X^{n-k}\ (\text{modulo } G(X)) \quad (9)$$
$$= r_{n-k-1}X^{n-k-1} + \ldots + r_0$$

Définir E(X) comme le vecteur de code d'erreur des différences (modulo-2) supplémentaires entre le vecteur de code transmis D(X) et le vecteur reçu V(X) résultant d'erreurs rencontrées (dans le fournisseur PhS et dans les ponts) entre les DLE émetteurs et destinataires.

$$E(X) = D(X) + V(X) \quad (10)$$

Si aucune erreur ne s'est produite, de sorte que E(X) = 0, alors R(X) sera égal à un polynôme de reste d'une constante non nulle

$$R_{ok}(X) = L(X)\ X^{n-k}\ (\text{modulo } G(X)) \quad (11)$$

dont la valeur est indépendante de D(X). Malheureusement, R(X) sera aussi égal à $R_{ok}(X)$, lorsque E(X) est un multiple exact non nul de G(X), auquel cas des erreurs sont « indétectables ». Dans tous les autres cas, R(X) ne sera pas égal à $R_{ok}(X)$. Ces DLPDU sont erronées et doivent être écartées sans autre forme d'analyse.

NOTE 2 Comme mise en œuvre classique, le reste initial de la division est prédéfini sur chacun d'eux. Le train de bits reçu est multiplié par $X^{n-k}$ et divisé (modulo 2) par le polynôme générateur G(X), spécifié dans l'équation (8).

### 5.3.9 DLPDU nulle

Une DLPDU dont le champ Lpacket est vide, appelée DLPDU nulle, doit être envoyée par un nœud pour indiquer que l'opportunité d'accès n'admet aucun envoi de données au niveau du QoS.

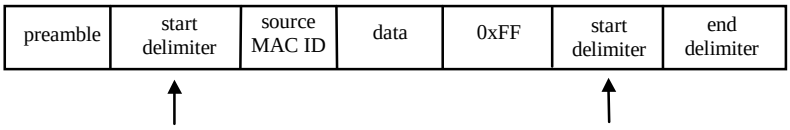
### 5.3.10 DLPDU annulée

Pour annuler une DLPDU partiellement transmise, la couche de liaison de données doit transmettre la séquence suivante:

– un octet 0xFF;

– le délimiteur de début;

– le délimiteur de fin




**Légende**

| Anglais | Français |
|---|---|
| preamble | préambule |
| start delimiter | délimiteur de début |
| source MAC ID | MAC ID source |
| data | données |
| end delimiter | délimiteur de fin |
| 2 start delimiters not separated by an end delimiter identify this as an aborted Mframe | 2 délimiteurs de début non séparés par un délimiteur de fin indiquent qu'il s'agit d'une Mframe annulée |

**Figure 11 – Annulation d'une DLPDU pendant la transmission**

NOTE 1 Certaines mises en œuvre de la présente spécification peuvent générer la DLPDU en cours de transmission. Si une couche supérieure n'a fourni qu'une DLPDU partielle à la couche de liaison de données et que la DLL a commencé la transmission de la DLPDU en supposant que le reste sera bientôt fourni, un dépassement négatif de capacité de transmission se produit si le reste de la DLPDU n'est pas fourni à la DLL suffisamment tôt pour terminer la transmission.

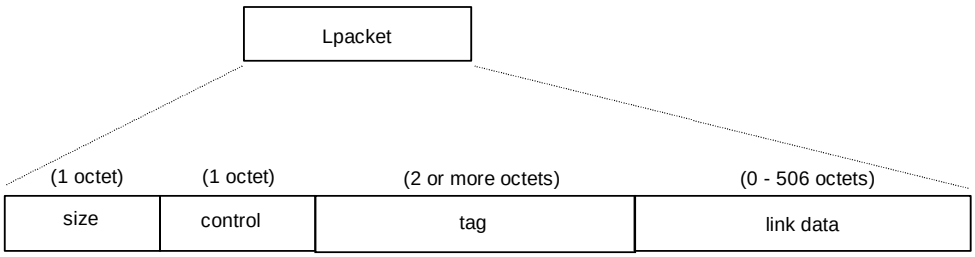
NOTE 2 Bien que la DLL assemble toute la DLPDU, y compris les délimiteurs, la PhL tente de détecter les délimiteurs de début et de fin. La PhL identifie une annulation de transmission comme une DLPDU contenant deux délimiteurs de début qui ne sont pas séparés par un délimiteur de fin (voir Figure 11).

NOTE 3 L'octet 0xFF est inséré avant la paire délimiteur de début/délimiteur de fin afin d'éviter d'enfreindre des contraintes PhL régissant les relations de symbole physique consécutif sur le support.

## 5.4 Composants Lpacket

### 5.4.1 Structure générale de Lpacket

Lpacket (ou unité de liaison) est le PDU utilisé pour acheminer les DLSDU entre les utilisateurs DLS homologues. Les composants du Lpacket doivent être transmis dans l'ordre suivant: taille, contrôle, balise et données de liaison (voir Figure 12).




**Légende**

| Anglais | Français |
|---|---|
| size | taille |
| control | contrôle |
| 2 or more octets | 2 octets ou plus |
| tag | balise |
| link data | données de liaison |

**Figure 12 – Format de Lpacket**

### 5.4.2 Taille

Le champ de taille doit préciser le nombre de paires d'octets (entre 3 et 255) que contient un Lpacket individuel. Le nombre de paires d'octets doit inclure les champs de taille, de contrôle, de balise et de données de liaison dénombrés conformément aux règles suivantes:

– lorsqu'ils sont combinés, la taille et le contrôle doivent compter comme une seule paire d'octets;

– le nombre de paires d'octets des champs de balise et de données de liaison doit être déterminé séparément, et le nombre d'octets pairs doit être arrondi;

– que les paires d'octets soient comptées de manière indépendante, le format d'un Lpacket placé dans une DLPDU ne doit pas inclure les octets supplémentaires générés par l'arrondi.

NOTE Par exemple, le champ de taille d'un Lpacket doté d'une balise de 3 octets (2 paires d'octets) et de données de liaison de 9 octets (5 paires d'octets) est de 1 + 2 + 5 = 8 paires d'octets. Toutefois, le Lpacket occupe 2 + 3 + 9 = 14 octets dans la DLPDU.

### 5.4.3 Contrôle

#### 5.4.3.1 Bit 0 et Bit 4 (type de Lpacket)

Les bits du champ de contrôle doivent être numérotés de 0 à 7, le bit 0 doit être le bit de poids faible. Le bit 0 doit indiquer le type de Lpacket.

– Lorsqu'une valeur lui est attribuée (bit 0 = 1), il doit s'agir d'un Lpacket à balise fixe (voir 4.3.4).

– Lorsqu'il est supprimé (bit 0 = 0), il doit s'agir d'un Lpacket à balise générique (voir 4.3.3).

Le bit 4 du champ de contrôle doit être l'inverse du bit 0. Si le bit 0 est supprimé, une valeur doit être attribuée au bit 4. Si une valeur est attribuée au bit 0, le bit 4 doit être annulé.

#### 5.4.3.2 Bit 1 (taille de balise paire)

Le bit 1 du champ de contrôle doit indiquer si le champ de balise contient un nombre d'octets pair ou impair. S'il est annulé (bit 1 = 0), ce bit doit indiquer que la balise contient un nombre d'octets pair. Si une valeur lui est attribuée (bit 1 = 1), il doit indiquer que la balise contient un nombre d'octets impair.

#### 5.4.3.3 Bit 2 (taille de données de liaison impaire)

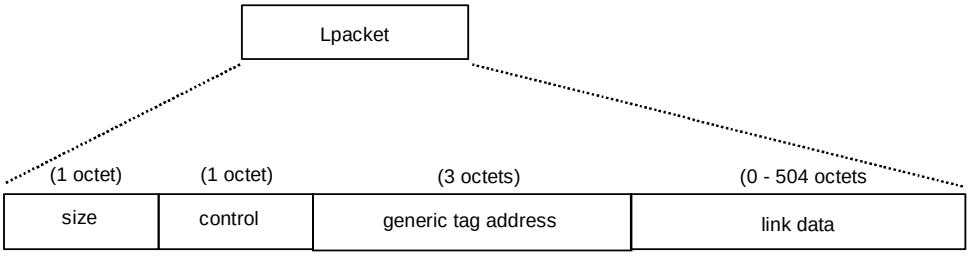
Le bit 2 du champ de contrôle doit indiquer si les données de liaison contiennent un nombre d'octets pair ou impair. S'il est annulé (bit 2 = 0), ce bit doit indiquer que les données de liaison contiennent un nombre d'octets pair. Si une valeur lui est attribuée (bit 2 = 1), il doit indiquer que les données de liaison contiennent un nombre d'octets impair.

#### 5.4.3.4 Bits 3, 5, 6 et 7 (bits réservés)

Les bits 3, 5, 6 et 7 du champ de contrôle sont réservés et doivent être nuls.

### 5.4.4 Lpackets à balise générique

Les unités de liaison génériques sont formées à partir de la balise générique DLS et des données utilisateur DLS fournies par l'utilisateur DLS, en plus des informations des champs de taille et de contrôle relatives aux balises génériques (voir Figure 13). L'adresse de balise générique à 3 octets (voir 4.3.3) doit identifier les informations de mode connexion acheminées dans le champ de données de liaison.




**Légende**

| Anglais | Français |
|---|---|
| size | taille |
| control | contrôle |
| generic tag address | adresse de balise générique |
| link data | données de liaison |

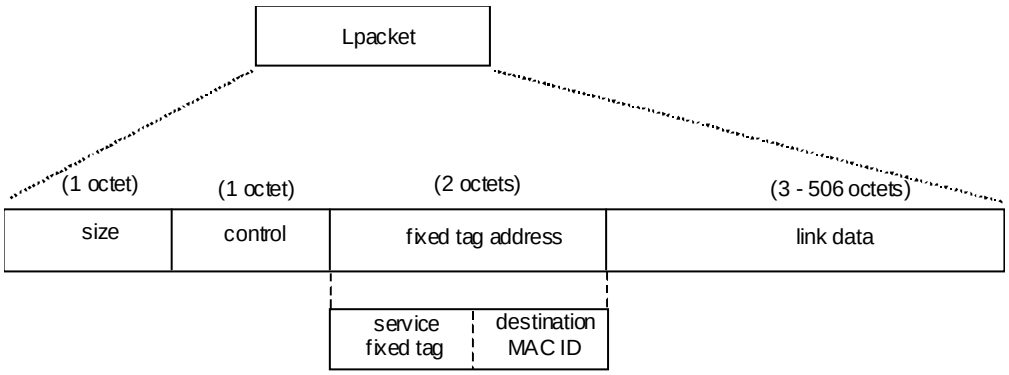
**Figure 13 – Format Lpacket à balise générique**

La longueur minimale du Lpacket à balise générique doit être de 5 octets (taille de données de liaison minimale de 0 octets). La taille maximale des données de liaison doit être de 504 octets.

NOTE En règle générale, les Lpackets génériques sont des transferts en mode connexion dont la valeur QoS est planifiée ou basse.

### 5.4.5 Lpackets à balise fixe

Les unités de liaison à balise fixe sont formées à partir de la balise fixe DLS et des données utilisateur DLS fournies par l'utilisateur DLS, en plus des informations des champs de taille et de contrôle relatives aux balises fixes (voir Figure 14). L'adresse de balise fixe à 2 octets (voir 4.3.4) doit identifier le point d'accès au service dans le nœud de destination et l'adresse de ce nœud.




**Légende**

| Anglais | Français |
|---|---|
| size | taille |
| control | contrôle |
| fixed tag address | adresse de balise fixe |
| link data | données de liaison |
| service fixed tag | balise fixe de service |
| destination MAC ID | MAC ID de destination |

**Figure 14 – Format Lpacket à balise fixe**

La longueur minimale du Lpacket à balise fixe doit être de 7 octets (taille de données de liaison minimale de 3 octets). La taille maximale des données de liaison doit être de 506 octets.

NOTE Les Lpackets fixes sont toujours des transferts en mode sans connexion. Ils ne sont habituellement pas envoyés avec la valeur du paramètre QoS planifiée.

## 5.5 Modes opératoires DLPDU

### 5.5.1 Généralités

Le protocole de transmission combine un ou plusieurs Lpackets en une seule DLPDU pour procéder à des envois en fonction de l'opportunité de transmission disponible. Les composants de DLPDU sont transmis à la couche Ph par TxM et le Convertisseur parallèle-série (voir 9.3).

Un nœud doit toujours envoyer une DLPDU à chaque opportunité d'accès valide pour qu'elle le transmette, et aucune DLPDU ne doit contenir des Lpackets avec un paramètre QoS inférieur à la valeur QoS applicable à l'opportunité d'accès.

Si un nœud ne contient pas de Lpackets dont la valeur est supérieure ou égale au QoS de l'opportunité d'accès, une DLPDU nulle doit être envoyée.

Si, pour des raisons internes, un nœud commence une DLPDU et n'est pas en mesure de la terminer, la séquence d'annulation doit être envoyée, la transmission doit s'arrêter pour cette opportunité d'accès et l'utilisateur DLS local doit en être informé.

NOTE Les DLPDU nulles et les DLPDU annulées informent les autres nœuds que l'utilisation de l'intervalle de transmission est terminée et autorisent le nœud suivant à commencer la transmission sans attendre le délai d'attente de l'intervalle.

Chaque Lpacket de la file d'attente de transfert comporte un identifiant local. A l'issue du transfert vers la PhL ou d'un transfert annulé, l'identifiant local est renvoyé à l'utilisateur DLS local avec une confirmation d'état OK ou TxAbort.

L'utilisateur DLS dispose de services locaux supplémentaires pour identifier les Lpackets non envoyés et demander leur éviction de la file d'attente, cette action renvoyant l'état de confirmation FLUSHED avec l'identifiant local des Lpackets évincés.

### 5.5.2 Envoi de DLPDU planifiées

En règle générale, les DLPDU planifiées permettent de transférer des opérations en mode connecté à l'aide de Lpackets génériques. Elles sont issues de la concaténation de plusieurs Lpackets (limités à 510 octets), commençant par des informations de contrôle Ph et un MAC ID de station source, et se terminant par une séquence de contrôle de trame et d'autres informations de contrôle Ph.

Les Lpackets sont compressés dans une DLPDU planifiée dans l'ordre FIFO, tels que l'utilisateur DLS les a envoyés, tant que la file d'attente contient encore des paquets avec un QoS planifié ou que la taille du Lpacket suivant ne dépasse pas l'espace restant.

### 5.5.3 Envoi de DLPDU non planifiées

Les DLPDU non planifiées peuvent contenir des Lpackets génériques et des Lpackets fixes. Elles sont issues de la concaténation de plusieurs Lpackets (limités à 510 octets), commençant par des informations de contrôle Ph et un MAC ID de station source, et se terminant par une séquence de contrôle de trame et d'autres informations de contrôle Ph.

Les Lpackets sont compressés dans une DLPDU non planifiée dans l'ordre de priorité QoS (priorités Planifiée, Elevée et Basse, respectivement). Dans chaque QoS, l'ordre FIFO de

mise en file d'attente par l'utilisateur DLS est respecté, tant que la file d'attente n'est pas vide ou que la taille du Lpacket suivant ne dépasse pas l'espace restant.

### 5.5.4 DLPDU destinataires

#### 5.5.4.1 Décompression

Les DLPDU reçues sont assemblées à partir du flux de symboles Ph entrant et décompressées par les fonctions Octet Deserializer, Receive Machine Framer (RxM) et Receive Logical Link Control (RxLLC) en deux étapes:

– une séquence de contrôle de trame est calculée sur le flux d'octets reçus (à l'exclusion des délimiteurs Ph) et la valeur résiduelle est vérifiée, si elle est correcte, puis

– chaque Lpacket que contient la DLPDU est examiné pour extraire sa balise. Chaque balise extraite est ensuite comparée au contenu d'un filtre de balise local pour déterminer si le Lpacket associé est pertinent pour la station destinataire. Les Lpackets dont les balises correspondent au filtre de balise font l'objet d'un traitement supplémentaire, puis sont transmis à l'utilisateur DL, au gestionnaire DL ou utilisés dans la DLL. Les indications transmises à l'utilisateur DL ou au gestionnaire DL contiennent la taille, la balise et les données du Lpacket. En outre, pour les Lpackets à balise fixe, l'ID source est également indiqué.

#### 5.5.4.2 Filtrage de balise

Chaque fournisseur DLS local gère une liste de balises permettant d'identifier les Lpackets pertinents pour son utilisateur DL ou ses fonctions de gestion DL internes. L'utilisateur DL peut accéder à cette liste et la modifier pour spécifier

– les balises génériques intéressantes

– les balises fixes intéressantes

NOTE Une balise générique spécifie les Lpackets d'un seul objet en mode connexion. Une balise fixe spécifie tous les messages en mode sans connexion d'une classe particulière (Répartition temporelle, par exemple).

## 5.6 Récapitulatif des services de support et objets DLL

Le fonctionnement et la coordination des fournisseurs DLL participants sont déterminés par des variables et des attributs placés dans des objets de gestion DL ou fournis par les services utilisateur DL figurant dans le Tableau 9. Ces éléments sont accessibles par l'intermédiaire d'adresses de balise fixe définies, de services utilisateur DL ou de services locaux non spécifiés. Les détails sont présentés dans les articles figurant dans la liste.

**Tableau 9 – Services de support et objets DLL**

| Elément DLL | Description | Emplacement |
|---|---|---|
| **Services utilisateur DLS** | | voir CEI 61158-3-2 |
| service de transfert en mode connecté | service local de demande, d'indication et de confirmation de transfert de données de balise générique (mode connecté) par les Lpackets | voir 6.2.2 |
| service de transfert en mode sans connexion | service local de demande, d'indication et de confirmation de transfert de données de balise générique (mode connecté) par les Lpackets | voir 6.2.2 |
| service de maintenance de la file d'attente | service local d'éviction des messages Lpacket non envoyés de la file d'attente de service DL interne | voir 6.2.3 |
| service de filtrage de balise | service local destiné à indiquer au fournisseur DL local de l'utilisateur DLS les balises Lpacket et les types de balise que la DLL doit accepter et traiter ou transmettre à l'utilisateur DLL | voir 6.2.4 |

| Elément DLL | Description | Emplacement |
|---|---|---|
| service de synchronisation de liaison | service local destiné à informer l'utilisateur DLS local du numéro NUT en cours pour maintenir les pointeurs planifiés | voir 6.2.5 |
| service de modification de paramètre synchronisé et compte à rebours de début tMinus | service local de chargement et de modification des listes locales de données de configuration DL en cours et en attente et, conjointement au service de compte à rebours de début tMinus et à la DLPDU modérateur, de gestion d'une modification précisément synchronisée aux nouvelles configurations de liaison DL | voir 6.2.6 |
| service de rapport d'événements<br>service de FCS erroné<br>modérateur en cours<br>mise sous tension et mise en ligne<br>activation du modérateur<br>écoute uniquement | services locaux rapportant les différents événements utilisés par le gestionnaire | voir 6.2.7<br>voir 6.2.8<br>voir 6.2.9<br>voir 6.2.10<br>voir 6.2.11<br>voir 6.2.12 |
| Service de balise générique | service Lpacket pour la communication en mode connecté | voir 6.3 |
| Services de balise fixe | ensemble d'adresses de service Lpacket pour différentes communications en mode sans connexion | voir 5.4.5 |
| modérateur | diffusion de Lpacket dans chaque NUT pour la gestion de la précision temporelle et la synchronisation de modification des paramètres de liaison | voir 6.4 |
| service de répartition temporelle | Lpacket de publications des valeurs et décalages temporels utilisés dans la coordination du même sens de la durée dans toutes les stations | voir 6.5 |
| UCMM | service Lpacket de transfert des DLSDU entre un utilisateur DLS et l'UCMM d'un utilisateur DLS homologue | voir 6.6 |
| Keeper UCMM | service Lpacket de transfert des DLSDU entre un utilisateur DLS et les objets Keeper d'autres nœuds | voir 6.7 |
| TUI | diffusion Lpacket à toutes les stations assistant leur synchronisation à la configuration de liaison en cours | voir 6.8 |
| paramètres de liaison | service Lpacket de publication des paramètres de liaison en attente avant une modification synchronisée des conditions de fonctionnement de liaison | voir 6.9 |
| tMinus | service Lpacket de démarrage d'un compte à rebours donnant lieu à une modification des paramètres de liaison et au renvoi du résultat. | voir 6.9 |
| I'm alive | diffusion Lpacket par de nouveaux appareils réinitialisés afin de faciliter leur entrée dans un système d'exploitation | voir 6.10 |
| demande et réponse ping | service Lpacket sollicitant des réponses d'identification de tous les autres appareils actifs de la liaison | voir 6.11 |
| WAMI | Lpacket permettant aux appareils temporaires d'identifier un MAC ID d'appareil local lorsqu'ils sont connectés à son port d'accès au réseau | voir 6.12 |
| débogage | service Lpacket de suivi des transitions d'état de l'objet | voir 6.13 |
| **Objets** | | |
| Objet ControlNet | détient et distribue des classes de données de gestion de station et des instances des couches Ph et DL, et collecte des informations de diagnostic à l'intention d'autres entités utilisateur DLS | voir 7.2 |

| Elément DLL | Description | Emplacement |
|---|---|---|
| Objet Keeper | accessible via Keeper UCMM ou la balise fixe UCMM générale, afin de fournir un ensemble de services de support pour les opérations de liaison, les planifications et les connexions<br><br>détient des données de liaison et des attributs pour tous les appareils d'une liaison<br><br>conserve des copies de toutes les données de l'auteur de la connexion pour les appareils à l'origine de la connexion d'un réseau<br><br>négocie avec d'autres objets Keeper d'une liaison afin de convenir du Keeper maître et des rôles du Keeper de sauvegarde<br><br>lorsque le Keeper maître distribue des attributs de liaison à tous les nœuds et maintient la synchronisation des Keepers de sauvegarde<br><br>émet régulièrement TUI Lpacket pour s'assurer que toutes les stations sont synchronisées à la configuration de liaison en cours<br><br>gère le sémaphore de ressource réseau permettant d'activer la modification régulière des données d'attribut de liaison | voir 7.3 |
| Objet de planification | offre des services au logiciel de planification de liaison externe en support à la lecture/écriture des informations de connexion et de planification de la part des appareils concernés<br><br>stocke les détails de la planification en cours et le mot de passe de connexion dans Keeper | voir 7.4 |
| Objet d'interface TCP/IP | fournit le mécanisme de configuration de l'interface TCP/IP d'un appareil | voir 7.5 |
| Objet de liaison Ethernet | gère les compteurs spécifiques à la liaison et les informations d'état d'un port Ethernet physique ISO/CEI 8802-3 | voir 7.6 |
| Objet DeviceNet | détient et distribue des classes de données de gestion de station et des instances des couches Ph et DL, et collecte des informations de diagnostic à l'intention d'autres entités utilisateur DLS | voir 7.7 |
| Objet de configuration de connexion | offre une interface de création, de configuration et de contrôle des connexions d'un appareil | voir 7.8 |

# 6 Structure DLPDU spécifique, codage et modes opératoires

## 6.1 Langage de modélisation

### 6.1.1 Description du diagramme d'états

Certains composants de la couche de liaison de données sont spécifiés à l'aide d'un langage de description de diagramme d'états. L'énoncé d'action est exprimé en langage C++. Les autres composants ne requièrent pas de description de diagramme d'états et sont spécifiés en C++ uniquement.

Un diagramme d'états est décrit par la syntaxe ci-dessous, où * et {} indiquent respectivement la répétition d'un ou de plusieurs composants précédents et un composant facultatif:

```
state-machine:=
    <header-statements>
    state*
state:=
    "state:" <state-name>
       transition*
transition:=
    "event:" <event-expression>
    {"condition:" <condition-expression>}
    "destination:" <state-name>
    {"action:" <action-statements>}
```

Les noms d'état sont des identifiants C++ normaux, les expressions d'événement et de condition sont des expressions C++ et les déclarations d'en-tête et d'action sont des listes de déclarations C++.

Les déclarations d'en-tête incluent

– #include et #define;

– les déclarations de variable et d'interface;

– les définitions de sous-routine.

Une expression d'événement indique l'événement spécifié lorsque la valeur de l'expression est TRUE. Un événement doit déclencher une transition. Une expression de condition qualifie une transition. Une transition qualifiée se produit lorsque son événement a lieu si la condition est vraie au moment de l'événement. En général, les expressions d'événement sont FALSE à l'entrée dans un état. Il existe peu de cas d'expressions d'événement vraies (TRUE) à l'entrée dans un état. Dans ces cas, la transition a lieu immédiatement.

Les déclarations d'action sont exécutées lors du déclenchement de la transition.

Tous les déclencheurs d'événement et toutes les conditions de l'état en cours sont évalués en continu et simultanément tant qu'une transition donnée n'est pas activée. Toutes les expressions et déclarations s'exécutent instantanément. Le modèle ci-dessous illustre la priorité des différentes parties d'une transition:

```
if(the event has occurred && the condition is true)
    {
    do the actions;
    go to the destination;
    }
```

Les mises en œuvre peuvent utiliser un autre code ou une autre syntaxe. Toutefois, les performances de la mise en œuvre doivent être équivalentes à celles spécifiées, sauf pour le déroulement d'exécution interne et les variables locales.

### 6.1.2 Utilisation du préfixe DLL

Les primitives et paramètres décrits dans la CEI 61158-3-2 utilisent les préfixes suivants:

DL- pour liaison de données

DLS- pour service de liaison DL

DLMS- pour service de gestion DL

Dans ce type 2: spécification du protocole de liaison de données, le préfixe générique DLL- est utilisé comme un équivalent au préfixe de description du service approprié.

### 6.1.3 Types de données

Les types de données utilisés dans les variables de liaison de données, les paramètres, les diagrammes d'états et l'exemple de code sont spécifiés dans la CEI 61158-5-2.

La spécification des types de données correspond à la notation de la CEI 61131-3. Une liste des types de données élémentaires, de leur description correspondante et de certaines de leurs valeurs admises est présentée dans le Tableau 10. Des détails supplémentaires sont spécifiés dans la CEI 61158-6-2.

**Tableau 10 – Types de données élémentaires**

| Mot clé | Description | Plage | |
|---|---|---|---|
| | | Minimum | Maximum |
| **BOOL** | Booléen | | |
| **SINT** | Entier court | -128 | 127 |
| **INT** | Entier | -32 768 | 32 767 |
| **DINT** | Entier double | $-2^{31}$ | $2^{31}-1$ |
| **LINT** | Entier long | $-2^{63}$ | $2^{63}-1$ |
| **USINT** | Entier court non signé | 0 | 255 |
| **UINT** | Entier non signé | 0 | 65 535 |
| **UDINT** | Entier double non signé | 0 | $2^{32}-1$ |
| **ULINT** | Entier long non signé | 0 | $2^{64}-1$ |
| **REAL** | Virgule flottante | | |
| **LREAL** | Virgule flottante longue | | |
| **ITIME** | Durée (court) | | |
| **TIME** | Durée | | |
| **FTIME** | Durée (haute résolution) | | |
| **LTIME** | Durée (long) | | |
| **DATE** | Date uniquement | | |
| **TIME_OF_DAY ou TOD** | Heure de la journée | | |
| **DATE_AND_TIME ou DT** | Date et heure de la journée | | |
| **STRING** | chaîne de caractères (1 octet par caractère) | | |
| **STRING2** | chaîne de caractères (2 octets par caractère) | | |
| **STRINGN** | Chaîne de caractères (N octets par caractère) | | |
| **SHORT_STRING** | Chaîne de caractères (1 octet par caractère, indicateur de 1 octet) | | |
| **SWORD** | Chaîne de bits - 8 bits | | |
| **WORD** | Chaîne de bits – 16 bits | | |
| **DWORD** | Chaîne de bits – 32 bits | | |
| **LWORD** | Chaîne de bits – 64 bits | | |

## 6.2 Services utilisateur DLS

### 6.2.1 Généralités

Les services utilisateur DLL fournis à l'utilisateur DLS comportent les interfaces suivantes utilisées pour les besoins spécifiés. Les définitions de service associées et les diagrammes de séquence temporelle sont donnés dans la CEI 61158-3-2.

### 6.2.2 Service de transfert en mode connecté et sans connexion

#### 6.2.2.1 Codage du service

Les services de demande et de confirmation utilisés pour mettre en file d'attente l'envoi de Lpackets par la DLL doivent se présenter sous la forme ci-dessous.

a) Les transferts en mode connexion utilisent la demande et confirmation de balise générique.

```
DLL_xmit_generic_request(
      IDENTIFIER   id,
      USINT     Lpacket[],
      UINT      Lpacket_size,
      PRIORITY  priority,
      USINT     gentag[3] );
```

```
DLL_xmit_generic_confirm(
      IDENTIFIER   id,
      TXSTATUS  status );
```

b) Les transferts en mode sans connexion utilisent la demande et confirmation de balise fixe.

```
DLL_xmit_fixed_request(
      IDENTIFIER   id,
      USINT     Lpacket[],
      UINT      Lpacket_size,
      PRIORITY  priority,
      USINT     service,
      USINT     destID );
```

```
DLL_xmit_fixed_confirm(
      IDENTIFIER   id,
      TXSTATUS  status );
```

c) Les transferts en mode connexion utilisent l'indication de balise générique.

```
DLL_recv_generic_indication(
      USINT     Lpacket[]
      UINT      Lpacket_size,
      USINT     gentag[3] );
```

d) Les transferts en mode sans connexion utilisent l'indication de balise fixe.

```
DLL_recv_fixed_indication(
      USINT     Lpacket[],
      UINT      Lpacket_size,
      USINT     service,
      USINT     sourceID );
```

#### 6.2.2.2 Modes opératoires de demande et de confirmation

Le paramètre `Lpacket` doit contenir la DLSDU sous la forme d'une séquence ordonnée d'octets. Le paramètre `Lpacket_size` doit spécifier le nombre d'octets que contient la matrice `Lpacket`. Le paramètre `QoS priority` doit spécifier la file d'attente interne dans laquelle la demande doit être placée, et doit prendre la valeur `LOW`, `HIGH` ou `SCHEDULED`. Seuls les Lpackets de la file d'attente `SCHEDULED` peuvent être envoyés lors de la partie planifiée de la NUT. La partie non planifiée de la NUT doit permettre de transmettre les Lpackets à partir des files d'attente de priorité QoS `HIGH` et `LOW`, et peut être utilisée pour transmettre les Lpackets à partir de la file d'attente `SCHEDULED`. Les files d'attente doivent faire l'objet de priorités de sorte que, lors d'une opportunité de transmission non planifiée, la file d'attente `SCHEDULED` est vidée en premier, suivie de la file d'attente de priorité QoS `HIGH`, puis de la file d'attente de priorité QoS `LOW`.

L'IDENTIFIANT, `id`, doit corréler une confirmation à une demande correspondante. Le paramètre `status` doit renvoyer l'état de la tentative de transmission et prendre la valeur `OK`, `TXABORT` ou `FLUSHED`.

Le service `DLL_xmit_fixed_request` met en file d'attente un Lpacket avec une balise fixe à deux octets pour la transmission. Le paramètre `service` doit spécifier le premier octet du Lpacket à balise fixe. Le paramètre `destID` doit spécifier le deuxième octet, qui doit déterminer le MAC ID du nœud de destination.

Le paramètre restant, gentag, du service DLL_xmit_generic_request doit spécifier la balise générique sur laquelle est transmis le Lpacket.

NOTE 1 L'utilisation d'une ou de plusieurs files d'attente de priorités différentes est un choix de mise en œuvre interne.

NOTE 2 L'assemblage de la structure Lpacket associée et le mode opératoire pour l'envoi sur la liaison sont décrits dans leurs articles respectifs (voir 5.3 et 5.4).

#### 6.2.2.3 Modes opératoires d'indication

Ces indications doivent signaler la délivrance d'un Lpacket avec une FCS correcte. Seuls les Lpackets à balise fixe dont le service a été activé par un appel réussi à DLL_enable_fixed_request doivent générer un DLL_recv_fixed_indication. Seuls les Lpackets à balise générique dont la balise générique a été activée par un appel réussi à DLL_enable_generic_request doivent générer un DLL_recv_generic_indication.

La matrice d'octets, Lpacket, doit contenir la DLSDU sous la forme d'une séquence ordonnée d'octets. L'entier, Lpacket_size, doit spécifier le nombre d'octets que contient la matrice Lpacket.

### 6.2.3 Service de maintenance de la file d'attente

#### 6.2.3.1 Codage du service

Une fois placé en file d'attente, un Lpacket doit rester dans une file d'attente de transmission tant qu'il n'a pas été transmis ou retiré de la file d'attente. Les services de demande et de confirmation utilisés pour retirer un Lpacket de la file d'attente, avant sa transmission, doivent se présenter sous la forme:

```
DLL_flush-requests-by-QoS_request( PRIORITY priority );
DLL_flush-requests-by-QoS_confirm( PRIORITY priority );
DLL_flush_single_request( PRIORITY priority, IDENTIFIER id );
```

#### 6.2.3.2 Modes opératoires de demande et de confirmation

DLL_flush-requests-by-QoS_request doit annuler toutes les transmissions en attente selon la priorité QoS spécifiée. Chaque transmission en attente doit immédiatement prendre fin avec une confirmation de l'état FLUSHED. DLL_flush_single_request doit annuler un Lpacket particulier identifié par l'id utilisé pour transmettre le Lpacket. Ce service peut ne pas disposer d'une confirmation correspondante étant donné que la confirmation du Lpacket placé dans la file d'attente doit être utilisée à cette fin.

### 6.2.4 Service de filtrage de balise

#### 6.2.4.1 Codage du service

Par défaut, la couche de liaison de données doit uniquement traiter le Lpacket modérateur (balise fixe 0x00) et tous les autres Lpackets doivent être écartés. Les autres couches de protocole peuvent autoriser ou interdire la réception de Lpackets particuliers à l'aide des services ci-dessous;

```
DLL_enable_generic_request(
      IDENTIFIER   id,
      USINT     gentag[3] );

DLL_disable_generic_request(
      IDENTIFIER   id,
      USINT     gentag[3] );

DLL_enable_fixed_request(
      IDENTIFIER   id,
      USINT     service );

DLL_disable_fixed_request(
      IDENTIFIER   id,
      USINT     service );
```

```
DLL_enable_generic_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_disable_generic_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_enable_fixed_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_disable_fixed_confirm(
      IDENTIFIER   id,
      BOOL      result );
```

#### 6.2.4.2 Modes opératoires de demande et de confirmation

L'IDENTIFIANT, `id`, doit corréler une confirmation à une demande correspondante. Le paramètre, `result`, doit renvoyer la valeur TRUE si la demande a abouti et la valeur FALSE dans le cas contraire. Le paramètre `gentag` doit préciser la balise générique à acheminer vers la couche supérieure qui l'a activée. De même, le paramètre `service` doit préciser la balise fixe à acheminer vers la couche supérieure qui a l'activée.

NOTE Le service `DLL_enable_generic_request` peut renvoyer un résultat erroné si la mise en œuvre de la couche de liaison de données n'est pas en mesure de filtrer d'autres balises génériques.

### 6.2.5 Service de synchronisation de liaison

#### 6.2.5.1 Codage du service

L'indication du commencement d'une nouvelle durée de mise à jour du réseau (NUT) doit se présenter sous la forme ci-dessous:

```
DLL_tone_indication( USINT cycle );
```

#### 6.2.5.2 Modes opératoires d'indication

Le paramètre `cycle` doit renvoyer le compteur d'intervalle de la dernière DLPDU modérateur reçue.

NOTE Si une DLPDU modérateur est attendue mais n'arrive pas, le nœud local simule la réception de la DLPDU modérateur en fonction d'un temporisateur interne de l'ACM.

### 6.2.6 Service de modification de paramètre synchronisé

#### 6.2.6.1 Codage du service

Tous les nœuds d'une liaison doivent conserver deux exemplaires des paramètres de liaison: un exemplaire en cours et un exemplaire en attente. L'exemplaire en cours des paramètres de liaison doit être utilisé pour le fonctionnement à venir de la couche de liaison de données. L'exemplaire en attente doit être conservé pour permettre une modification synchronisée des paramètres de liaison. Les services de l'utilisateur DLL local qui lisent et écrivent ces paramètres de liaison doivent se présenter sous la forme:

```
DLL_set_pending_request(
      IDENTIFIER      id,
      DLL_config_data  params );

DLL_set_current_request(
      IDENTIFIER      id,
      DLL_config_data  params );

DLL_get_pending_request(
      IDENTIFIER   id );


DLL_get_current_request(
      IDENTIFIER   id );
```

```
DLL_set_pending_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_set_current_confirm(
      IDENTIFIER   id,
      BOOL      result );

DLL_get_pending_confirm(
      IDENTIFIER      id,
      DLL_config_data  params );

DLL_get_current_confirm(
      IDENTIFIER      id,
      DLL_config_data  params );
```

DLL_config_data doit contenir les paramètres de liaison et se présenter sous la forme:

```
class DLL_config_data
{
 public:
 USINT myaddr; // the MAC ID of this node
 UINT NUT_length; // the length of the NUT in 10 µs increments
 USINT smax; // highest MAC ID allowed to transmit scheduled
 USINT umax; // highest MAC ID allowed to transmit unscheduled
 USINT slotTime; // time allowed for line turnaround in 1 µs increments
 USINT blanking; // time to disable RX after DLPDU in 1,6 µs increments
 USINT gb_start; // 10 µs intervals from start of guardband to tone
 USINT gb_center; // 10 µs intervals from start of moderator to tone
 USINT modulus; // modulus of the interval counter
 USINT gb_prestart; // transmit cut-off, 10 µs intervals before tone
}; // may not transmit passed this limit
```

Les services de l'utilisateur DLL local qui demandent et confirment le début d'un compte à rebours tMinus doivent se présenter sous la forme:

```
DLL-tMinus-start-countdown-request(
      IDENTIFIER   id,
      USINT     start_count );
```

```
DLL-tMinus-start-countdown-confirm(
      IDENTIFIER   id,
      BOOL      result );
```

NOTE Le service de balise fixe tMinus (voir 6.9) est utilisé par le Keeper maître pour demander toutes les stations sur la liaison pour initier le compte à rebours de temps de passage. Il s'agit du mode opératoire de l'utilisateur DLL local pour demander et confirmer la participation locale.

A l'expiration du compte à rebours tMinus, chaque station participante transmet une indication locale à son objet ControlNet pour demander une modification des paramètres de liaison en cours en paramètres de liaison en attente.

```
DLL_tminus_zero_indication( );
```

#### 6.2.6.2 Modes opératoires de demande, de confirmation et d'indication

L'IDENTIFIANT, id, doit corréler une confirmation à une demande correspondante. Le paramètre, result, doit renvoyer la valeur TRUE si la demande a abouti et la valeur FALSE dans le cas contraire.

Le Lpacket modérateur contient un champ appelé tMinus qui doit être utilisé pour synchroniser la mise à jour des paramètres de liaison en cours. DLL-tMinus-start-countdown-request doit favoriser la participation d'un nœud au compte à rebours tMinus et, s'il s'agit d'un nœud modérateur, doit initialiser le champ tMinus du Lpacket modérateur. Le nœud modérateur doit décrémenter ce champ jusqu'à zéro avant de transmettre chaque modérateur. Lorsque le champ tMinus passe de 1 à 0, la DLL de chaque nœud participant

au compte à rebours doit générer un DLL_tminus_zero_indication en local et copier ses paramètres de liaison en attente dans son exemplaire en cours. Si le champ tMinus passe à 0 à partir d'une valeur autre que 1, le compte à rebours doit être annulé et aucun DLL_tminus_zero_indication ne doit être généré.

### 6.2.7 Service de rapport d'événements

L'indication permettant d'alerter l'entité de gestion de station de différents événements internes à la couche de liaison de données doit prendre la forme ci-dessous:

```
DLL_event_indication(
      DLL_event event,
      USINT     mac_id /*[optional]*/ );
```

Le DLL_event, event, doit être l'un des événements énumérés dans le Tableau 11. Le paramètre mac_id doit être utilisé si (event = DLL_EV_badFrame). Il doit indiquer le MAC ID source du nœud qui a transmis la mauvaise DLPDU.

NOTE La DLPDU ayant été endommagée, le MAC ID source pourrait être incorrect.

**Tableau 11 – Evénements DLL**

| Evénement DLL | Description |
|---|---|
| **DLL_EV_rxGoodFrame** | Une DLPDU correcte a été reçue. Elle doit inclure les DLPDU ne contenant aucune donnée (DLPDU nulles), mais doit exclure les modérateurs |
| **DLL_EV_txGoodFrame** | Une DLPDU correcte a été transmise. Elle doit inclure les DLPDU ne contenant aucune donnée (DLPDU nulles), mais doit exclure les modérateurs |
| **DLL_EV_badFrame** | Une DLPDU endommagée a été reçue. Le paramètre facultatif, qui est le MAC ID source apparent du nœud en cause, doit également être reporté |
| **DLL_EV_errA** | Une DLPDU incorrecte a été reçue sur le canal A du support physique ou une DLPDU correcte a été reçue sur le canal B et la valeur FALSE est toujours attribuée au paramètre ph_frame_indication du canal A |
| **DLL_EV_errB** | Une DLPDU incorrecte a été reçue sur le canal B du support physique ou une DLPDU correcte a été reçue sur le canal A et la valeur FALSE est toujours attribuée au paramètre ph_frame_indication du canal B |
| **DLL_EV_txAbort** | Une DLPDU transmise s'est terminée par une séquence d'annulation |
| **DLL_EV_NUT_overrun** | NUT n'est pas assez importante pour assurer tout le trafic planifié |
| **DLL_EV_dribble** | Les Lpackets planifiés n'ont pas été envoyés pendant la durée planifiée |
| **DLL_EV_nonconcurrence** | Un événement a été détecté et indique que ce nœud n'est pas synchronisé avec le protocole de contrôle d'accès |
| **DLL_EV_rxAbort** | Une DLPDU a été transmise et s'est terminée par une séquence d'annulation |
| **DLL_EV_lonely** | N'a pas entendu une DLPDU provenant d'un autre nœud sur la liaison pour 8 NUT |
| **DLL_EV_dupNode** | Un autre nœud de la liaison utilise le MAC ID de ce nœud |
| **DLL_EV_noisePulse** | ph_lock_indication prend la valeur TRUE, puis la valeur FALSE avant que ph_frame_indication prenne la valeur TRUE, mais ph_lock_indication n'a pas suffisamment pris la valeur TRUE pour indiquer une éventuelle DLPDU endommagée |
| **DLL_EV_collision** | La valeur de ph_frame_indication était TRUE lorsque ce nœud était sur le point d'être transmis |
| **DLL_EV_invalidModAddress** | Un modérateur a été reçu d'un autre nœud dont le MAC ID n'était pas le plus bas sur la liaison |
| **DLL_EV_rogue** | Une DLPDU modérateur a été reçue et ne correspond pas aux informations de configuration de liaison sur ce nœud |
| **DLL_EV_deafness** | Impossible d'écouter la DLPDU modérateur, même en présence d'un autre trafic de liaison |
| **DLL_EV_supernode** | Un modérateur a été reçu du MAC ID 0. |

### 6.2.8 Service de FCS erroné

L'indication permettant d'alerter l'entité de gestion de station qu'une DLPDU reçue contient une FCS non valide doit se présenter sous la forme:

```
DLL_badFCS_indication( CHANNEL channel );
```

Le paramètre channel doit indiquer l'entité PhL ayant fourni la DLPDU erronée. Ce paramètre doit être CHA ou CHB, correspondant respectivement à la PhL canal A et à la PhL canal B. Cette indication doit être fournie au maximum une fois par DLPDU erronée par canal.

### 6.2.9 Service du modérateur en cours

L'indication permettant de signaler à l'entité de gestion de station le nœud qui est actuellement le modérateur doit se présenter sous la forme:

```
DLL_currentMod_indication( USINT mac_ID );
```

Le paramètre mac_ID doit indiquer le MAC ID du dernier nœud ayant transmis une DLPDU modérateur valide.

### 6.2.10 Services de mise sous tension et de mise en ligne

#### 6.2.10.1 Codage du service

La demande et confirmation permettant de placer la couche de liaison de données en ligne doivent se présenter sous la forme:

```
DLL_online_request( BOOL online );          DLL_online_confirm( BOOL online );
```

L'indication qui spécifie que l'initialisation de la couche de liaison de données est terminée doit se présenter sous la forme:

```
DLL_powerup_indication( void );
```

#### 6.2.10.2 Modes opératoires de demande et de confirmation

A la mise sous tension, la DLL doit attendre que la valeur du paramètre online soit TRUE. Ensuite, la DLL doit commencer le processus de mise en ligne. Si la valeur du paramètre online est FALSE, la DLL doit être mise hors ligne à la fin de la NUT en cours. Hors ligne, la DLL ne doit pas transmettre et doit ignorer les activités de liaison.

### 6.2.11 Service d'activation du modérateur

#### 6.2.11.1 Codage du service

Les services de demande et de confirmation permettant et interdisant au nœud de jouer le rôle de modérateur doivent se présenter sous la forme:

```
DLL_enable_moderator_request( BOOL enable );
DLL_enable_moderator_confirm( BOOL enable );
```

#### 6.2.11.2 Modes opératoires de demande et de confirmation

Si la valeur du paramètre enable est TRUE, la DLL doit activer la transmission des DLPDU modérateurs. Si la valeur du paramètre enable est FALSE, la transmission des DLPDU modérateurs doit être désactivée.

NOTE Cette demande est utilisée par les entités de gestion de station pour qu'un nouveau nœud puisse joindre une liaison de fonctionnement sans la perturber. Le protocole de basculement du modérateur ne tolère pas le nœud doté du MAC ID le plus bas en supprimant les DLPDU modérateurs pendant une période étendue. Le

Moniteur de connexion au réseau (NAM) active de nouveau la transmission du modérateur à chaque fois qu'un autre appareil est détecté sur la liaison.

### 6.2.12 Service d'écoute uniquement

#### 6.2.12.1 Codage du service

Les services de demande et de confirmation permettant à un appareil de recevoir, mais pas de transmettre, doivent se présenter sous la forme:

```
DLL_listen_only_request( BOOL enable );
DLL_listen_only_confirm( BOOL enable );
```

#### 6.2.12.2 Modes opératoires de demande, de confirmation et d'indication

Si la valeur du paramètre `enable` est TRUE, la couche de liaison de données doit participer au protocole d'accès et transmettre des DLPDU. Si la valeur du paramètre `enable` est FALSE, la transmission des DLPDU doit être désactivée. Toutefois, le nœud doit toujours avoir la possibilité de recevoir des DLPDU.

## 6.3 Lpacket à balise générique

### 6.3.1 Généralités

Le Lpacket à balise générique permet de transférer des données de connexion. L'ID connexion ou la balise générique est un identifiant unique des ressources et paramètres négociés se trouvant à sa source, aux destinations ou en tout autre point de transit intermédiaire. Seule la balise générique est nécessaire à l'identification des données du message.

### 6.3.2 Structure du Lpacket à balise générique

Les champs de taille et de contrôle doivent se présenter au format de balise générique. L'adresse doit être la balise générique DLS fournie par le service de demande générique DL local. Le champ de liaison de données doit contenir les données utilisateur DLS fournies par la demande générique DL.

### 6.3.3 Envoi et réception du Lpacket à balise générique

La station d'envoi doit envoyer le Lpacket au niveau de la priorité QoS spécifiée par la demande générique DL.

L'état de chaque transmission de Lpacket à balise générique est confirmé à l'utilisateur DL par un renvoi de l'ID utilisateur DLS (descripteur ou identifiant interne) et du DLS Txstatus comportant l'une des valeurs ci-dessous.

a) "OK" — le message a été envoyé avec succès;

b) "TXABORT" — le processus d'envoi n'a pas abouti;

c) "FLUSHED" — le message a été retiré de la file d'attente avant l'envoi.

NOTE 1 Txstatus est une variable locale. Le codage n'est donc pas spécifié.

NOTE 2 L'état FLUSHED est uniquement utilisé suite à une suppression demandée par le service de maintenance de la file d'attente.

NOTE 3 La valeur de paramètre OK n'indique pas que le message a été reçu.

Toutes les stations destinataires dont le service de filtrage de balise local contient une adresse correspondant à l'adresse Lpacket reçue doivent transmettre chaque unité de données correctement reçue à leur utilisateur DL local, avec la taille et l'adresse de l'unité de données (la balise générique pour la connexion).

## 6.4 Lpacket modérateur

### 6.4.1 Généralités

Le Lpacket MAC modérateur est utilisé pour la gestion DLL, la temporisation d'accès et la synchronisation. Il est envoyé en tant que DLPDU modérateur ne contenant qu'un Lpacket: le Lpacket modérateur.

### 6.4.2 Structure du Lpacket modérateur

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe du modérateur et de 0xFF, l'adresse de diffusion de tous les nœuds MAC IC sur la liaison. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class moderatorLpacket: public fixedLpacket
{
 public:
 UINT NUT_length; // the length of the NUT in 10 µs increments
 USINT smax; // highest MAC ID allowed to transmit scheduled
 USINT umax; // highest MAC ID allowed to transmit unscheduled
 USINT slotTime; // 1 µs increments allowed for line turnaround
 USINT blanking; // 1,6 µs increments to disable RX after DLPDU
 USINT gb_start; // 10 µs intervals from start of guardband to tone
 USINT gb_center; // 10 µs intervals from start of moderator to tone
 USINT usr; // unscheduled start register
 USINT interval_count; // current NUT number (0 to modulus)
 USINT modulus; // modulus of the interval counter
 USINT tMinus; // countdown to synchronized link parameter change
 USINT gb_prestart; // transmit cut-off, 10 µs intervals before tone
 USINT reserved; //
};
```

### 6.4.3 Envoi et réception du Lpacket modérateur

Le nœud doté du MAC ID le plus bas doit transmettre la DLPDU modérateur contenant le Lpacket modérateur une fois par NUT. Il doit être envoyé dans la bande de garde, partie fixe de chaque NUT réservée au DLPDU modérateur.

Pour prendre en charge les actions de gestion DLL, l'émetteur et les récepteurs de la DLPDU modérateur doivent utiliser le contenu du champ de données de liaison du modérateur, comme suit.

`NUT_length` doit spécifier la durée de la NUT en cours. `NUT_length` doit être un entier de 16 bits représentant le nombre d'impulsions de 10 µs et doit être compris entre 50 (500 µs) et 10 000 (100 ms). Le début de la NUT doit être appelé « tonalité ». A la tonalité, un compteur qui décrémente toutes les 10 µs doit être chargé avec `NUT_length`.

NOTE 1 Le Paragraphe 8.2 contraint la valeur de `NUT_length`.

Pour garantir le calme de la liaison pendant la transmission du modérateur, tous les nœuds doivent cesser de transmettre lorsque le compteur interne atteint la valeur de `gb_prestart`. Tous les nœuds peuvent alors supposer que la liaison est calme entre `gb_start` et `gb_center`. Lorsque le compteur du nœud modérateur atteint la valeur de `gb_center`, il doit commencer à transmettre une DLPDU contenant uniquement le Lpacket modérateur. Le Lpacket modérateur doit être intégralement transmis au moins 30 µs avant l'expiration du compteur de décrémentation.

NOTE 2 Le Paragraphe 8.2 calcule les valeurs de `gb_prestart`, `gb_start` et `gb_center` pour garantir ces contraintes. Les facteurs ayant un impact sur le calcul de ces valeurs incluent la précision d'horloge, les nœuds

manquants et le délai de propagation entre les nœuds. Dans tous les cas, gb_prestart > gb_start > gb_center > 7.

Le MAC ID le plus élevé assurant la transmission pendant la partie planifiée de la NUT doit être smax. Il doit être compris entre 0 et 254. umax doit déterminer le MAC ID le plus élevé qui assure la transmission pendant la partie non planifiée de la NUT. Il doit être compris entre smax et 254. Le registre de démarrage non planifié, usr, doit sélectionner le nœud qui va transmettre en premier dans la partie non planifiée de la NUT. usr doit incrémenter chaque NUT. Ce compteur doit être un modulo mis à jour (umax + 1).

Un nœud doit écouter la PhL directement à l'issue de la transmission. blanking doit déterminer la durée de suspension d'écoute et doit être dans des impulsions de 1,6 µs.

Le nœud modérateur doit incrémenter un compteur de 8 bits, appelé interval_count, une fois par NUT. Ce compteur doit être un modulo mis à jour (modulus + 1). Etant donné que la bande de garde se trouve à la fin de la NUT, la valeur de interval_count doit correspondre à la NUT qui vient de se terminer.

NOTE 3 Le Paragraphe 8.2 et les spécifications de l'objet ControlNet contraignent la valeur de smax, umax, blanking et modulus.

Le champ tMinus doit maintenir un compte à rebours tant que tous les nœuds n'ont pas adopté de nouveaux paramètres de liaison (voir 6.9). Le champ slotTime doit déterminer le temps d'attente d'un nœud manquant et doit être dans des impulsions de 1,0 µs 1,0 µs supplémentaire doit être ajouté à la valeur de slotTime. Le champ slotTime doit être compris entre 8 et 254. Les autres valeurs doivent être réservées. Une valeur égale à zéro doit être attribuée à l'octet reserved.

## 6.5 Lpacket de répartition temporelle

### 6.5.1 Généralités

La DLPDU modérateur offre un marqueur de référence commun synchronisé entre tous les nœuds du réseau. En distribuant et traitant les datations en fonction de la référence au lieu du message de répartition temporelle, les mises en œuvre sont simplifiées tout en améliorant l'exactitude de plusieurs ordres de grandeur. La synchronisation phase-fréquence est inhérente à ce protocole DL à un niveau très élevé d'exactitude. L'exactitude de la synchronisation temporelle utilisant le format de répartition temporelle défini en 6.5 dépend de la mise en œuvre. Toutefois, elle peut être supérieure à 10 µs.

### 6.5.2 Structure du Lpacket de répartition temporelle

#### 6.5.2.1 Format du paquet

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe de répartition temporelle et de l'adresse de diffusion 0xFF. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué:

```
class TimeDist_Lpacket: public fixedLpacket
{
   public:
   USINT   revision; // revision of time distribution format
   USINT   leap; // leap second offset
   UINT    goodness; // time relay control field
   DINT    gse; // global squared error relative to ultimate master
   LINT    dctz; // distribution channel time zero
   LINT    ts_ref; // time stamp of previous reference pulse
   LINT    ts_tx; // time stamp of this message's transmission
};
```

#### 6.5.2.2 Revision

Une valeur égale à zéro doit être attribuée au paramètre revision. Ce paramètre doit représenter la révision du format de répartition temporelle.

#### 6.5.2.3 Leap

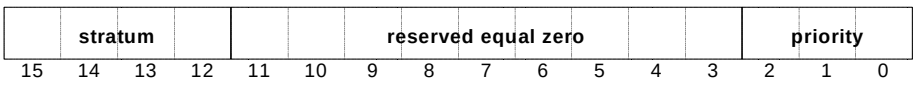
Le paramètre leap doit spécifier les secondes intercalaires UTC (Universal Coordinated Time). Ce nombre, lorsqu'il est ajouté au temps système, doit donner le temps UTC réel. Il peut augmenter ou diminuer de un à deux fois par an, aux solstices, comme indiqué par l'US Naval Observatory, afin de maintenir le synchronisme entre le temps terrestre et le temps astronomique. S'il est nul, le nombre de secondes intercalaires n'est alors pas connu.

Il convient de ne pas utiliser le paramètre leap dans des situations de contrôle. Cependant, il peut être nécessaire dans certains relais temporels de canaux de distribution reposant sur le temps UTC plutôt que GPS (Global Positioning Satellite).

#### 6.5.2.4 Goodness

##### 6.5.2.4.1 Structure de paramètre

Le paramètre goodness doit être partitionné comme indiqué dans la Figure 15.




**Légende**

| Anglais | Français |
|---|---|
| stratum | strate |
| reserved equal zero | réservés, valeur nulle |
| priority | priorité |

**Figure 15 – Paramètre goodness de TimeDist_Lpacket**

##### 6.5.2.4.2 Strate

Le champ le plus significatif, appelé strate, doit indiquer le nombre de relais temporels entre ce message et une source de temps absolu. Une valeur égale à 0 doit signifier une référence exacte, et la valeur doit être incrémentée pour tous les relais temporels intermédiaires. Si une valeur nulle est attribuée au champ de priorité (verrouillage non obtenu) ou si le nombre de relais temporels intermédiaires est supérieur à 15, la valeur 15 doit être attribuée au paramètre d'exactitude. Les bits 3 à 11 doivent être réservés et nuls.

NOTE Un relais temporel est un routeur liaison à liaison qui distribue les messages de synchronisation temporelle de sa liaison en fonction des messages de synchronisation temporelle de ses autres liaisons.

#### 6.5.2.4.3 Qualité de la source

Le champ de priorité doit indiquer la qualité du message temporel (voir Tableau 12).

**Tableau 12 – Priorité de répartition temporelle**

| Valeur | Signification |
|---|---|
| 7 | Temps système absolu. Fait office de maître |
| 6 | Temps système absolu. Fait office de subordonné |
| 5 | Temps système défini par l'homme. Fait office de maître |
| 4 | Temps système défini par l'homme. Fait office de subordonné |
| 3 | Verrou appliqué au nœud du canal de distribution autre que celui-là |
| 2 | Verrou appliqué au nœud de ce canal. Temps système inconnu |
| 1 | (non valide) |
| 0 | Non synchronisé avec un autre nœud |

### 6.5.2.5 gse

Le paramètre gse doit indiquer la stabilité rms cumulative au carré. Ce paramètre doit approximer la stabilité du cas le moins favorable du nœud par rapport au reste du système. Les unités de ces paramètres doivent être 0,1 μs)$^2$. Si la stabilité rms est inconnue ou pas encore déterminée, elle doit être de 0xFFFFFFFF.

### 6.5.2.6 Paramètres temporels LINT

Dans chaque paramètre temporel LINT, le bit de poids fort doit être nul (uniquement 63 bits doivent être utilisés). Les unités de ces paramètres doivent être 0,1 μs.

### 6.5.2.7 dctz

Le paramètre dctz doit indiquer le décalage de temps système du temps zéro arbitraire du canal de distribution, établi lors de la synchronisation des réseaux.

### 6.5.2.8 ts_ref

Le paramètre ts_ref doit indiquer la datation de la dernière tonalité suivant une DLPDU modérateur dont l'intervalle interval_count est nul. La valeur nulle doit indiquer que cette valeur est inconnue. Le temps zéro du système doit être défini comme celui étant utilisé pour les systèmes GPS: 6 janvier 1980 12:00 minuit GMT.

### 6.5.2.9 ts_tx

Le paramètre ts_tx doit indiquer la datation au niveau de la transmission de ce message. La valeur nulle doit indiquer que cette valeur est inconnue. Le temps zéro du système doit être défini comme celui étant utilisé pour les systèmes GPS: 6 janvier 1980 12:00 minuit GMT. Une valeur nulle doit être attribuée à ce paramètre.

NOTE Ce protocole n'utilise pas le temps GPS, mains uniquement le temps zéro GPS. Par conséquent, 1999 GPS n'a aucun intérêt pour le temps système.

### 6.5.3 Envoi et réception du Lpacket de répartition temporelle

Le Keeper maître en cours est chargé d'envoyer le Lpacket de répartition temporelle à des intervalles et des valeurs QoS configurés par l'utilisateur DL en fonction des exigences de son application.

Tous les nœuds sont chargés de recevoir le Lpacket de répartition temporelle.

## 6.6 Lpacket UCMM

### 6.6.1 Généralités

L'UCMM (Unconnected Message Manager) est une entité utilisateur DLS offrant un service de message de demande/réponse sans connexion pour au moins un message en attente. Elle utilise le service de transmission DLL et le Lpacket UCMM pour ses messages.

### 6.6.2 Structure du Lpacket UCMM

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe UCMM et du MAC ID de destination fourni par le service de transmission DLL. Le champ de liaison de données doit contenir les données utilisateur DLS fournies par le service de transmission DLL.

### 6.6.3 Envoi et réception du Lpacket UCMM

Le Lpacket UCMM doit être envoyé à la priorité QoS spécifiée par le service de transfert de données DLL.

L'état de chaque transmission UCMM est confirmé à l'utilisateur DL par un renvoi de l'ID utilisateur DLS (descripteur ou identifiant interne) et du DLS Txstatus comportant l'une des valeurs ci-dessous.

a) "OK" — le message a été envoyé avec succès;

b) "TXABORT" — le processus d'envoi n'a pas abouti;

c) "FLUSHED" — le message a été retiré de la file d'attente avant l'envoi.

NOTE 1 Txstatus est une variable locale. Le codage n'est donc pas spécifié.

NOTE 2 L'état FLUSHED est uniquement utilisé suite à une suppression demandée par le service de maintenance de la file d'attente.

NOTE 3 La valeur de paramètre OK n'indique pas que le message a été reçu.

La station destinataire dont le MAC ID correspond à celui de l'adresse doit transmettre chaque unité de données correctement reçue à son utilisateur DL local, avec la taille de l'unité de données, le type de service Lpacket et son ID source obtenu auprès de la DLPDU qui a acheminé le Lpacket.

## 6.7 Lpacket Keeper UCMM

### 6.7.1 Généralités

Les appareils qui mettent en œuvre l'objet Keeper doivent également prendre en charge l'envoi et la réception des Lpackets Keeper UCMM. Seul l'objet Keeper maître doit envoyer le Lpacket Keeper UCMM.

NOTE L'utilisation d'une adresse spécifique pour les objets Keeper permet de réduire l'activité de décodage requise par les appareils qui ne contiennent pas d'objet Keeper.

### 6.7.2 Structure du Lpacket Keeper UCMM

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe Keeper UCMM et de l'adresse de diffusion 0xFF. Le champ de liaison de données doit contenir les données utilisateur fournies par l'objet Keeper.

### 6.7.3 Envoi et réception du Lpacket Keeper UCMM

Le Lpacket Keeper UCMM doit uniquement être envoyé par une station dont l'objet Keeper est le maître. Il doit être demandé à la priorité QoS spécifiée par l'objet Keeper appelant.

L'état de chaque transmission Keeper UCMM est confirmé à l'utilisateur DL par un renvoi de l'ID utilisateur DLS (descripteur ou identifiant interne) et du DLS Txstatus comportant l'une des valeurs ci-dessous.

a) "OK" — le message a été envoyé avec succès;

b) "TXABORT" — le processus d'envoi n'a pas abouti;

c) "FLUSHED" — le message a été retiré de la file d'attente avant l'envoi.

NOTE 1 Txstatus est une variable locale. Le codage n'est donc pas spécifié.

NOTE 2 L'état FLUSHED est uniquement utilisé suite à une suppression demandée par le service de maintenance de la file d'attente.

NOTE 3 La valeur de paramètre OK n'indique pas que le message a été reçu.

La station destinataire dont le MAC ID correspond à celui de l'adresse doit transmettre chaque unité de données correctement reçue à son utilisateur DL local, avec la taille de l'unité de données, le type de service Lpacket et son ID source obtenu auprès de la DLPDU qui a acheminé le Lpacket.

Etant donné qu'il s'agit d'un message sans connexion, le MAC ID source est requis pour activer une réponse appropriée.

## 6.8 Lpacket TUI

### 6.8.1 Généralités

Un Identifiant unique de table (TUI) est utilisé par tous les objets ControlNet pour faire référence à un ensemble cohérent d'attributs de configuration de liaison. Deux ensembles distincts d'attributs sont traités, chacun doté de leur propre TUI: l'un pour le fonctionnement en cours et l'autre pour le fonctionnement en attente. Le passage des données TUI de la configuration de liaison en cours vers celles de la configuration de liaison en attente doit avoir lieu à l'issue d'une modification de paramètre synchronisé.

Le Lpacket TUI est publié comme un Lpacket planifié par le Keeper maître pour activer tous les appareils connexes afin de s'assurer qu'ils comportent des données de configuration et de planification en cours.

### 6.8.2 Structure du Lpacket TUI

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe TUI et de l'adresse de diffusion 0xFF. Le champ de données de liaison doit contenir les éléments suivants du Tableau 13.

**Tableau 13 – Format du Lpacket TUI**

| Nom | Type de données | Description du paramètre | Sémantique des valeurs |
|---|---|---|---|
| **Taille** | USINT | Taille du Lpacket, en mots | 0x0D |
| **Contrôle** | USINT | Octet de contrôle du Lpacket de couche de liaison | 0x01 (Lpacket à balise fixe) |
| **Fixed_Tag** | USINT | Valeur de balise fixe | 0x84 (Lpacket TUI) |
| **Destination_Mac_Id** | USINT | Destinataire du Lpacket | 0xFF (Diffusion) |
| **Unique_Id** | UDINT | CRC des attributs en cours de Keeper | Octet de poids faible en premier |
| **Status_Flag** | UINT | Valeurs de balise TUI | Voir l'attribut TUI de l'objet Keeper pour plus de détails |
| **Keeper_Mac_Id** | USINT | MAC ID du Keeper diffusant le TUI | Voir l'attribut TUI de l'objet Keeper pour plus de détails |
| **Réservé** | USINT | Réservé pour l'alignement des données | |
| **Net_Resource_Vendor_Id** | UINT | ID fournisseur de l'objet détenant le Net Resource pour une utilisation exclusive | |
| **Net_Resource_Serial_Number** | UDINT | Numéro de série de l'objet détenant le Net Resource pour une utilisation exclusive | |
| **Net_Resource_Class** | UDINT | Numéro de classe de l'objet détenant le Net Resource pour une utilisation exclusive | |
| **Net_Resource_Instance** | UDINT | Numéro d'instance de l'objet détenant le Net Resource pour une utilisation exclusive | |

Lors de la génération du Lpacket TUI transmis, tous les champs réservés doivent utiliser les valeurs spécifiées dans les champs réservés correspondants de l'attribut d'objet Keeper 0x0101. Lors du processus de configuration, l'outil de programmation doit affecter une valeur nulle à tous les champs réservés de l'attribut 0x0101.

### 6.8.3 Envoi et réception du Lpacket TUI

La priorité QoS demandée doit être planifiée.

Si l'objet Keeper est à l'état de fonctionnement MASTER_VERIFY, FAULTED_MASTER_VERIFY, MASTER, FAULTED_MASTER, NET_CHANGE_MASTER ou NET_CHANGE_FAULTED_MASTER, il doit tenter de transmettre un Lpacket TUI en tant que données planifiées une fois toutes les 4 NUT (NUT 0, 4, 8, 12, 16 ...). Tous les appareils Keeper doivent réserver 26 octets de durée de transmission de liaison planifiée une fois toutes les 4 NUT pour transmettre ce Lpacket (en cas d'envoi non planifié, il doit s'agir des informations non planifiées de priorité QoS la plus élevée).

La réception des Lpackets TUI doit être activée à la mise sous tension par l'objet ControlNet dans chaque appareil. Les Lpackets TUI réservés doivent être transmis à l'objet ControlNet local.

## 6.9 Paramètres de liaison Lpacket et tMinus Lpacket

### 6.9.1 Généralités

Pour synchroniser la modification des paramètres de liaison locaux, l'entité de gestion de station doit activer la réception des deux Lpackets à balise fixe: 0x15 (tMinus) et 0x81 (paramètres de liaison) via le service `DLL_enable_fixed_request` de la DLL.

Tous les nœuds d'une liaison doivent conserver deux exemplaires des paramètres de liaison: un exemplaire en cours et un exemplaire en attente. L'exemplaire en cours des paramètres de liaison doit être utilisé pour le fonctionnement à venir de la couche de liaison de données. L'exemplaire en attente doit être conservé pour permettre une modification synchronisée des paramètres de liaison. La synchronisation de modification est prise en charge par le service tMinus.

### 6.9.2 Structure des paramètres de liaison et des Lpackets tMinus

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe pour le service et de l'adresse de diffusion 0xFF. Le champ de données de liaison du Lpacket approprié doit contenir les éléments suivants dans l'ordre indiqué.

a) Contenu du champ de données Lpacket des paramètres de liaison:

```
class LinkParm_Lpacket: public fixedLpacket
{
   public:
 UINT NUT_length; // the length of the NUT in 10 µs increments;
 USINT smax; // highest MAC ID allowed to transmit scheduled
 USINT umax; // highest MAC ID allowed to transmit unscheduled
 USINT slotTime; // time allowed for line turnaround in 1 µs increments
 USINT blanking; // time to disable RX after DLPDU in 1,6 µs increments
 USINT gb_start; // 10 µs intervals from start of guardband to tone
 USINT gb_center; // 10 µs intervals from start of moderator to tone
 UINT zero; // reserved
 USINT modulus; // modulus of the interval counter
 USINT gb_prestart; // transmit cut-off, 10 µs intervals before tone
 UDINT unique_id; // 32 bit CRC calculated from Keeper configuration table
 UINT status_flag; // 16 bit status table set by the Keeper
 UINT reserved[8]; // all zeros
};
```

NOTE Le format des attributs d'instance 0x80 et 0x81 de l'objet ControlNet est exactement identique aux données de liaison du Lpacket 0x81 à balise fixe.

b) Contenu du champ de données Lpacket tMinus:

```
class tMinus_Lpacket: public fixedLpacket
{
   public:
   USINT     start_count;
   USINT     reserved[3];
};
```

où le start_count est le nombre de NUT à compter avant la transition et ne doit pas être inférieur à 8, le champ réservé doit contenir des zéros.

### 6.9.3 Envoi et réception des Lpackets tMinus et des paramètres de liaison

Les paramètres de liaison et les Lpackets tMinus sont émis par le Keeper maître lorsqu'une modification des paramètres de liaison est requise. Ils doivent être demandés à la priorité QoS spécifiée par l'objet Keeper appelant.

A la réception de LinkParm_Lpacket, chaque nœud doit charger l'exemplaire en attente de ses paramètres de liaison à l'aide de son service DLL_set_pending_request. Les paramètres en attente doivent être chargés en 3 opportunités de transmission non planifiées ou en 1 s, selon la valeur la plus élevée. L'attribut de paramètre de liaison en attente de l'objet ControlNet doit également être mis à jour avec l'exemplaire en attente des paramètres de liaison.

A la réception d'un tMinus_Lpacket, le nœud doit lancer une modification de paramètre synchronisé à l'aide du service DLL-tMinus-start-countdown-request transmettant le paramètre start_count. L'objet ControlNet doit copier son attribut de paramètre de liaison en attente dans son attribut de paramètre de liaison en cours lorsque DLL_tminus_zero_indication signale la fin de la modification de paramètre synchronisé.

NOTE La DLL copie ses propres paramètres de liaison en attente dans l'exemplaire en cours à l'issue du compte à rebours tMinus, aucun DLL_tminus_zero_indication n'étant signalé, sauf si la fin est précédée d'un DLL-tMinus-start-countdown-request.

Le Lpacket modérateur contient un champ appelé tMinus qui doit être utilisé pour synchroniser la mise à jour des paramètres de liaison en cours. DLL-tMinus-start-countdown-request doit favoriser la participation d'un nœud au compte à rebours tMinus et, s'il s'agit d'un nœud modérateur, doit initialiser le champ tMinus du Lpacket modérateur. Le nœud modérateur doit décrémenter ce champ jusqu'à zéro avant de transmettre chaque modérateur. Lorsque le champ tMinus passe de 1 à 0, la DLL de chaque nœud participant au compte à rebours doit générer un DLL_tminus_zero_indication en local dans son objet ControlNet et copier ses paramètres de liaison en attente dans son exemplaire en cours. Si le champ tMinus passe à 0 à partir d'une valeur autre que 1, le compte à rebours doit être annulé et aucun DLL_tminus_zero_indication ne doit être généré.

### 6.10 Lpacket I'm-alive

#### 6.10.1 Généralités

Ce service permet à tous les nœuds d'une liaison de reconnaître qu'un autre nœud vient d'être réinitialisé ou activé.

#### 6.10.2 Structure du Lpacket I'm-alive

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe I'm-alive et de l'adresse de diffusion 0xFF. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class Im_Alive_Lpacket: public fixedLpacket
{
   public:
   USINT     Lpacket_variant;
   USINT     sourceID;
   USINT     reserved[6];
};
```

Les champs Lpacket_variant et reserved doivent être nuls. Le MAC ID du nœud qui transmet le Lpacket doit être attribué au champ sourceID.

### 6.10.3 Envoi et réception de l'm Alive

La priorité QoS doit être spécifiée par l'utilisateur DL appelant.

Lorsqu'un nœud est d'abord mis en ligne avant de commencer des opérations complètes, le moniteur de connexion au réseau (NAM) du nouveau nœud doit diffuser au moins 3 `Im_Alive_Lpackets`, soumis aux règles de l'm Alive de traitement de l'état.

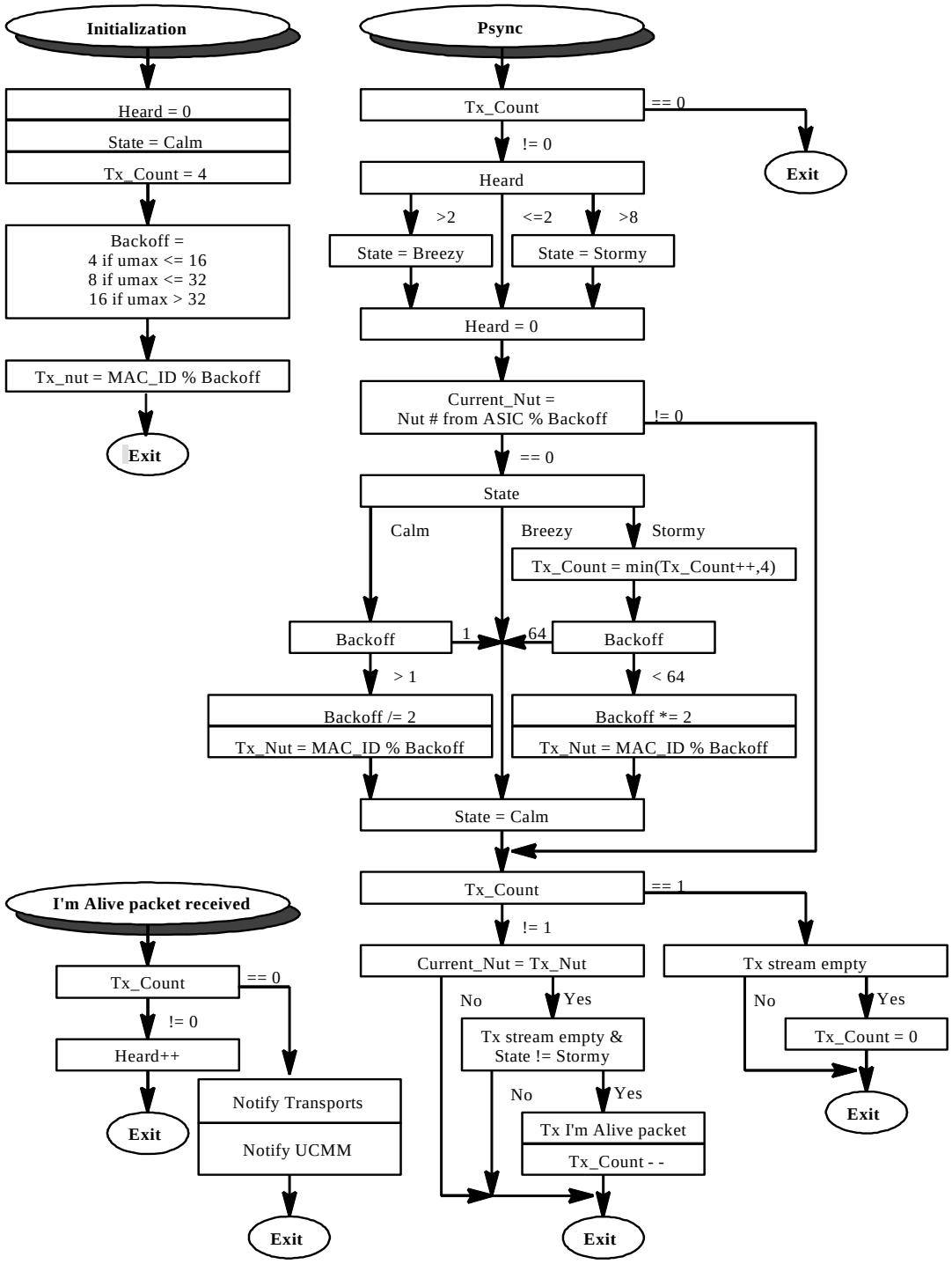
Les stations destinataires doivent informer leurs services de gestion.

NOTE Parmi les clients de ces diffusions se trouvent les gestionnaires de connexion de la couche supérieure de chaque nœud de la liaison. Les entités annuleront tous les enregistrements de transaction associés à un MAC ID qui vient d'être activé, ainsi que toutes les connexions transmises par son intermédiaire. Etant donné que les ID connexion (balises génériques) sont attribués de manière dynamique, cela garantit que l'ID connexion émis par l'ancien appareil au niveau du MAC ID n'est pas réutilisé improprement.

### 6.10.4 Traitement d'état l'm alive

Un nœud ne doit transmettre qu'un `Im_Alive_Lpacket` par NUT. Pour limiter les exigences de traitement dans chaque nœud, un nœud qui diffuse ses trois `Im_Alive_Lpacket` ne doit pas considérer valider la transmission de tous les `Im_Alive_Lpacket` précédés dans la NUT de plus de sept autres Lpackets de ce type. Ces Lpackets doivent être transmis dans la partie non planifiée de la NUT (priorité QoS `LOW` ou `HIGH`).

Etant donné qu'il est impossible de contrôler le moment où les nœuds sont activés ou réinitialisés les uns par rapport aux autres, il existe une possibilité de saturation des transmissions de Lpacket I'm alive si un grand nombre de nœuds entrent simultanément à l'état I'm alive. Pour limiter les exigences de traitement, un brouillage de transmission et un algorithme de réduction de puissance sont recommandés pour surveiller et contrôler l'intensité d'une saturation des transmissions. Ce type d'algorithme diffuse les Lpackets I'm alive sur des périodes de plus en plus longues. Il convient donc qu'un grand nombre de nœuds soient activés simultanément. Si peu de nœuds sont activés simultanément, il convient que l'algorithme leur permette de quitter rapidement l'état I'm alive étant donné qu'il ne peut pas y avoir de saturation des transmissions. Un exemple de ce type d'algorithme est présenté dans la Figure 16.




**Légende**

| Anglais | Français |
|---|---|
| initialization | initialisation |
| heard | entendu |
| state | état |
| exit | sortie |
| backoff | réduction de puissance |
| .. if ... | ... si .. |
| I'm Alive packet received | paquet I'm Alive reçu |
| notify | notifier |

**Figure 16 – Exemple d'algorithme de traitement I'm alive**

### 6.11 Lpackets ping

#### 6.11.1 Généralités

Le service ping permet à l'entité de gestion de station de lancer une demande ping et de recevoir une réponse ping d'une autre station active ou d'une station adressée associée au fournisseur DLS.

Toutes les stations contiennent le service ping et, lors de la mise sous tension, le service enable-fixed-request est utilisé par l'utilisateur DLS afin d'activer la réception et la réponse aux demandes ping dans le fournisseur DLS local.

#### 6.11.2 Structure des Lpackets ping

Les champs de taille et de contrôle doivent se présenter au format de balise fixe.

a) Lors de l'envoi d'une demande ping, l'adresse doit être composée de la balise fixe de demande ping et d'une adresse spécifiée par l'utilisateur DL. L'adresse spécifiée peut être un MAC ID unique ou l'adresse de diffusion 0xFF.

   Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class Ping_request: public fixedLpacket
{
   public:
   USINT     client_ID;
   USINT     reserved[3];
};
```

   où client_ID est le MAC ID de l'auteur de la demande, et le champ réservé doit contenir des zéros.

b) Lors de l'envoi d'une réponse ping, l'adresse doit être composée de la balise fixe de réponse ping et de l'adresse client_ID provenant de l'indication de demande ping correspondante (utilisée ici comme destination de la réponse).

   Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué;

```
class Ping_reply: public fixedLpacket
{
   public:
   USINT     server_ID;
   USINT     vendor_specific[4];
   USINT     reserved[3];
};
```

   où le champ server_ID doit contenir le MAC ID du nœud répondant à la demande, où le champ vendor_specific peut contenir des données et où le champ réservé doit contenir des zéros.

Le service `DLL_xmit_fixed_request(id, Lpacket, sizeof(Ping_reply), HIGH, 0x29, client_ID)` doit être utilisé pour envoyer la réponse au MAC ID spécifié dans le champ `client_ID` du Lpacket `Ping_request` reçu.

NOTE En général, le champ vendor_specific est utilisé pour les informations de débogage ou de diagnostic.

#### 6.11.3 Envoi et réception des Lpackets ping

La priorité QoS d'une demande ping doit être spécifiée par l'utilisateur DL appelant. La priorité QoS d'une réponse ping doit être High (Elevée).

Les Lpackets ping-request peuvent être envoyés par une entité de gestion locale dans une station.

Lors de la mise sous tension, le service interne `DLL_enable_fixed_request(id, 0x09)` de la couche de liaison de données doit être appelé pour activer la réception des demandes ping à balise fixe.

Lors de la réception d'une demande ping, chaque station destinataire compose un Lpacket à balise fixe de réponse ping et l'envoie à la station demandeuse.

Lors de la réception d'une réponse ping, le nœud demandeur transmet l'indication à l'entité de gestion locale.

### 6.12 Lpacket WAMI

#### 6.12.1 Généralités

Un serveur WAMI (Where Am I?) doit être présent dans tous les nœuds permanents dotés d'un port d'accès au réseau (NAP). Le serveur ne doit pas être présent sur les nœuds transitoires.

NOTE Le serveur WAMI (where am I) permet à un nœud transitoire (un nœud qui se connecte au réseau par l'intermédiaire du port d'accès au réseau) de déterminer le MAC ID du nœud auquel il est associé. Cela est très pratique pour les logiciels d'un appareil transitoire qui souhaitent établir une communication avec le nœud permanent local.

#### 6.12.2 Structure du Lpacket WAMI

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe WAMI et de 0xFF, l'adresse de diffusion de tous les nœuds MAC IC sur la liaison. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class WAMI_Lpacket: public fixedLpacket
{
   public:
   USINT     requestID;
   USINT     responseID;
   USINT     reserved[6];
};
```

Dans une demande WAMI, le `requestID` doit contenir le MAC ID du nœud transitoire qui envoie le Lpacket. Les autres champs doivent contenir des zéros. Pour convertir une demande en réponse, le serveur doit copier son MAC ID dans le champ `responseID`, puis répondre. La priorité QoS est Low (Basse).

NOTE 1 Lorsque le service `DLL_enable_fixed_request(id, 0x86)` de la DLL active une réception de balise fixe WAMI, le service `DLL_recv_fixed_indication` reçoit les balises fixes.

NOTE 2 Le Moniteur de connexion au réseau (voir 8.1) décrit les modes opératoires visant à s'assurer que le MAC ID transitoire ne duplique pas un MAC ID de liaison existant.

#### 6.12.3 Envoi et réception du Lpacket WAMI

La priorité QoS doit être Low (Basse).

Le serveur WAMI est uniquement mis en œuvre dans les appareils dotés d'un NAP, ces appareils doivent répéter tous les messages entre la liaison principale et le NAP, et inversement. De plus, ces appareils doivent comporter des moyens de détection des messages reçus par l'intermédiaire de leur NAP, et doivent uniquement répondre aux Lpackets respectant chacun des critères ci-dessous:

– Balise fixe WAMI 0x86;
– diffusion de destination (0xFF);
– reçu par l'intermédiaire du NAP local de l'appareil.

Les Lpackets répondant à ces critères doivent être appelés Lpackets WAMI de demande. Pour ce type de Lpackets, le serveur doit générer une réponse WAMI adressée au MAC ID qui envoie la demande WAMI. Le serveur WAMI doit ignorer tous les autres Lpackets 0x86 à balise fixe WAMI.

### 6.13 Lpacket Debug

Ce service peut être utilisé pour transmettre l'état d'un objet en générant une trace des transitions d'état d'objet sur le réseau.

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe Debug et du MAC ID de destination fourni par le service de transmission DLL. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class Debug_Lpacket: public fixedLpacket
{
   public:
   UINT      object_class;
   UINT      data1;
   UINT      data2;
};
```

La valeur de `object_class` doit être le code de classe (voir CEI 61158-6-2) de l'objet qui transmet son état. La signification des champs `data1` et `data2` doit être définie dans la définition d'objet. Les appareils ne doivent pas générer plusieurs Lpackets Debug à balise fixe toutes les 10 s en moyenne, de manière à ne pas gêner les autres transmissions non planifiées.

### 6.14 Lpacket IP

Ce service (balise fixe 0x85) permet d'envoyer des trames IP brutes. La taille maximale de trame IP qui peut être envoyée dans un Lpacket est de 506 octets. Les diffusions sont envoyées avec la destination 0xFF. Les trames IP entre deux nœuds doivent comporter le bon MAC ID de destination à balise fixe. Un protocole de résolution d'adresse doit être utilisé afin de déterminer le MAC ID associé à une adresse IP particulière.

### 6.15 Lpacket Ethernet

Ce service (balise fixe 0x89) permet d'envoyer des trames Ethernet autres que des trames IP. Il s'agit, sans s'y limiter, de trames Ethernet ARP (Address Resolution Protocol). Cela permet de mettre simplement en œuvre un protocole IP et un protocole de résolution d'adresse de style Ethernet sur la couche physique de Type 2, une pile TCP/IP/Ethernet existante pouvant être utilisée avec de petites modifications de paquet.

L'octet le plus bas de l'adresse physique Ethernet est utilisé pour représenter le MAC ID de Type 2 (les adresses physiques Ethernet sont envoyées avec l'octet le plus élevé en premier). Une diffusion de Type 2 (MAC ID 0xFF) doit être étendue à une adresse de diffusion Ethernet (FF FF FF FF FF FF).

Lors du démarrage, une pile TCP/IP contenant Ethernet ARP doit connaître l'adresse physique du nœud local. Il est reporté sous la forme 00 00 00 00 00 <MAC ID>.

L'octet bas de l'adresse de destination d'une trame Ethernet fait office de destination à balise fixe de Type 2. L'octet bas de l'adresse source d'une trame Ethernet sera identique au MAC ID du nœud local (signalé à la pile TCP/IP au démarrage) et est envoyé comme MAC ID source de la DLPDU. Le « type de trame » Ethernet (08 06 pour ARP) compose les deux premiers octets de la partie données du Lpacket, et jusqu'à 504 octets sont disponibles pour la partie données de la trame Ethernet (demande/réponse ARP utilise 28 octets).

Etant donné que le « type de trame » Ethernet est inclus dans le Lpacket, d'autres types de trames Ethernet (RARP, par exemple) peuvent également être envoyés à l'aide de ce service à balise fixe Ethernet.

# 7 Objets de gestion de la station

## 7.1 Généralités

Cette spécification de protocole est assez souple. Elle peut assurer un transfert E-S déterministe et synchronisé à intervalles cycliques (toutes les 1 ms) et des séparations de nœud jusqu'à 25 km. Ces performances peuvent être ajustées en ligne en configurant les paramètres de liaison. Ces paramètres, qui régissent l'accès à la liaison, peuvent être réglés si nécessaire pour correspondre à des applications différentes.

La gestion de station permet de modifier ces paramètres en ligne, le réseau fonctionnant, elle permet également à la liaison de continuer à fonctionner en cas d'ajout ou de retrait de nœuds.

Les fonctions de gestion de station permettent

– d'accéder aux variables et événements de chacune des couches;
– d'utiliser une interface utilisateur commune;
– de coordonner la modification des paramètres de liaison;
– d'ajouter des nœuds à la liaison sans interruption;
– d'ajuster les paramètres de liaison;
– d'assurer la synchronisation d'horloge entre les nœuds.

L'accès aux variables et événements dans chacune des couches est possible en définissant les interfaces d'objet de ces paramètres.

L'objet ControlNet offre une interface cohérente aux couches de liaison physiques et de données.

L'objet Keeper détient tous les attributs de liaison de tous les appareils d'une liaison et doit être chargé de les distribuer aux appareils de manière ordonnée. Il détient également (pour le logiciel de planification de liaison) un exemplaire des données de l'auteur de la connexion de tous les appareils à l'origine de la connexion utilisant un réseau. Si plusieurs objets Keeper sont présents sur une liaison, ils procèdent à des négociations afin de déterminer celui qui va devenir maître et ceux qui vont assurer les fonctions de sauvegarde. Le Keeper maître est celui qui distribue activement les paramètres de liaison et les attributs aux nœuds du réseau. Un objet Keeper de sauvegarde est celui qui surveille l'activité réseau associée au Keeper et qui peut assurer le rôle d'objet Keeper maître en cas de défaillance du Keeper maître.

L'objet Keeper contient également un support pour l'obtention, la gestion et la délivrance de la ressource réseau, sémaphore réseau utilisé pour éliminer les conflits lors de la modification des données d'attribut détenues par le/les objet(s) Keeper sur une liaison.

L'objet de planification doit exister dans les appareils à l'origine de la connexion (CO). L'objet de planification doit être utilisé par un logiciel de planification de liaison (LSS) pour

– lire les informations de connexion planifiée;
– écrire les informations de planification;
– offrir une signature (clé logicielle) à l'auteur de la connexion, qui doit être utilisée pour authentifier l'accès de l'appareil CO à une liaison particulière.

L'objet d'interface TCP/IP fournit le mécanisme de configuration de l'interface TCP/IP d'un appareil.

L'objet de liaison Ethernet gère les compteurs spécifiques à la liaison et les informations d'état d'un port Ethernet physique ISO/CEI 8802-3.

L'objet DeviceNet offre une interface cohérente aux couches de liaison physiques et de données.

L'objet de configuration de connexion offre une interface de création, de configuration et de contrôle des connexions d'un appareil.

L'objet DLR (Device Level Ring) offre une interface de configuration et de surveillance du protocole DLR.

L'objet QoS offre une interface de configuration de certains comportements liés à QoS.

L'objet de port décrit les ports de Type 2 présents sur l'appareil.

## 7.2 Objet ControlNet

### 7.2.1 Présentation

L'objet ControlNet doit fournir une interface cohérente de gestion de station aux couches de liaison physiques et de données. Cet objet doit mettre les informations de diagnostic de ces couches à la disposition des applications client. Chaque nœud doit prendre en charge un objet ControlNet par liaison.

Un appareil peut contenir plusieurs nœuds.

### 7.2.2 Attributs de classe

L'objet ControlNet doit prendre en charge les attributs de classe tel que spécifié dans le Tableau 14.

**Tableau 14 – Attributs de classe de l'objet ControlNet**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| **0x01** | Obligatoire | Get | Revision | UINT | Révision de cet objet | Première révision, valeur = 1 |
| **0x02** | Obligatoire | Get | Max. instance | UDINT | Nombre maximal d'instances | Valeur déterminée par les caractéristiques du nœud |

### 7.2.3 Attributs d'instance

#### 7.2.3.1 Généralités

L'objet ControlNet doit prendre en charge les attributs d'instance tel que spécifié dans le Tableau 15.

**Tableau 15 – Attributs d'instance de l'objet ControlNet**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| 0x80 | Facultatif | Get | Pending_Link_Config | STRUCT de 34 octets | Paramètres de configuration de liaison en attente | voir 6.9.2 |
| | | | Link_Config | STRUCT de 12 octets | paramètres de liaison en attente | |
| | | | NUT_length | UINT | DLL NUT_length | dans des impulsions de 10 µs |
| | | | smax | USINT | DLL smax | 0 à 99 |
| | | | umax | USINT | DLL umax | 1 à 99 |
| | | | slotTime | USINT | DLL slotTime | dans des impulsions de 1 µs |
| | | | blanking | USINT | DLL blanking | dans des impulsions de 1,6 µs |
| | | | gb_start | USINT | DLL gb_start | dans des impulsions de 10 µs |
| | | | gb_center | USINT | DLL gb_center | dans des impulsions de 10 µs |
| | | | réservé | UINT | réservé | |
| | | | modulus | USINT | DLL modulus | 127 requis |
| | | | gb_prestart | USINT | DLL gb_prestart | dans des impulsions de 10 µs |
| | | | TUI | STRUCT de 22 octets | Keeper TUI | voir 7.2.3.3 |
| | | | unique_ID | UDINT | Keeper CRC | voir 7.2.3.4 |
| | | | status_flag | UINT | balise TUI | voir 7.2.3.5 |
| | | | réservé | USINT[16] | réservé | |
| 0x81 | Obligatoire | Get | current_link_config | STRUCT de 34 octets | Paramètres de configuration de liaison en cours | identique à l'attribut 0x80 voir 6.9.2 |
| 0x82 | Obligatoire | Get/ Get_and_Clear | diagnostic_counters | STRUCT de 42 octets | compteurs de diagnostic | voir 7.2.3.7 |
| | | | buffer_errors | UINT | compteur d'événements de la mémoire tampon | voir 7.2.3.8 |
| | | | error_log | SWORD[8] | journalisation DLPDU erronée | voir 7.2.3.9 |
| | | | event_counters | STRUCT de 32 octets | compteurs de diagnostic | voir 7.2.3.10 |
| | | | good_frames_ transmitted | SWORD[3] | DLPDU correctes transmises (LSB en premier) | |
| | | | good_frames_ received | SWORD[3] | DLPDU correctes reçues (LSB en premier) | |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| | | | selected_channel_ frame_errors | USINT | erreurs de tramage détectées sur le canal de réception actif | |
| | | | channel_A_ frame_errors | USINT | erreurs de tramage détectées sur le canal A | |
| | | | channel_B_ frame_errors | USINT | erreurs de tramage détectées sur le canal B | |
| | | | aborted_frames_ transmitted | USINT | DLPDU annulées pendant la transmission (dépassement négatif de capacité de transmission) | |
| | | | highwaters | USINT | dépassement négatif de capacité de transmission LLC et dépassement de capacité de réception LLC | |
| | | | NUT_overloads | USINT | aucune durée planifiée dans la NUT (toutes les durées utilisées pour les transmissions planifiées) | |
| | | | slot_overloads | USINT | plus de données planifiées mises en file d'attente pour une NUT que le nombre admis par le paramètre sched_max_frame | |
| | | | blockages | USINT | une seule taille de Lpacket dépasse le paramètre de trame sched_max_ | |
| | | | non_concurrence | USINT | au moins deux nœuds n'ont pas pu convenir de celui qui doit assurer la transmission | |
| | | | aborted_frames_ received | USINT | DLPDU incomplètes reçues | |
| | | | lonely_counter | USINT | nombre de fois que rien n'a été entendu sur le réseau pendant au moins 8 NUT | |
| | | | duplicate_node | USINT | DLPDU reçue d'un nœud avec un MAC ID de nœud local | |
| | | | noise_hits | USINT | bruit détecté ayant verrouillé le modem rx PLL | |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| | | | collisions | USINT | données Rx entendues juste au moment de la transmission | |
| | | | mod_MAC_ID | USINT | MAC ID du nœud modérateur en cours | |
| | | | non_lowman_mods | USINT | DLPDU modérateurs entendues depuis des nœuds non lowman | |
| | | | rogue_count | USINT | événements suspects entendus | |
| | | | unheard_moderator | USINT | DLPDU entendues, mais pas les modérateurs | |
| | | | vendor_specific | USINT | spécifique au fournisseur | |
| | | | réservé | SWORD[4] | réservé | |
| | | | vendor_specific | USINT | spécifique au fournisseur | |
| | | | vendor_specific | USINT | spécifique au fournisseur | . |
| | | | réservé | SWORD | réservé | |
| 0x83 | Obligatoire | Get | station_status | STRUCT de 6 octets | état de la station | voir 7.2.3.11 |
| | | | MAC_ver | SWORD | mise en œuvre MAC | voir 7.2.3.12 |
| | | | vendor_specific | SWORD[4] | spécifique au fournisseur | |
| | | | channel_state | SWORD | DEL d'état du canal, avertissement de redondance et bits de canal actif | voir 7.2.3.12 |
| 0x84 | Get obligatoire, Set-(une seule fois) Facultatif | Get/Set | MAC_ID | STRUCT de 4 octets | commutateur MAC ID et paramètres en cours | |
| | | | MAC_ID_current | USINT | MAC ID en cours | Plage comprise entre 1 et 99. Voir 7.2.3.14 |
| | | | MAC_ID_switches | USINT | paramètres de commutation MAC ID | 0 à 99, 0xFF. Voir 7.2.3.15 |
| | | | MAC_ID_changed | BOOL | commutateurs MAC ID modifiés depuis la réinitialisation | voir 7.2.3.16 |
| | | | Réservé | USINT | réservé | |
| 0x85 | Facultatif | Get/Set | Sched_max_frame | STRUCT de 2 octets | limite de données planifiée | |
| | | | Sched_max_frame | USINT | DLPDU maximal planifié | en paires d'octets. Voir 7.2.3.17 |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| | | | Réservé | USINT | réservé | |
| 0x86 | Obligatoire | Get | error_log | STRUCT de 10 octets | Comptes d'erreur de la mémoire tampon du microprogramme du pilote et liste des nœuds gênants | voir 7.2.3.18 |
| | | | buffer_errors | UINT | compteur d'événements de la mémoire tampon | |
| | | | error_log | SWORD[8] | journalisation DLPDU erronée | . |
| 0x87 | Facultatif | Get/ Get_and_Clear | extended_diagnostic_counters | STRUCT de 264 octets | compteurs de diagnostics supplémentaires | |
| | | | unsched_transmitted | UDINT | nombre de Lpackets non planifiés transmis | |
| | | | sched_highwater | UDINT | Nombre max. de Lpackets partagés/non planifiés dans la file d'attente de transmission | |
| | | | sched_data | 128 STRUCTS de 2 octets (256 octets) | informations de planification | |
| | | | words_in_use | USINT | nombre de mots planifiés utilisés par ce nœud pendant la NUT | |
| | | | Lpkt_in_use | USINT | nombre de Lpackets planifiés utilisés par ce nœud pendant la NUT | |
| 0x88 | Facultatif | Get | Active_node_table | MATRICE de 32 octets | un bit pour chaque MAC ID<br>1 = nœud présent<br>0 = nœud manquant | voir 7.2.3.19 |
| 0x89 | Facultatif | Get | New_node_table | MATRICE de 32 octets | un bit pour chaque MAC ID<br>1 = le nœud a rejoint la liaison récemment<br>0 = le nœud n'a pas rejoint la liaison récemment | voir 7.2.3.20 |

où LSB = octet de poids faible.

### 7.2.3.2 pending_link_config

Les valeurs de l'attribut permettent de lire les paramètres de liaison en attente. Cet attribut doit être modifié sur la base de la fonction de gestion de station appelée modification de paramètre synchronisé et ne doit pas être défini par l'intermédiaire des services de l'objet ControlNet.

#### 7.2.3.3 Keeper TUI

L'objet ControlNet doit conserver des exemplaires distincts du sous-attribut TUI pour les attributs pending_link_config et current_link_config. Les données TUI de l'attribut pending_link_config doivent être modifiées à la réception d'une balise fixe 0x81. Les données TUI de l'attribut current_link_config doivent être modifiées à l'issue d'une modification de paramètre synchronisé.

#### 7.2.3.4 unique_ID

Le champ unique_ID doit être composé d'une valeur CRC de 32 bits calculée en fonction du contenu de la table de configuration de réseau gérée par l'objet Keeper. Cette valeur doit être copiée à partir du Lpacket 0x81 à balise fixe.

#### 7.2.3.5 status_flag

Le champ status_flag du sous-attribut TUI doit être composé d'une valeur de 16 bits. Les bits non utilisés doivent être réservés et définis sur zéro.

Comme le montre le Tableau 16, le bit d'état de l'objet Keeper (bit 4) doit indiquer si ce nœud a déjà entendu un TUI provenant d'un objet Keeper maître. L'objet ControlNet doit annuler ce bit lors de l'initialisation. Etant donné qu'une valeur est attribuée au bit de tous les Lpackets TUI envoyés par l'objet Keeper maître, l'objet ControlNet peut ne pas manipuler ce bit.

La réception des Lpackets TUI doit être activée à la mise sous tension à l'aide du service `DLL_enable_fixed_request(id, 0x84)`. Le champ d'état des Lpackets TUI doit être vérifié. Si le bit 3 = 0, le bit 4 = 1 et le bit 5 = 1, le Lpacket TUI doit être copié dans l'attribut 0x81 et la réception TUI être désactivée à l'aide de `DLL_disable_fixed_request(id, 0x84)`. Les bits 0 et 1 du champ d'état TUI doivent être utilisés pour remplacer les paramètres de redondance de réseau par défaut du nœud. Ils doivent indiquer si le réseau est en cours de fonctionnement en mode de canal unique ou de canal redondant. Seules les informations de redondance du réseau provenant des Lpackets TUI dont les bits configurés Keeper State et Keeper sont définis doivent être utilisées.

**Tableau 16 – Bits de balise d'état TUI**

| Bit | Signification |
|---|---|
| 0 – 1 | Redondance du réseau<br>Bit 1 = 0, bit 0 = 0; Combinaison illégale<br>Bit 1 = 0, bit 0 = 1; Canal B uniquement<br>Bit 1 = 1, bit 0 = 0; Canal A uniquement<br>Bit 1 = 1, bit 0 = 1; Redondant |
| 2 | Bit de gestion des ressources réseau<br>0 = Ressources réseau non gérées<br>1 = Ressources réseau gérées pour l'objet identifié dans le TUI |
| 3 | Bit de modification en cours du réseau<br>0 = Pas de modification de réseau en cours<br>1 = Modification de réseau en cours |
| 4 | Bit d'état de l'objet Keeper<br>0 = Le TUI transmis par l'objet Keeper n'est pas à l'état MAITRE<br>1 = Le TUI transmis par l'objet Keeper est à l'état MAITRE |
| 5 | Bit configuré de l'objet Keeper<br>0 = Aucun objet Keeper maître entendu depuis l'entrée dans la liaison<br>1 = Objet Keeper maître entendu depuis l'entrée dans la liaison |
| 6 – 15 | réservé (valeur 0 attribuée) |

#### 7.2.3.6 current_link_config

Les valeurs de l'attribut current_link_config correspondent aux paramètres de liaison actuellement utilisés. Le format de cet attribut doit être identique à celui de l'attribut pending_link_config.

#### 7.2.3.7 diagnostic_counters

L'attribut diagnostic_counters doit gérer les comptages d'événements dans les couches physiques et de données.

NOTE L'attribut diagnostic_counters peut être lu sans affecter les valeurs grâce à l'une des demandes de service Get_Attribute. Les informations peuvent être lues et les valeurs réinitialisées à l'aide de la demande de service Get_and_Clear. Voir 7.2.5.

Les informations d'événement gérées par le microprogramme doivent également prendre la valeur get (mais pas clear) grâce à l'attribut error_log. Cela pour des raisons d'efficacité. Les informations gérées par le microprogramme doivent être immédiatement et directement accessibles.

#### 7.2.3.8 buffer_errors

La valeur du sous-attribut buffer_errors doit être incrémentée à chaque événement de dépassement de capacité de réception et dépassement de capacité de transmission. Si une mise en œuvre ne dépasse ni ne fait jamais l'objet d'un dépassement négatif de ses capacités, une valeur nulle doit être attribuée à buffer_errors.

#### 7.2.3.9 error_log

Si `DLL_event_indication` renvoie une valeur de `DLL_EV_badFrame`, le sous-attribut error_log enregistre les huit dernières indications de ce type. Le paramètre `mac_id` de cette indication doit être enregistré. L'ordre des MAC ID doit commencer par l'erreur la plus récente et se terminer par l'erreur la plus ancienne. Les valeurs nulles représentent les positions non utilisées dans le journal des erreurs.

#### 7.2.3.10 event_counters

Le sous-attribut event_counters doit contenir les comptages d'événements de liaison. Le compteur doit s'arrêter lorsque les valeurs dépassent leurs limites.

#### 7.2.3.11 station_status

Cet attribut doit uniquement prendre la valeur get.

#### 7.2.3.12 MAC_ver

Les bits de l'octet MAC_ver sont décrits dans le Tableau 17.

**Tableau 17 – Bits de Mac_ver**

| Bits | Description |
|---|---|
| 0-3 | **Révision MAC**[a] – révision de la mise en œuvre MAC<br>0x1 à 0xF, 0x0 est réservé |
| 4 | Réservé, mis à 0 |
| 5-7 | **Mise en œuvre MAC**[a] – nom du vendeur et de la mise en œuvre<br>0 – 2 = affecté<br>3 = réservé – non affecté pour de futures mises en œuvre<br>4 = affecté<br>5 – 6 = réservé – non affecté pour de futures mises en œuvre<br>7 = réservé pour l'extension du champ étendu MAC_ver |
| a La révision et la mise en œuvre MAC sont affectées et décrites par ODVA, Inc. | |

### 7.2.3.13 channel_state

L'octet channel_state doit représenter l'état en cours des voyants d'état du réseau, s'ils sont mis en place. Les bits de l'octet channel_state sont décrits dans le Tableau 18.

NOTE Les voyants d'état du réseau sont présentés dans l'0.

**Tableau 18 – Bits d'état du canal**

| Bits | Description |
|---|---|
| 0,1,2 | Etat de la DEL du canal A<br>0 = Inactif<br>1 = Vert fixe<br>2 = Vert clignotant<br>3 = Rouge clignotant<br>4 = Rouge-vert clignotant<br>5 = Chenillard rouge<br>6 = Chenillard rouge-vert<br>7 = Rouge fixe |
| 3,4,5 | Etat de la DEL du canal B<br>Idem DEL du canal A. |
| 6 | **Avertissement de redondance** – Le contrôleur ne peut pas utiliser le canal inactif d'une configuration redondante.<br>0 = normal (pas d'avertissement)<br>1 = avertissement |
| 7 | **Canal actif** – Indique lequel des deux canaux le récepteur en cours est en train d'écouter.<br>0 = Canal B<br>1 = Canal A |

### 7.2.3.14 MAC_ID_current

La valeur MAC_ID_current doit être accessible à l'aide des services Get_Attribute. Le paramétrage doit être requis sur les appareils sans commutateur d'adresse. Seul le premier ensemble de valeurs MAC_ID_current après la réinitialisation d'un l'objet ControlNet doit être accepté. Le paramétrage doit être facultatif sur les appareils dotés de commutateurs d'adresse.

La plage de valeurs admises de MAC ID doit être comprise entre 1 et 99.

#### 7.2.3.15 MAC_ID_switches

MAC_ID_switches doit prendre la valeur get et pas set sur tous les appareils. La valeur renvoyée doit être 0xFF pour les nœuds à adresse automatique et les nœuds sans commutateur d'adresse.

#### 7.2.3.16 MAC_ID_changed

La valeur MAC_ID_changed doit être « clear » lorsque l'appareil est réinitialisé et « set » lorsque l'appareil détecte une modification des commutateurs d'adresse. Une fois définie, elle reste tant que l'appareil n'est pas réinitialisé, même si les commutateurs d'adresse reprennent leurs paramètres d'origine. La fréquence de vérification des commutateurs d'adresse doit dépendre de l'appareil.

En cas de détection d'une modification des commutateurs d'adresse réseau, une valeur doit être attribuée à cette balise et une erreur mineure signalée à l'hôte.

#### 7.2.3.17 sched_max_frame

L'attribut sched_max_frame doit être utilisé pour limiter la taille maximale des transmissions planifiées du nœud. Une valeur égale à 0 doit indiquer qu'un nœud ne doit pas transmettre pendant la partie planifiée de la NUT.

#### 7.2.3.18 error_log

L'attribut d'instance 0x86 doit être une copie du sous-attribut error_log contenu dans l'attribut d'instance 0x82.

NOTE Les informations de cet attribut sont une copie des informations disponibles dans l'attribut diagnostic_counters. Cette copie peut uniquement prendre la valeur get. Les valeurs de cet attribut peuvent être annulées uniquement à l'aide de la demande de service Get_and_Clear sur l'attribut diagnostic_counters. Voir 7.2.5 pour plus de détails.

Cette copie d'error_log doit uniquement être accessible par l'intermédiaire des services Get_Attribute. Cet attribut doit uniquement être annulé si la copie principale dans 0x82 l'est.

#### 7.2.3.19 Table des nœuds actifs (attribut d'instance = 0x88)

L'attribut d'instance 0x88 de l'objet ControlNet doit être composé d'une matrice de 256 bits (un par MAC ID). Le bit de poids faible doit correspondre à MAC ID = 0. Le bit de poids fort doit correspondre à MAC ID = 255. Le bit d'un MAC ID particulier doit être défini (1) pour un nœud dans un délai d'1 s après la transmission de nœud sur la liaison. Le bit doit être nul (0) dans les 20 s du nœud qui quitte la liaison. Un nœud doit être considéré comme ayant quitté la liaison s'il manque trois opportunités de transmission consécutives.

#### 7.2.3.20 Nouvelle table de nœuds (attribut d'instance = 0x89)

L'attribut d'instance 0x89 de l'objet ControlNet doit être composé d'une matrice de 256 bits (un par MAC ID). Le bit de poids faible doit correspondre à MAC ID = 0. Le bit de poids fort doit correspondre à MAC ID = 255. Le bit d'un MAC ID particulier doit être défini (1) pour un nœud dans un délai d'1 s après une transmission de nœud sur la liaison. Le bit doit être nul (0) entre 10 s et 20 s après que le nœud a rejoint la liaison.

### 7.2.4 Services communs

#### 7.2.4.1 Généralités

L'objet ControlNet doit prendre en charge les services communs spécifiés dans le Tableau 19.

**Tableau 19 – Services communs de l'objet ControlNet**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| **0x05** | N/A | Obligatoire | Reset | Renvoie l'objet ControlNet à l'état de mise sous tension |
| **0x10** | N/A | Sous condition | Set_Attribute_Single | Définit les informations de configuration spécifiques au réseau ou au nœud |
| **0x0E** | Obligatoire | Obligatoire | Get_Attribute_Single | Obtient les informations de configuration spécifiques au réseau ou au nœud |
| **0x01** | N/A | Facultatif | Get_Attribute_All | Obtient les informations de configuration spécifiques au réseau et au nœud |
| **0x02** | N/A | Facultatif | Set_Attribute_All | Définit les informations de configuration spécifiques au réseau et au nœud |
| **0x03** | N/A | Facultatif | Get_Attribute_List | Obtient les informations de configuration spécifiques au réseau et/ou au nœud |
| **0x04** | N/A | Facultatif | Set_Attribute_List | Définit les informations de configuration spécifiques au réseau et/ou au nœud |

#### 7.2.4.2 Service de réinitialisation

Le service Reset doit réinitialiser le Moniteur de connexion de réseau (NAM) afin de surveiller l'activité de liaison, puis tente de rejoindre la liaison. Les commutateurs MAC ID ne doivent pas être réévalués en raison de la réinitialisation.

L'objet ControlNet peut ne pas répondre à une demande Reset avant de réellement lancer une opération de réinitialisation.

#### 7.2.4.3 Set_Attribute_Single

Si un attribut a été mis en œuvre comme étant définissable, le service Set_Attribute_Single doit être requis.

#### 7.2.4.4 Réponse Get_Attribute_All

La partie de données de réponse spécifique à l'objet/au service d'une réponse Get_Attribute réussie est présentée dans les attributs d'instance de la table d'objets ControlNet. La taille de cette réponse dépend des instances d'attribut en cours d'accès.

Compte tenu de la taille importante de la réponse Get_Attribute_All, des appareils aux ressources limitées peuvent ne pas prendre en charge ce service.

La partie de données d'une demande de service Get_Attribute_All doit contenir les attributs suivants dans l'ordre suivant:

0x81 – current_net_config

0x82 – diagnostic_counters

0x83 – station_status

0x84 – MAC_ID

0x86 – error_log

### 7.2.5 Services spécifiques à la classe

#### 7.2.5.1 Généralités

L'objet ControlNet doit prendre en charge les services spécifiques à la classe spécifiés dans le Tableau 20.

**Tableau 20 – Services spécifiques à la classe de l'objet ControlNet**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| **0x4C** | N/A | Obligatoire | Get_and_Clear | Obtient, puis annule les compteurs de diagnostic |
| **0x4D** | N/A | Obligatoire | Enter_Listen_Only | Force le nœud à entrer et rester en mode d'écoute seulement jusqu'à la réinitialisation |
| **0x4E** | N/A | Obligatoire (Voir 7.2.5.4) | Where_am_I | Détermine le MAC ID de l'appareil auquel est associé ce nœud par l'intermédiaire du NAP |
| **0x4F** | N/A | Sous condition | Auto_Address | Requis dans tous les nœuds d'adresse automatique. Sélectionner un nouveau MAC ID à l'aide de la séquence d'adresse automatique du NAM |

#### 7.2.5.2 Service Get_and_Clear

Ce service spécifique à la classe doit appeler une réponse identique au service Get_Attribute_Single. Toutefois, après la génération du Lpacket de réponse, la valeur de l'attribut doit être nulle. Seuls les attributs d'instance 0x82 et 0x87 doivent prendre en charge ce service.

#### 7.2.5.3 Service Enter_Listen_Only

Le service Enter_Listen_Only doit inciter cet objet à envoyer un `DLL_listen_only_request(FALSE)` à la DLL associée à cette instance de l'objet ControlNet. Ce service peut ne pas générer une réponse étant donné que le nœud peut s'arrêter de transmettre avant l'envoi de la réponse.

Il convient que le nœud reste à l'état d'écoute seulement tant que la demande de réinitialisation n'est pas envoyée à l'objet ControlNet ou Identity.

Tant que les appareils sont à l'état d'écoute seulement à cause du service Enter_Listen_Only, ils ne sont pas obligé d'appliquer les bits de redondance du réseau de la TUI (voir Tableau 26), et les appareils de support redondants peuvent autoriser les deux canaux.

NOTE Le service Enter_Listen_Only peut être utilisé pour les besoins du diagnostic afin de forcer les nœuds à diminuer la liaison tant qu'ils n'ont pas été spécifiquement invités à la rejoindre. De cette façon, on peut retirer effectivement un nœud de la liaison sans déconnecter physiquement le câble réseau.

#### 7.2.5.4 Service Where_am_I

Tous les nœuds transitoires doivent mettre en œuvre le service Where_am_I. Ce service doit déterminer le MAC ID du nœud auquel le nœud transitoire est connecté par l'intermédiaire de son port d'accès au réseau (NAP). Le service peut être mis en œuvre à l'aide de la fonctionnalité de serveur WAMI de l'entité de gestion de station.

NOTE Un nœud transitoire peut utiliser ce service afin d'identifier le MAC ID du nœud au NAP duquel il est connecté.

La réponse Where_am_I doit être un USINT unique contenant le MAC ID du nœud permanent auquel le nœud transitoire est connecté. Si l'objet ControlNet n'est pas en mesure de terminer la requête WAMI, il doit renvoyer un MAC ID de 255.

#### 7.2.5.5 Service Auto_Address

Pour les nœuds qui prennent automatiquement en charge la sélection d'un MAC ID, ce service doit réinitialiser le NAM afin de surveiller l'activité de liaison, puis tenter de joindre la liaison avec un nouveau MAC ID, si possible. Les nœuds qui ne prennent pas automatiquement en charge la sélection d'un MAC ID ne doivent pas mettre en œuvre ce service.

Le service Auto_Address peut ne pas répondre avant la recherche d'un nouveau MAC_ID.

### 7.2.6 Comportement

#### 7.2.6.1 Effet de DLL_tminus_zero_indication

A la réception de cette indication, l'objet doit copier le contenu de l'attribut d'instance 0x80 (pending_link_config) dans l'attribut d'instance 0x81 (current_link_config). Une copie interne de l'attribut d'instance 0x80 doit être conservée, même si la mise en œuvre ne l'a pas rendue accessible par l'intermédiaire des services Get_Attribute.

#### 7.2.6.2 Code de duplication

Si un DLL_event_indicaton(DLL_EV_dupNode) est reçu de la couche de liaison de données, l'objet ControlNet doit forcer la DLL à passer à l'état d'écoute seulement. L'objet ControlNet doit conserver cet état tant qu'un service de réinitialisation n'a pas été reçu. Une réinitialisation de l'objet Identity doit également mettre fin à l'état d'écoute seulement.

#### 7.2.6.3 Redondance

Une valeur doit être attribuée au paramètre de redondance du réseau d'un nœud au démarrage afin d'obtenir les meilleures capacités de l'appareil: le canal A pour les appareils à canal unique, les deux canaux pour les appareils redondants du réseau. Le paramètre de redondance de mise sous tension doit apparaître dans le TUI de l'attribut d'instance 0x82.

L'objet Keeper doit mettre à jour le paramètre de redondance du réseau d'un nœud avec le paramètre en cours de l'une des deux manières suivantes:

– Le nœud reçoit un Lpacket TUI du réseau contenant l'objet Keeper maître, les bits de l'objet Keeper configuré et le bit de variation nette en cours réinitialisé dans le mot Status_Flags;
– Le nœud reçoit un Lpacket 0x81 à balise fixe.

Dans tous les cas, les données TUI reçues doivent être copiées dans l'attribut 0x80.

Si un nœud à canal unique rejoint une liaison redondante, il doit se comporter comme un nœud redondant avec un canal défaillant.

### 7.2.7 Voyant d'état du module

Le voyant d'état du module, s'il est mis en œuvre, doit apporter les indications d'état suivantes:

a) vert fixe lorsque les connexions sont actives dans l'état *en ligne*;
b) vert clignotant/désactivé à l'état de *vérification de l'écoute du modérateur uniquement*;
c) vert clignotant/désactivé lorsque des conditions suspectes sont détectées;
d) rouge clignotant/désactivé en cas d'erreur réparable;
e) rouge clignotant/désactivé à l'état DUP_LISTEN_ONLY;
f) rouge fixe en cas d'erreur irréparable.

## 7.3 Objet Keeper

### 7.3.1 Présentation

L'objet Keeper doit détenir tous les attributs de liaison de tous les appareils d'une liaison et doit être chargé de les distribuer aux appareils de manière ordonnée. Il détient également (pour le logiciel de planification de liaison) un exemplaire des données de l'auteur de la connexion de tous les appareils à l'origine de la connexion utilisant un réseau. Si plusieurs objets Keeper sont présents sur une liaison, ils procèdent à des négociations afin de déterminer celui qui va devenir maître et ceux qui vont assurer les fonctions de sauvegarde. Le Keeper maître est celui qui distribue activement les attributs aux nœuds du réseau. Un objet Keeper de sauvegarde est celui qui surveille l'activité réseau associée au Keeper et qui peut assurer le rôle d'objet Keeper maître en cas de défaillance du Keeper maître.

L'objet Keeper contient également un support pour l'obtention, la gestion et la délivrance de la ressource réseau, sémaphore réseau utilisé pour éliminer les conflits lors de la modification des données d'attribut détenues par le/les objet(s) Keeper sur une liaison.

### 7.3.2 Historique de révision

L'historique de révision de l'objet Keeper est décrit dans le Tableau 21.

**Tableau 21 – Historique de révision de l'objet Keeper**

| Révision de la classe | Description des modifications |
|---|---|
| **01** | Emission initiale |
| **02** | Emission pour la série CEI 61158 |

### 7.3.3 Attributs de classe

L'objet Keeper doit prendre en charge les attributs de classe tel que spécifié dans le Tableau 22.

**Tableau 22 – Attributs de classe de l'objet Keeper**

| Numéro | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| **0x1** | Obligatoire | Get | Revision | UINT | Révision de cet objet | 2 (révision 2) (Voir Note 2) |
| NOTE 1 Les appareils Revision 1 Keeper peuvent fonctionner uniquement si MAC ID = 1. Ces appareils exécutent les fonctions de l'objet Keeper conformément à la présente spécification tant qu'un autre objet Keeper n'est pas présent sur le réseau. Avec un autre MAC ID, les appareils contenant les objets Keeper – révision 1 fonctionnent comme si aucun objet n'était présent.<br><br>NOTE 2 Les appareils Revision 2 Keepers fonctionnent avec n'importe quelle valeur MAC ID légale. Ces appareils exécutent les fonctions de l'objet Keeper conformément à la présente spécification en présence d'autres objets Revision 2 Keeper sur le réseau. | | | | | | |

### 7.3.4 Attributs d'instance

#### 7.3.4.1 Généralités

L'objet Keeper doit prendre en charge les attributs d'instance tel que spécifié dans le Tableau 23. Une seule instance de l'objet Keeper est admise. Cette instance doit être sensible au port sur lequel la demande a été reçue. Cela est particulièrement important en présence d'un routeur sur lequel une instance Keeper reçoit des demandes provenant de deux liaisons.

**Tableau 23 – Attributs d'instance de l'objet Keeper**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| 0x00 | Obligatoire | Get | état | STRUCT de 2 octets | Etat de l'objet Keeper | |
| | | | état | USINT | Etat de fonctionnement de l'objet Keeper en cours | |
| | | | réservé | USINT | réservé pour l'alignement des données | |
| 0x01 à 0x63 | Obligatoire | Get/Set | port_status | STRUCT de 10 octets | Etat du port des nœuds 1 à 99 | |
| | | | port_status | UINT | Etat du port | |
| | | | ID | STRUCT de 8 octets | Identification du type de nœud | |
| | | | vendor | UINT | code fournisseur | |
| | | | type | UINT | type de produit | |
| | | | code | UINT | code produit | |
| | | | major | USINT | révision majeure | |
| | | | minor | USINT | révision mineure | |
| 0x64 – 0xFE | Réservé | | | | | Réservé pour extension ultérieure |
| 0xFF | Obligatoire | Get/Set | net_config | STRUCT de 12 octets | paramètres de réseau en cours | |
| | | | NUT | UINT | Durée de mise à jour du réseau | Dans des impulsions de 10 µs |
| | | | smax | USINT | ID nœud max planifié | 0 à 99 |
| | | | umax | USINT | ID nœud max non planifié | 0 à 99 |
| | | | Slot_Time | USINT | Intervalle | Dans des impulsions de 1 µs |
| | | | Blank_Time | USINT | Temps de blocage | En octets |
| | | | Gb_Start | USINT | Début de la bande de garde | Dans des impulsions de 10 µs |
| | | | Gb_Center | USINT | Centre de la bande de garde | Dans des impulsions de 10 µs |
| | | | Réservé | UINT | Réservé pour l'alignement des données | |
| | | | Int_Cnt mod | USINT | Module de comptage interne (longueur de macrocycle) | 127 |
| | | | Gb_Prestart | USINT | Démarrage préalable de la bande de garde | Dans des impulsions de 10 µs |
| 0x100 | Obligatoire | Get/Set | nom | UINT [33] | nom actuel de cette liaison | 32 caractères unicode et un NULL unicode |
| 0x101 | Obligatoire | Get/Set | RT_TUI | STRUCT de 22 octets | Identifiant unique de table | |
| | | | unique_ID | UDINT | attribut CRC | |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| | | | status_flag | UINT | Bits de balise TUI | Voir 7.2.3.3 |
| | | | réservé | USINT[16] | réservé à l'admission d'un format commun avec l'attribut 0x102 | |
| 0x102 | Obligatoire | Get | Link_TUI | STRUCT de 22 octets | Identifiant unique de table (liaison en cours uniquement) | |
| | | | unique_ID | UDINT | attribut CRC | |
| | | | status_flag | UINT | Bits de balise TUI | Voir 7.2.3.3 |
| | | | Keeper_MAC_ID | USINT | MAC ID du nœud diffusant le TUI | Un MAC ID légal |
| | | | réservé | USINT | réservé pour l'alignement des données | |
| | | | Net_Resource_ Vendor_Id | UINT | ID fournisseur de l'objet détenant le Net Resource | |
| | | | Net_Resource_ Serial_Number | UDINT | Numéro de série de l'objet détenant le Net Resource | |
| | | | Net_Resource_ Class | UDINT | Classe de l'objet détenant le Net Resource | |
| | | | Net_Resource_ Instance | UDINT | Instance de l'objet détenant le Net Resource | |
| 0x103 | Obligatoire | Get/Set | Cable_Config | STRUCT de 100 octets au maximum | Configuration de câble en cours | |
| | | | Id | USINT | Valeur de l'ID | Toujours 0xFF |
| | | | Num_Elements | USINT | Nombre d'éléments de configuration de câble dans une configuration de réseau | Compris entre 1 et 24 |
| | | | Propagation_time | UINT | Nombre d'impulsions 100 n s | |
| | | | Elément physique | Elément Phy [24] | | |
| | | | Phy_element | STRUCT de 4 octets | | |
| | | | Vendor_id | UINT | Code fournisseur | |
| | | | Product_code | USINT | Code produit | |
| | | | How_many | USINT | | |
| 0x104 | Obligatoire | Get/Set | CO_summary | STRUCT de 204 octets | Informations générales relatives à l'attribut *co_data* | |
| | | | data_size | UINT | Taille de l'attribut *co_data* | En mots |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| | | | connection_info_ revision | USINT | | Utilisé par les outils de programmation<br>0 = Format 1<br>1 = Format 2<br>2 – 255 réservés |
| | | | tool_keeper_ revision | USINT | Règle d'analyse syntaxique co_data de niveau le plus élevé prise en charge | Niveau pris en charge δ ≤révision de l'objet Keeper<br>0 = Format 1<br>1 – 255 réservés<br>Voir 7.3.6.7 |
| | | | 0ffsets | UINT [100] | Matrice de décalages de mot dans l'attribut *co_data*, par MAC ID | En mots 0xFFFF = aucune donnée pour ce nœud 0 MAC ID selon l'indice |
| 0x105 | Obligatoire | Fragment Get/Set | CO_data | STRUCT de 14 988 octets au maximum | Informations sur l'auteur de la connexion | Format 1 de l'outil Keeper Revision |
| | | | Branch | STRUCT de taille variable | Informations relatives à tous les routeurs ou auteurs de la connexion sur cette liaison | |
| | | | num_devices | UINT | Nombre d'appareils sur cette liaison | |
| | | | appareil | STRUCT de taille variable | détails de cet appareil | |
| | | | type | USINT | types d'appareil | 1 = routeur<br>2 = auteur de la connexion |
| | | | Path_Size | USINT | taille du chemin vers le nœud | En mots de 16 bits |
| | | | Chemin | MATRICE d'UINT | Chemin vers l'appareil | |
| | | | CO_data | STRUCT de taille variable | Détails des données CO | Présent uniquement pour les appareils CO (type = 2) |
| | | | COP | UDINT | Mot de passe CO, tel qu'indiqué au CO à la fin d'une session planifiée (voir 7.4.5.8) | |
| | | | Taille | UINT | Taille de la connexion | En mots de 16 bits |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| | | | Connexion | STRUCT de taille variable | Données de connexion | Révision d'informations de connexion au format spécifié par connection_ info_revision de l'attribut 0x104<br><br>Une par connexion pour cet appareil CO. Voir 7.3.4.7 |

#### 7.3.4.2 Définitions de l'état de fonctionnement

Les valeurs de l'état de fonctionnement de l'objet Keeper doivent être telles que définies dans le Tableau 24.

**Tableau 24 – Définitions de l'état de fonctionnement de l'objet Keeper**

| Valeur | Description |
|---|---|
| 0 | Mise sous tension – Pas en ligne |
| 1 | Mise sous tension – Attente TUI |
| 2 | Mise sous tension – Interrogation TUI |
| 3 | Sauvegarde |
| 4 | Vérification du maître |
| 5 | Maître |
| 6 | Echec de la sauvegarde |
| 7 | Echec de la vérification du maître |
| 8 | Echec du maître |
| 9 | Variation nette – Sauvegarde |
| 10 | Variation nette – Maître |
| 11 | Variation nette – Echec de la sauvegarde |
| 12 | Variation nette – Echec du maître |

#### 7.3.4.3 Définitions de bit de balise d'état du port

Les valeurs de l'état de fonctionnement de l'objet Port doivent être telles que définies dans le Tableau 25.

**Tableau 25 – Définitions de bit de balise d'état du port**

| Bit | Signification |
|---|---|
| 0 | Le nœud accepte le bit de connexion planifiée<br>0 = Le nœud n'accepte pas les connexions planifiées<br>1 = Le nœud accepte les connexions planifiées |
| 1 – 15 | réservé (valeur 0 attribuée) |

#### 7.3.4.4 status_flag

Le bit configuré de l'objet Keeper annulé dans le mot d'état TUI doit identifier un objet Keeper doté d'une configuration non valide ou d'une mémoire effacée.

Conformément au Tableau 26, le champ status_flag du sous-attribut TUI doit être composé d'une valeur de 16 bits. Les bits de balise non utilisés doivent être réservés et définis sur 0.

Le bit de gestion de ressource réseau doit uniquement être utilisé par l'objet Keeper.

Le bit de l'objet Keeper maître doit indiquer si ce nœud a déjà entendu ou pas un objet Keeper maître. L'objet ControlNet doit annuler ce bit lors de l'initialisation. Une valeur doit être attribuée à ce bit de tous les TUI envoyés par l'objet Keeper maître. Si un Lpacket TUI est reçu et qu'aucune valeur n'a été attribuée au bit de l'objet Keeper maître, le Lpacket doit être conservé comme étant un TUI reçu, mais les informations de redondance ne doivent pas être utilisées.

Les bits de redondance doivent être utilisés pour remplacer les paramètres de redondance par défaut pour le nœud. Ils doivent indiquer si la liaison est en cours de fonctionnement en mode de canal unique ou redondant. Les informations de redondance ne doivent pas être utilisées si un Lpacket TUI est reçu et que le bit de l'objet Keeper maître est absent.

Le bit de variation nette en cours doit indiquer qu'une séquence de modification de réseau synchronisée est en cours et que de nouveaux nœuds doivent retarder leur mise en ligne tant que la séquence de modification n'est pas terminée et que le bit n'est pas supprimé. Ce bit doit être annulé en cas de dépassement de délai de ressource réseau ou si la ressource réseau est libérée.

**Tableau 26 – Bits de balise d'état TUI**

| Bit | Signification | Utilisé dans: | | |
|---|---|---|---|---|
| | | RT_TUI | Network_TUI | TUI de l'objet ControlNet |
| 0 – 1 | Redondance du réseau<br>Bit 1 = 0, bit 0 = 0; Combinaison illégale<br>Bit 1 = 0, bit 0 = 1; Canal B uniquement<br>Bit 1 = 1, bit 0 = 0; Canal A uniquement<br>Bit 1 = 1, bit 0 = 1; Redondant | X | X | X |
| 2 | Bit de gestion des ressources réseau<br>0 = Ressources réseau non gérées<br>1 = Ressources réseau gérées pour l'objet identifié dans le TUI | | X | |
| 3 | Bit de modification en cours du réseau<br>0 = Pas de modification de réseau en cours<br>1 = Modification de réseau en cours | X | X | X |
| 4 | Bit d'état de l'objet Keeper<br>0 = Le TUI transmis par l'objet Keeper n'est pas à l'état MAITRE<br>1 = Le TUI transmis par l'objet Keeper est à l'état MAITRE. | | X | X |
| 5 | Bit configuré de l'objet Keeper<br>0 = Attributs Keeper non configurés (RT_TUI)<br>1 = Attributs Keeper configurés (RT_TUI) | X | X | X |
| 6 – 15 | réservé (valeur 0 attribuée) | | | |

#### 7.3.4.5 unique_ID

Le champ unique_ID de l'attribut TUI doit être composé d'une valeur de 32 bits calculée en fonction du contenu du tableau de l'attribut Keeper (outre l'attribut TUI) par l'outil de configuration logicielle. Les attributs suivants doivent être utilisés, dans l'ordre donné pour calculer l'attribut unique_ID TUI:

0x1 à 0x63 (paramètres de port des nœuds 1 à 99)

0xFF (paramètres de liaison)

0x100 (nom de liaison);

0x101 (RT TUI) bits de redondance uniquement;

0x103 (configuration de câble);

0x105 (données de mot de passe CO uniquement);

L'identifiant unique TUI doit être calculé comme suit:

– l'identifiant unique TUI doit être remis à 0;

– Les attributs de données 0x1 à 0x63, l'attribut 0xFF, l'attribut 0x100, le paramètre de redondance (attribut 0x0101, balise d'état RT_TUI, bits 0-1), 0x103 et chaque mot de passe CO (dans l'ordre indiqué) doivent être exécutés par l'intermédiaire du polynôme CRC.

Le polynôme utilisé pour calculer le CRC doit être
$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + 1$
(voir Figure 17).

```
UDINT CRC(                    // (r) the updated CRC
      unsigned long CRC,  // (i) the CRC to start with
      SWORD* buffer,          // (i) pointer to buffer of SWORD/octets
      WORD size)              // (i) the number of octets in the buffer

#   define KEEPER_CRC_POLY 0xEDB88320L
   WORD octetIndex;
   WORD bitIndex;
   SWORD oneOctet;

// loop through the octets passed, including them in the CRC
   for (octetIndex=0; octetIndex < size; octetIndex++)
   {
      oneOctet = buffer[octetIndex];

// loop through the bits passed, including them in the CRC
      for (bitIndex = 0; bitIndex < 8; bitIndex++)
      {
         if ((oneOctet & 0x1) ^ (CRC & 0x1))
         {
            CRC = (CRC >> 1) ^ KEEPER_CRC_POLY;
         } else {
            CRC = (CRC >> 1);
         }
         oneOctet >>= 1;
      }
   }

// return the new CRC
   return(CRC);
}
```

**Figure 17 – Algorithme CRC de l'objet Keeper**

#### 7.3.4.6 MAC_ID Keeper

Il convient que le Keeper_MAC_ID soit le MAC ID de l'objet Keeper maître en cours diffusant le TUI. Si aucun TUI n'a été reçu au cours des 12 dernières NUT, une valeur de 0xFF doit être renvoyée.

#### 7.3.4.7 Connexion

Il existe différents formats de données de connexion. Chacun d'eux est décrit ci-dessous. L'ensemble des entrées de données de connexion doit être un nombre entier de mots.

Le format 1 ne précise pas la taille du chemin. Par conséquent, son chemin [ ] doit se présenter sous deux formes:

– segment de classe, segment d'instance, puis deux segments de point de connexion; ou

– un segment de symbole étendu ANSI.

```
// Format 1
  USINT  O2T_Size     // in 16 bit words
  UINT   O2T_Rpi      // in 1 ms ticks from 2 to 32 767
  USINT  T2O_Size     // in 16 bit words
  UINT   T2O_RPI      // bit 15 = connection point, 0 = point to point, 1 = multipoint
                      // bit 14 = API in ms from 2 to 32 767
  USINT  O2T_Nui_Api  // The highest bit set indicates the API;
                      //   bit 7 = 128*NUT, bit 6 = 64*NUT, etc.
                      // Remaining bits indicate the O2T_NUI.  If API = 1, NUI = 0
  USINT  T_Node       // Range 1 – 99
  USINT  T2O_Nui_Api  // The highest bit set indicates the API;
                      //   bit 7 = 128*NUT, bit 6 = 64*NUT, etc.
                      // Remaining bits indicate the O2T_NUI.  If API = 1, NUI = 0
  UINT   path[]       // forward open connection path from this
```

```
// Format 2
  UINT   O2T_net_params  // copied from forward open
  UINT   T2O_net_params  // copied from forward open
  UINT   O2T_RPI         // RPI in floating point format
  UINT   T2O_RPI         // RPI in floating point format
  USINT  O2T_schedule    // from schedule segment in forward open
  USINT  MAC_ID
  USINT  T2O_schedule    // from schedule segment in forward open
  USINT  path_size       // in 16-bit words
  UINT   path[]          // forward open connection path from this
```

```
UDINT convert_to_10us_ticks (UINT RPI)
{
   return (RPI <= 0x4000) ? RPI:
          (RPI <= 0xE000) ? (RPI & 0x1FFF) + 0x2000 << (RPI>>13) - 1:
                          : (RPI & 0x0FFF) + 0x1000 << (RPI>>12 & 1) + 7;
}

UINT convert_to_Keeper_format (UDINT ticks)
{
   if (ticks < 0x00004000) return RPI;                        //       0 - 16383 by 1
   if (ticks < 0x00008000) return RPI>>1 & 0x1FFF | 0x4000; //   16384 - 32766 by 2
   if (ticks < 0x00010000) return RPI>>2 & 0x1FFF | 0x6000; //   32768 - 65532 by 4
   if (ticks < 0x00020000) return RPI>>3 & 0x1FFF | 0x8000; //   65536 - 131064 by 8
   if (ticks < 0x00040000) return RPI>>4 & 0x1FFF | 0xA000; //  131072 - 262128 by 16
   if (ticks < 0x00080000) return RPI>>5 & 0x1FFF | 0xC000; //  262144 - 524256 by 32
   if (ticks < 0x00100000) return RPI>>7 & 0x0FFF | 0xD000; //  524288 - 1048448 by 128
   if (ticks < 0x00200000) return RPI>>8 & 0x0FFF | 0xE000; // 1048576 - 2096896 by 256
   return 0xFFFF;
)
```

#### 7.3.4.8 Données d'attribut

Un objet Keeper conserve la liaison et ses informations de configuration nœud dans une structure de données d'attribut. L'objet Keeper doit conserver deux exemplaires de la structure d'attribut: un exemplaire en attente dans la mémoire RAM et un exemplaire en cours dans la mémoire non volatile. Lorsque de nouveaux attributs sont définis dans l'objet Keeper, ils doivent l'être dans l'exemplaire en attente dans la mémoire RAM. L'exemplaire RAM doit

être écrit dans l'exemplaire actuellement actif de la mémoire non volatile dès la réception d'une demande spéciale.

Les attributs doivent s'appuyer sur les règles suivantes pour l'obtention et la définition:

– Les demandes de service de définition définissent l'exemplaire en cours dans la mémoire RAM;
– Les demandes de service d'obtention obtiennent l'exemplaire en cours dans la mémoire non volatile.

L'objet Keeper doit conserver l'ensemble d'attributs spécifié dans le Tableau 27:

**Tableau 27 – Attributs Keeper**

| Numéro d'attribut | Attributs Keeper |
|---|---|
| 0x01 – 0x63 | (paramètres de port des nœuds 1 à 99) |
| 0xFF | (paramètres de liaison) |
| 0x100 | (nom de liaison) |
| 0x101 | (Keeper TUI) |
| 0x103 | (configuration de câble) |
| 0x104 | (informations de décalage de nœud dans les données CO) |
| 0x105 | (chemin CO et informations sur le mot de passe) |

Toutes les définitions d'attribut Keeper supposent la présence de 99 nœuds au maximum sur la liaison. L'objet Keeper doit disposer d'une mémoire suffisante pour stocker les attributs CO_summary et CO_data.

Les exigences en matière de mémoire (en octets) des attributs Keeper sont spécifiées dans le Tableau 28.

**Tableau 28 – Exigences en matière de mémoire (en octets) des attributs Keeper**

| Octets | Attributs Keeper |
|---|---|
| 2 | Etat Keeper |
| 990 | Paramètres de port (99 nœuds de 10 octets chacun) |
| 12 | paramètres de liaison |
| 66 | nom de liaison |
| 22 | TUI |
| 100 | Configuration de câble |
| 204 | Récapitulatif CO |
| 14 988 | Données CO (7 494 mots) |
| 16 384 | **Total** (Exactement 16 Ko) |

La séquence d'événements requise pour mettre à jour les attributs Keeper correctement et en toute sécurité doit se présenter comme suit:

a) obtenir la ressource réseau;
b) lancer une demande de service Change_Start à l'objet Keeper;

c) lancer la/les demande(s) de service Set_Attribute appropriée(s) afin de mettre à jour l'exemplaire en attente des attributs Keeper;

d) lancer une demande de service Change_Complete à l'objet Keeper. Cette procédure doit initier une modification de paramètre synchronisé;

e) libérer la ressource réseau.

### 7.3.5 Services communs

Les services communs Keeper doivent tous fonctionner avec un temps interne afin d'éviter qu'une condition d'erreur matérielle ne bloque indéfiniment les opérations. Les valeurs réelles de délai d'attente doivent dépendre de l'appareil.

L'objet Keeper doit prendre en charge les services communs spécifiés dans le Tableau 29.

**Tableau 29 – Services communs de l'objet Keeper**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x03 | N/A | Obligatoire | Get_Attribute_List | Obtient les informations de configuration spécifiques au réseau et au nœud |
| 0x04 | N/A | Obligatoire | Set_Attribute_List | Définit les informations de configuration spécifiques au réseau et au nœud |
| 0x0E | Obligatoire | Obligatoire | Get_Attribute_Single | Obtient les informations de configuration spécifiques au réseau et au nœud |
| 0x10 | N/A | Obligatoire | Set_Attribute_Single | Définit les informations de configuration spécifiques au réseau et au nœud |

L'objet Keeper est en mesure de recevoir ces demandes de service par l'intermédiaire des balises fixes 83 et 88. S'il les reçoit par la balise fixe 83, la demande est traitée et fait l'objet d'une réponse indépendamment de l'état de l'objet Keeper. S'il les reçoit par la balise fixe 88, l'objet Keeper répondra uniquement à la demande s'il est à l'état Maître, Variation nette – Maître, Echec du maître ou Variation nette – Echec du maître. Les nœuds Keeper se trouvant dans d'autres états ne doivent pas répondre à la demande.

### 7.3.6 Services spécifiques à la classe

#### 7.3.6.1 Généralités

Tous les appareils ayant mis en œuvre l'objet Keeper doivent également mettre en œuvre le Keeper UCMM.

L'objet Keeper doit prendre en charge les attributs de classe suivants tel que spécifié dans le Tableau 30.

**Tableau 30 – Services spécifiques à la classe de l'objet Keeper**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service | Paramètre(s) |
|---|---|---|---|---|---|
| | Classe | Instance | | | |
| 0x4B | N/A | Obligatoire | Obtain_ Network_Resource (ONR) | Tentative d'obtention de la ressource réseau pour cette liaison.<br>Si la tentative aboutit, maintenir pendant 60 s | UINT ID fournisseur<br>UDINT numéro de série<br>UDINT classe<br>UDINT instance |
| 0x4C | N/A | Obligatoire | Hold_ Network_Resource (HNR) | Continuer à maintenir la ressource réseau précédemment obtenue pour cette liaison pendant 60 s | UINT ID fournisseur<br>UDINT numéro de série<br>UDINT classe<br>UDINT instance |
| 0x4D | N/A | Obligatoire | Release_ Network_Resource (RNR) | Libérer la ressource réseau pour cette liaison | UINT ID fournisseur<br>UDINT numéro de série<br>UDINT classe<br>UDINT instance |
| 0x4E | N/A | Obligatoire | Change_Start (CS) | Copier les attributs en cours de la mémoire non volatile dans les attributs en attente de la mémoire RAM.<br>Entrer dans l'état de fonctionnement de *variation nette* approprié | <aucun> |
| 0x4F | N/A | Obligatoire | Change_Complete (CC) | Copier les données d'attribut en attente de la mémoire RAM dans la mémoire non volatile.<br>Reprendre le fonctionnement normal de l'objet Keeper.<br>Lancer une modification de réseau synchronisée si les attributs nets ont été modifiés.<br>Si le nombre d'ensembles ne correspond pas, procéder à un traitement *change_abort* | Nombre d'« ensembles » à 16 bits exécutés depuis *Change_Start* |
| 0x50 | N/A | Obligatoire | Change_Abort (CA) | Ignorer les modifications apportées aux données d'attribut en attente, libérer la ressource réseau, puis rétablir le fonctionnement normal de l'objet Keeper | <aucun> |
| 0x51 | N/A | Obligatoire | Get_Signature (GS) | Obtenir la valeur de signature d'un auteur de connexion particulier (mot de passe CO) | Demande: taille de chemin (octet) (la taille est en mots) chemin<br>Réponse: cop 32 bits |
| 0x52 | N/A | Obligatoire | Get_Attribute_ Fragment (GAF) | Obtenir une partie d'un attribut spécifiquement conçu pour être fragmenté et trop volumineux pour entrer dans un seul Lpacket.<br>Utilisation réservée à l'attribut co_data | Taille UINT; //words<br>Décalage UINT; //offset |

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service | Paramètre(s) |
|---|---|---|---|---|---|
| | Classe | Instance | | | |
| 0x53 | N/A | Obligatoire | Set_Attribute_ Fragment (SAF) | Définir une partie d'un attribut spécifiquement conçu pour être fragmenté et trop volumineux pour entrer dans un seul Lpacket.<br>Utilisation réservée à l'attribut CO_data | Taille UINT; //words<br>Décalage UINT; //offset<br>Données UINT [ ] |

Les codes d'erreur de ces services doivent être ceux spécifiés dans le Tableau 31, les codes de service étant les abréviations des noms de service spécifiés dans le Tableau 30.

**Tableau 31 – Codes d'erreur du service**

| Code d'erreur | Signification | Code d'erreur utilisé par le service | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | ONR | HNR | RNR | CS | CC | CA | GS | GAF | SAF |
| 0x02 | resource_unavailable_err | X | X | X | X | | | | | |
| 0x03 | invalid_parameter_err | | | | | X | | X | X | X |
| 0x04 | path_segment_err | | | | | | | X | X | X |
| 0x05 | path_dest_unknown_err | X | X | X | X | X | X | X | X | X |
| 0x08 | unimplemented_service_err | X | X | X | X | X | X | X | X | X |
| 0x09 | invalid_attr_err | | | | | | | | X | X |
| 0x0C | wrong_object_state_err | | | | | X | X | | | X |
| 0x0D | already_exists_err | X | | | X | | | | | |
| 0x0F | no_permission_err | | X | X | | | | | | |
| 0x11 | reply_too_large_err | | | | | | | X | | |
| 0x13 | not_enough_data_err | X | X | X | | X | | X | X | X |
| 0x14 | undefined_attr_err | | X | | | | | | | |
| 0x15 | too_much_data_err | | | | | | | X | | X |
| 0x16 | nonexistant_object_err | X | X | X | X | X | X | X | X | X |
| 0x26 | invalid_path_size_err | X | X | X | X | X | X | X | X | X |

#### 7.3.6.2 Ressource réseau

La ressource réseau (NR) doit être un sémaphore de liaison permettant d'éliminer les conflits en cas de modification des informations d'attribut des objets Keeper.

En cours de fonctionnement, la ressource réseau doit apparaître sous la forme d'un bit de balise d'état et d'autres champs de données du Lpacket TUI qui doivent être diffusés de manière répétitive par l'objet Keeper maître.

Un appareil souhaitant mettre à jour les informations d'attribut de l'objet Keeper doit en premier lieu soumettre une demande à l'objet Keeper maître afin d'obtenir la ressource réseau pour son compte (par l'intermédiaire d'un service Obtain_Network_Resource). Cette demande doit être diffusée étant donné que l'identité de l'objet Keeper maître n'est en général pas connue. Une fois obtenu, l'appareil doit régulièrement demander (toutes les 15 s) à l'objet Keeper maître de continuer à gérer la ressource réseau par l'intermédiaire d'une demande de service Hold_Network_Resource. L'objet Keeper maître doit gérer un temporisateur de 60 s, qui est démarré dès l'obtention de la ressource réseau, puis de nouveau déclenché dès la réception d'une demande Hold_Network_Resource consécutive provenant du même nœud. Si

le délai du temporisateur expire, l'objet Keeper maître doit automatiquement libérer la ressource réseau.

Un objet Keeper de sauvegarde doit répondre aux demandes de service de ressource réseau comme suit. En cas de réception d'une demande de service *Obtain_Network_Resource*, un temporisateur de 60 s doit commencer, puis être déclenché lors de la réception d'une demande de service *Hold_Network_Resource*. Le temporisateur doit également être démarré lorsqu'un objet Keeper est activé sur un réseau existant et qu'il constate que la ressource réseau est en cours d'obtention. Le temporisateur doit être arrêté lorsqu'un service *Release_Network_Resource* est reçu ou que le temporisateur expire.

Lorsqu'un objet Keeper de sauvegarde devient un objet Keeper maître, il doit utiliser les données provenant du dernier Lpacket TUI reçu afin de récupérer la ressource réseau de l'ancien objet Keeper maître.

Eu égard aux objets Keeper maître et de sauvegarde, le temporisateur de 60 s doit permettre de déterminer si le nœud qui demande la ressource réseau n'est pas tombé en panne ou a été prématurément retiré du réseau. Lorsque le temporisateur de 60 s d'un objet Keeper expire, l'objet Keeper doit

a) annuler toutes les modifications d'attribut en cours;

b) réinitialiser le bit d'état TUI de variation nette en cours;

c) quitter l'état de fonctionnement de variation nette;

d) reprendre le fonctionnement normal.

Etant donné que la ressource réseau peut être conservée pendant une période assez longue si l'utilisateur procède à des modifications importantes, elle doit être clairement transférée vers le nouvel objet Keeper maître en cas de modification de l'objet Keeper maître. Ce transfert doit être invisible au nœud pour lequel l'objet Keeper maître gère la ressource réseau. Etant donné que toutes les informations relatives à la ressource réseau sont contenues dans le Lpacket TUI que l'objet Keeper maître diffuse, et que les objets Keeper de sauvegarde reçoivent le TUI diffusé et l'utilisent pour déterminer s'il convient qu'ils jouent le rôle d'objet Keeper maître et à quel moment ils doivent le faire, les objets Keeper de sauvegarde doivent détenir les informations nécessaires pour transmettre correctement la ressource.

L'objet Keeper de sauvegarde et l'objet Keeper maître défaillants doivent traiter les demandes de ressource réseau comme s'ils ne l'étaient pas.

Si la ressource réseau n'est pas traitée, les champs connexes des Lpackets TUI de réseau doivent être réinitialisés:

– NR VENDOR_ID;
– NR SERIAL_NUMBER;
– NR CLASS;
– NR_INSTANCE.

#### 7.3.6.3 Service Obtain_Network_Resource

La demande de service Obtain_Network_Resource doit permettre à l'objet Keeper maître de gérer la ressource réseau pendant 60 s pour lui-même ou un autre nœud, s'il n'est pas déjà géré. La période de gestion de la ressource réseau peut être étendue de manière indéfinie par l'utilisation de la demande de service Hold_Network_Resource.

Si la ressource réseau est déjà gérée pour un autre nœud, un état d'erreur doit être renvoyé.

Le service Obtain_Network_Resource doit renvoyer les codes d'erreur (voir Tableau 31) comme indiqué dans la colonne ONR pour ce service.

#### 7.3.6.4 Service Hold_Network_Resource

La demande de service Hold_Network_Resource doit permettre à l'objet Keeper maître de continuer à gérer la ressource réseau pendant les 60 s qui suivent si le nœud qui le demande est celui pour lequel l'objet Keeper maître est en train de gérer la ressource réseau.

Si la ressource réseau est gérée pour un autre nœud ou si elle n'est pas gérée, un état d'erreur doit être renvoyé.

Le service Hold_Network_Resource doit renvoyer les codes d'erreur (voir Tableau 31) comme indiqué dans la colonne HNR pour ce service.

#### 7.3.6.5 Service Release_Network_Resource

La demande de service Release_Network_Resource doit permettre à l'objet Keeper maître d'arrêter la ressource réseau si le nœud qui demande la libération est celui pour lequel l'objet Keeper maître gère la ressource réseau. Si une modification de réseau est en cours et que la ressource réseau est libérée, la modification sera annulée.

Que la ressource réseau soit gérée pour un autre nœud ou qu'elle ne soit pas gérée, un état d'erreur doit être renvoyé.

Le service Release_Network_Resource doit renvoyer les codes d'erreur (voir Tableau 31) comme indiqué dans la colonne RNR pour ce service.

#### 7.3.6.6 Service Change_Start

La demande de service Change_Start doit permettre à l'objet Keeper:

a) de copier l'exemplaire en cours des attributs de la mémoire non volatile dans l'exemplaire en attente de la mémoire RAM;

b) d'annuler le nombre de demandes de service Set_Attribute (y compris individuel, liste et fragment) reçues par l'objet ControlNet;

c) de placer l'objet Keeper à l'état de fonctionnement net_change approprié.

Cette demande de service doit uniquement être traitée si la ressource réseau est gérée par l'objet Keeper maître comme indiqué par le bit net_resource dans le Lpacket TUI diffusé.

Le service Change_Start doit renvoyer les codes d'erreur (voir Tableau 31) comme indiqué dans la colonne CS pour ce service.

#### 7.3.6.7 Service Change_Complete

La demande de service Change_Complete doit permettre à l'objet Keeper de:

a) copier l'exemplaire en attente des attributs de la mémoire RAM vers l'exemplaire en cours des attributs de la mémoire non volatile;

b) procéder à une modification de réseau synchronisée;

c) vérifier que tool_keeper_revision de l'attribut 105 est $\delta$ révision de l'objet Keeper (si ce n'est pas le cas, marquer le tableau d'attribut comme étant non valide);

d) quitter l'état de fonctionnement de modification de réseau dans lequel il se trouve;

e) reprendre le fonctionnement Keeper normal.

Si l'objet Keeper ne se trouve pas dans l'un des états de fonctionnement de variation nette, la demande de service Change_Complete doit être ignorée et une réponse d'erreur d'état incorrect générée.

Si le nombre de demandes de service Set_Attribute (notamment Set_Attribute_Single, Set_Attribute_List et Set_Attribute_Fragment) reçues par l'objet Keeper depuis la demande de service Change_Start ne correspond pas à celui du paramètre Change_Complete, un Change_Abort doit être exécuté et une réponse d'erreur générée.

La réponse à ce service doit être renvoyée après la diffusion d'au moins un TUI, le bit net_change_in_progress étant annulé. Une modification synchronisée doit avoir lieu avant la transmission TUI avec le bit remis à zéro.

Le service Change_Complete doit renvoyer les codes d'erreur (voir Tableau 31) comme indiqué dans la colonne CC pour ce service.

#### 7.3.6.8 Service Change_Abort

Le service Change_Abort doit permettre différentes opérations à l'objet Keeper:

a) écarter les modifications apportées aux données d'attribut en attente;

b) libérer la ressource réseau;

c) reprendre le fonctionnement Keeper normal.

La réponse à ce service doit être renvoyée après la diffusion d'au moins un TUI, avec le bit net_change_in_progress, et le bit holding_network_resource étant annulé.

Le service Change_Abort doit renvoyer les codes d'erreur (voir Tableau 31) comme indiqué dans la colonne CA pour ce service.

#### 7.3.6.9 Service Get_Signature

Le service Get_Signature doit permettre à l'objet Keeper maître de consulter les valeurs de mot de passe CO de l'auteur d'une connexion dans l'attribut CO_data. Le chemin vers l'appareil doit être l'attribut de la demande. Ce chemin doit faire l'objet d'une analyse syntaxique et utilisé pour rechercher la structure en arborescence des données CO_data, une branche à la fois, afin de détecter la valeur du mot de passe CO approprié, qui est renvoyée en tant que paramètre dans la réponse.

Le chemin doit faire l'objet d'une analyse syntaxique une branche à la fois, étant donné que les entrées du chemin de l'attribut CO_data spécifient uniquement la partie du chemin entre deux branches, et pas l'ensemble du chemin vers cette branche.

Le service Get_Signature doit renvoyer les codes d'erreur (voir Tableau 31) comme indiqué dans la colonne GS pour ce service.

#### 7.3.6.10 Service/réponse Get_Attribute_Fragment

Le service Get_Attribute_Fragment doit collecter et renvoyer une partie des données d'attribut CO_data gérées par l'objet Keeper. Toutes les parties de cet attribut peuvent être lues. L'attribut CO_data doit être le seul attribut Keeper qui répond aux services de fragment. L'accès aux autres attributs doit générer une erreur à renvoyer.

Lors de l'obtention de données d'attribut, il convient de renvoyer les données d'attribut en cours issues de la mémoire non volatile, et pas l'exemplaire en attente de l'attribut.

Le service Get_Attribute_Fragment doit renvoyer les codes d'erreur (voir Tableau 31) comme indiqué dans la colonne GAF pour ce service.

#### 7.3.6.11 Service Set_Attribute_Fragment

Le service Set_Attribute_Fragment doit être utilisé pour définir une partie de l'attribut CO_data géré par l'objet Keeper. Seul l'attribut CO_data doit être accessible par ce service. L'accès aux autres attributs doit générer une erreur à renvoyer.

Le service Set_Attribute_Fragment doit être traité uniquement lorsque l'objet Keeper se trouve dans l'un des états de fonctionnement de variation nette. L'objet Keeper doit renvoyer une erreur de mode non valide s'il ne se trouve pas dans l'un des états de fonctionnement de variation nette lors de la réception d'une demande de service Set_Attribute_Fragment.

La définition des données d'attribut doit définir l'exemplaire en attente de l'attribut, pas l'exemplaire en cours de la mémoire non volatile. Les attributs en attente doivent être copiés dans les attributs en cours de la mémoire non volatile dès la réception d'une demande de service Change_Complete spécifique à la classe.

L'objet Keeper ne doit pas vérifier la valeur d'attribut pendant une opération de définition.

Le service Set_Attribute_Fragment doit renvoyer les codes d'erreur (voir Tableau 31) comme indiqué dans la colonne SAF pour ce service.

#### 7.3.6.12 Identifiant unique de table (diffusion), balise fixe 0x84

Si l'objet Keeper est à l'état de fonctionnement master_verify, faulted_master_verify, master, faulted_master, net_change_master ou net_change_faulted_master, il doit tenter de transmettre un TUI (Identifiant unique de table) Lpacket particulier en tant que données planifiées une fois toutes les 4 NUT (c'est-à-dire les numéros NUT 0, 4, 8, 12, 16 ...). Tous les appareils Keeper doivent réserver 26 octets de durée de transmission de liaison planifiée une fois toutes les 4 NUT pour transmettre ce Lpacket. En cas d'envoi non planifié, il doit s'agir des informations non planifiées de priorité QoS la plus élevée.

Lors de la génération du TUI Lpacket transmis, tous les champs réservés doivent utiliser les valeurs spécifiées dans les champs réservés correspondants de l'attribut 0x0101. L'outil de programmation doit affecter une valeur nulle à tous les champs réservés de l'attribut 0x0101.

Le format des données de liaison TUI Lpacket doit être tel que défini dans le Tableau 32.

**Tableau 32 – Format de virement du TUI Lpacket**

| Nom | Type de données | Description du paramètre | Sémantique des valeurs |
|---|---|---|---|
| Taille | USINT | Taille du Lpacket, en mots. | 0x0D |
| Contrôle | USINT | Octet de contrôle du Lpacket de couche de liaison. | 0x01 (Lpacket à balise fixe) |
| Fixed_Tag | USINT | Valeur de balise fixe. | 0x84 (Lpacket TUI) |
| Destination_Mac_Id | USINT | Destinataire du Lpacket. | 0xFF (Diffusion) |
| Unique_Id | UDINT | CRC des attributs en cours de Keeper. | Octet de poids faible en premier. |
| Status_Flag | UINT | Valeurs de balise TUI. | Voir l'attribut TUI pour plus de détails. |
| Keeper_Mac_Id | USINT | MAC ID du Keeper diffusant le TUI. | Voir l'attribut TUI pour plus de détails. |
| Réservé | USINT | Réservé pour l'alignement des données | |
| Net_Resource_Vendor_Id | UINT | ID fournisseur de l'objet détenant le Net Resource pour une utilisation exclusive | |

| Nom | Type de données | Description du paramètre | Sémantique des valeurs |
|---|---|---|---|
| Net_Resource_Serial_Number | UDINT | Numéro de série de l'objet détenant le Net Resource pour une utilisation exclusive | |
| Net_Resource_Class | UDINT | Numéro de classe de l'objet détenant le Net Resource pour une utilisation exclusive | |
| Net_Resource_Instance | UDINT | Numéro d'instance de l'objet détenant le Net Resource pour une utilisation exclusive | |

### 7.3.7 Codes d'erreur de service

Le Tableau 33 répertorie les codes d'erreur possibles et les conditions les plus probables dans lesquelles le code est renvoyé.

**Tableau 33 – Codes d'erreur du service**

| Code d'erreur | Signification | Condition de renvoi |
|---|---|---|
| 0x02 | resource_unavailable_err | La ressource réseau demandée n'est pas disponible |
| 0x03 | invalid_parameter_err | Un paramètre non valide a été fourni au service |
| 0x04 | path_segment_err | A chaque spécification d'un chemin supérieur au numéro d'attribut |
| 0x05 | path_dest_unknown_err | L'instance d'objet n'existe pas |
| 0x08 | unimplemented_service_err | A chaque fois que le service n'est pas pris en charge pour un attribut |
| 0x09 | invalid_attr_err | A chaque fois qu'un attribut spécifié n'est pas pris en charge dans cet objet |
| 0x0C | wrong_object_state_err | L'état de l'objet ne permet pas d'exécuter le service |
| 0x0D | already_exists_err | Le service a déjà été obtenu par le nœud demandeur |
| 0x0E | not_settable_err | L'attribut ne peut pas être défini |
| 0x0F | no_permission_err | Le nœud demandant ce service ne détient pas la ressource |
| 0x11 | reply_too_large_err | Espace insuffisant dans la mémoire tampon de réponse |
| 0x13 | not_enough_data_err | A chaque fois que la demande ne contient pas assez de données |
| 0x14 | undefined_attr_err | Tentatives d'accès à un attribut non défini |
| 0x15 | too_much_data_err | Plus de données que prévu ont été rencontrées pour cette demande de service |
| 0x16 | nonexistant_object_err | Objet Keeper non disponible |
| 0x26 | invalid_path_size_err | A chaque fois qu'un segment de chemin non valide est spécifié pour ce service |

### 7.3.8 Comportement

L'indication numérique de l'état de l'objet Keeper doit être telle que définie dans le Tableau 34.

**Tableau 34 – Etats de fonctionnement de l'objet Keeper**

| Etat | Description |
|---|---|
| 0,1,2<br>- Mise sous tension | L'objet keeper attend la mise en ligne du nœud ou détermine s'il s'agit d'un objet Keeper légitime pour cette liaison |
| 3<br>- Sauvegarde | L'objet Keeper a déterminé qu'il s'agit d'un objet Keeper légitime pour cette liaison, mais a déterminé qu'il s'agit d'un objet Keeper de sauvegarde ou n'a pas encore déterminé s'il s'agit de l'objet Keeper maître pour cette liaison |

| Etat | Description |
|---|---|
| 4<br>- Vérification du maître | L'objet Keeper a déterminé qu'il s'agit d'un objet Keeper légitime pour cette liaison, mais n'a pas entendu d'autres objets Keeper actifs sur la liaison, a reçu un TUI correct provenant d'un objet Keeper correspondant avec un numéro de nœud supérieur ou a entendu un TUI provenant d'un objet Keeper défaillant. Il commence à diffuser le TUI Lpacket, mais ne traite pas les modifications de paramètre synchronisées ni ne répond aux demandes de l'objet Keeper maître |
| 5<br>- Maître | L'objet Keeper a déterminé qu'il s'agit d'un objet Keeper légitime et qu'il s'agit de l'objet Keeper maître pour cette liaison. Il diffuse le TUI Lpacket, traite les modifications de liaison synchronisées, le cas échéant, pour modifier les paramètres de liaison et doit répondre aux demandes de l'objet Keeper maître |
| 6<br>- Echec de la sauvegarde | L'objet Keeper a déterminé qu'il ne s'agit pas d'un objet Keeper légitime pour cette liaison, et qu'il n'est pas l'objet Keeper si la liaison ne contient que des objets Keeper défaillants |
| 7<br>- Echec de la vérification du maître | L'objet Keeper est défaillant, mais n'a pas entendu un autre objet Keeper actif sur le réseau pendant 12 NUT. Il commence à diffuser le TUI Lpacket, mais ne traite pas les modifications de liaison synchronisées ni ne répond aux demandes de l'objet Keeper maître |
| 8<br>- Echec du maître | L'objet Keeper est défaillant et a supposé les fonctions de l'objet Keeper maître. Le réseau ne contient aucun objet Keeper non défaillant. Il diffuse le TUI Lpacket, traite les modifications de liaison synchronisées, le cas échéant, pour modifier les paramètres de réseau et doit répondre aux demandes de l'objet Keeper maître |
| 9,10,11,12<br>- Variation nette | Les attributs de l'objet Keeper sont mis à jour. Il existe un sous-état *Variation* nette pour les états *sauvegarde, maître, défaut* et *échec du maître* préalablement indiqués. Les sous-états séparés sont nécessaires pour permettre à l'objet Keeper de pouvoir quitter l'état *Variation nette* correctement lorsque des mises à jour se terminent normalement et qu'elles sont annulées. Les attributs en attente peuvent uniquement être définis lorsque l'objet Keeper se trouve à l'état *Variation nette*. Les nœuds du *maître de variation nette* et du *maître défaillant de variation nette* doivent répondre aux demandes de l'objet Keeper maître. Les nœuds des états *sauvegarde de variation nette* ou *sauvegarde défaillante de variation nette* ne doivent pas répondre aux demandes de l'objet Keeper maître |

### 7.3.9 Notes diverses

Les objets Keeper doivent gérer les paramètres des nœuds de la même liaison uniquement.

Chaque objet Keeper doit être en mesure de distribuer les paramètres de liaison à tous les autres nœuds de cette liaison. L'objet Keeper maître doit être l'objet Keeper dont le MAC ID est le plus bas parmi tous les objets Keepers légitimes de la liaison. Un objet Keeper maître défaillant sur une liaison peut ou peut ne pas être le MAC ID le plus bas.

L'objet Keeper doit être chargé de la configuration des appareils qui apparaissent dans ces attributs. L'objet Keeper maître doit être le seul objet Keeper qui réponde aux demandes de service par l'intermédiaire de la balise fixe 0x88 et diffusées à tous les objets Keeper d'un réseau.

Tous les objets Keeper légitimes d'une liaison doivent détenir des exemplaires identiques des attributs de réseau pour tous les appareils de la liaison. Les objets Keeper qui n'acceptent pas l'objet Keeper maître doivent rester à l'état d'échec de la sauvegarde tant qu'ils n'ont pas changé (accepter signifie que leurs paramètres TUI *unique_id* sont identiques).

Un seul terminal de programmation peut modifier les attributs Keeper simultanément. Le terminal de programmation doit acquérir la ressource réseau avant de modifier l'objet Keeper, et doit toujours être en possession de la ressource réseau pendant toute la procédure de modification. Si un terminal de programmation modifie un objet Keeper, il doit mettre à jour tous les autres objets Keeper de la liaison, et diffuser les demandes de service Change_Start, Set_Attribute et Change_Stop.

L'objet Keeper doit vérifier la révision de l'outil Keeper dans l'attribut 0x0104 pour un tableau d'attributs valide. Si l'objet Keeper n'est pas en mesure de comprendre cette révision,

il doit supprimer le bit 5 des transmissions TUI, de manière à ce que d'autres objets Keeper ayant cette capacité de compréhension puissent assumer les responsabilités d'objet Keeper maître. Les révisions 0 et 1 de l'objet Keeper doivent être identiques à la révision 2.

La valeur de révision d'informations de connexion est indépendante de la révision de l'objet Keeper et doit être utilisée par les outils de programmation.

Pour les besoins de diagnostic du réseau, l'objet Keeper peut diffuser un Lpacket à balise fixe de débogage (0x90) à tous les nœuds avec trois mots de données. Le premier mot identifie que l'objet Keeper envoie le diagnostic. Le deuxième mot est l'état en cours de l'objet Keeper, le troisième mot étant l'état suivant de l'objet Keeper.

### 7.3.10 Séquence d'activation de l'objet Keeper

#### 7.3.10.1 Généralités

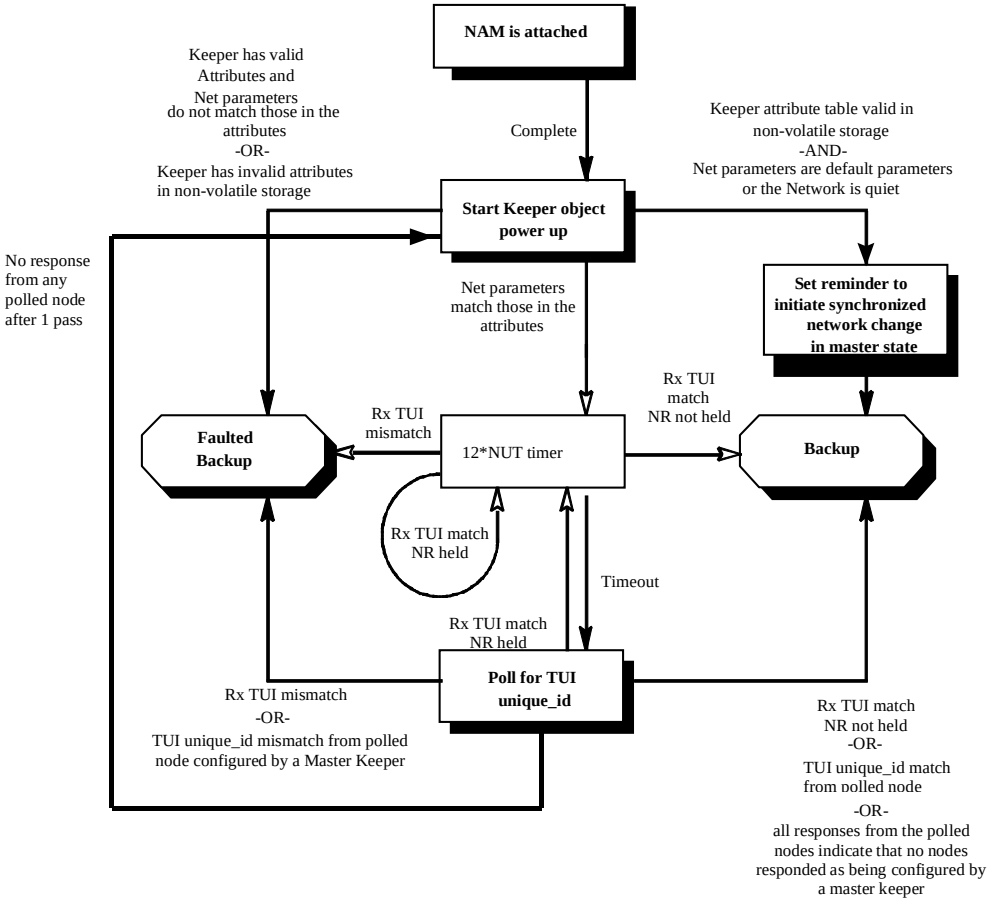
La séquence d'activation de l'objet Keeper doit lui permettre de commencer des opérations dans un large éventail de conditions, en passant d'un état d'activation de réseau configuré normal à un état dans lequel de nombreux appareils de réseau sont nouveaux et non configurés.

La séquence d'activation de l'objet Keeper ne doit pas commencer tant que le moniteur d'accès au réseau n'a pas mis le nœud Keeper en ligne. L'activation de l'objet Keeper doit déterminer si cet appareil Keeper particulier est un objet Keeper légitime pour la liaison locale et placer l'objet Keeper à l'état de fonctionnement de sauvegarde ou d'échec de la sauvegarde.

S'il s'agit d'un objet Keeper légitime pour la liaison, l'objet Keeper doit déterminer s'il s'agit d'un objet Keeper maître ou d'un objet Keeper de sauvegarde. S'il ne s'agit pas d'un objet Keeper légitime, l'objet Keeper passe à l'état d'anomalie. Un objet Keeper non configuré doit passer à l'état d'anomalie à l'activation.

Le temporisateur 12*NUT doit être de nouveau déclenché à chaque réception d'un TUI Lpacket indiquant que la ressource réseau est gérée. Cela doit indiquer que les attributs Keeper sont en cours de mise à jour.

La séquence d'activation de l'objet Keeper est illustrée dans la Figure 18.




**Légende**

| Anglais | Français |
|---|---|
| NAM is attached | NAM est associé |
| Keeper has valid attributes and net parameters do not match those on the attributes OR Keeper has invalid attributes in non-volatile storage | L'objet Keeper a des attributs valides et les paramètres Net ne correspondent pas à ceux des attributs -OU- l'objet Keeper a des attributs invalides dans la mémoire non volatile |
| Complete | Complet |
| Keeper attribute table valid in non-volatile storage AND Net parameters are default parameters or the network is quiet | Table d'attributs de l'objet Keeper valide dans la mémoire non volatile -ET- les paramètres Net sont des paramètres défaillants ou le réseau est calme |
| Start keeper object power up | Démarrage de l'activation de l'objet Keeper |
| No response from any polled node after 1 pass | Aucune réponse de tout nœud interrogé après 1 passage |
| Net parameters match those in the attributes | Les paramètres Net correspondent à ceux des attributs |
| Set reminder to initiate synchronized network change in master state | Définir le reste pour initier les modifications de réseau synchronisées à l'état maître |
| Faulted backup | Echec de sauvegarde |
| Rx TUI mismatch | Défaut d'adaptation Rx TUI |
| 12*NUT timer | Temporisateur 12 NUT |
| Rx TUI match NR not held | Correspondance Rx TUI NR non détenu |
| Backup | Sauvegarde |
| Rx TUI match NR held | Correspondance Rx TUI NR détenu |
| Timeout | Temporisation |

| Anglais | Français |
|---|---|
| -OR- | -OU- |
| TUI unique_id mismatch from polled node configured by a master keeper | Non correspondance du TUI unique_id avec le nœud interrogé configuré par un objet Keeper maître |
| Poll for TUI unique_id | Interrogation du TUI unique_id |
| TUI unique_id match from polled node | Correspondance du TUI unique_id avec le nœud interrogé |
| All responses from the polled nodes indicate that no nodes responded as being configured by a master keeper | Toutes les réponses du nœud interrogé indiquent qu'aucun nœud n'a répondu en ayant été configuré par l'objet Keeper maître |

**Figure 18 – Diagramme d'états d'activation de l'objet Keeper**

#### 7.3.10.2 Interrogation de TUI unique_id

L'état d'interrogation de TUI unique_id de la Figure 18 doit impliquer l'opération suivante, réalisée par l'objet Keeper:

a) Emettre une demande UCMM à l'objet ControlNet du nœud adressé pour lire les paramètres de configuration en cours de l'objet. Si la demande n'aboutit pas, passer au MAC ID suivant tant que UMAX n'est pas atteint. Plusieurs demandes simultanées sont admises pour réduire la durée d'interrogation.

b) Chaque MAC ID de la plage 1 à UMAX (à l'exclusion du MAC ID du nœud Keeper) peut être interrogé individuellement ou simultanément avec une demande Get_Attribute_Single sur l'attribut current_link_config de l'objet ControlNet du nœud. L'attribut contient un exemplaire de l'attribut TUI en cours pour le nœud. L'accès simultané à plusieurs nœuds permettra d'accélérer le processus d'interrogation.

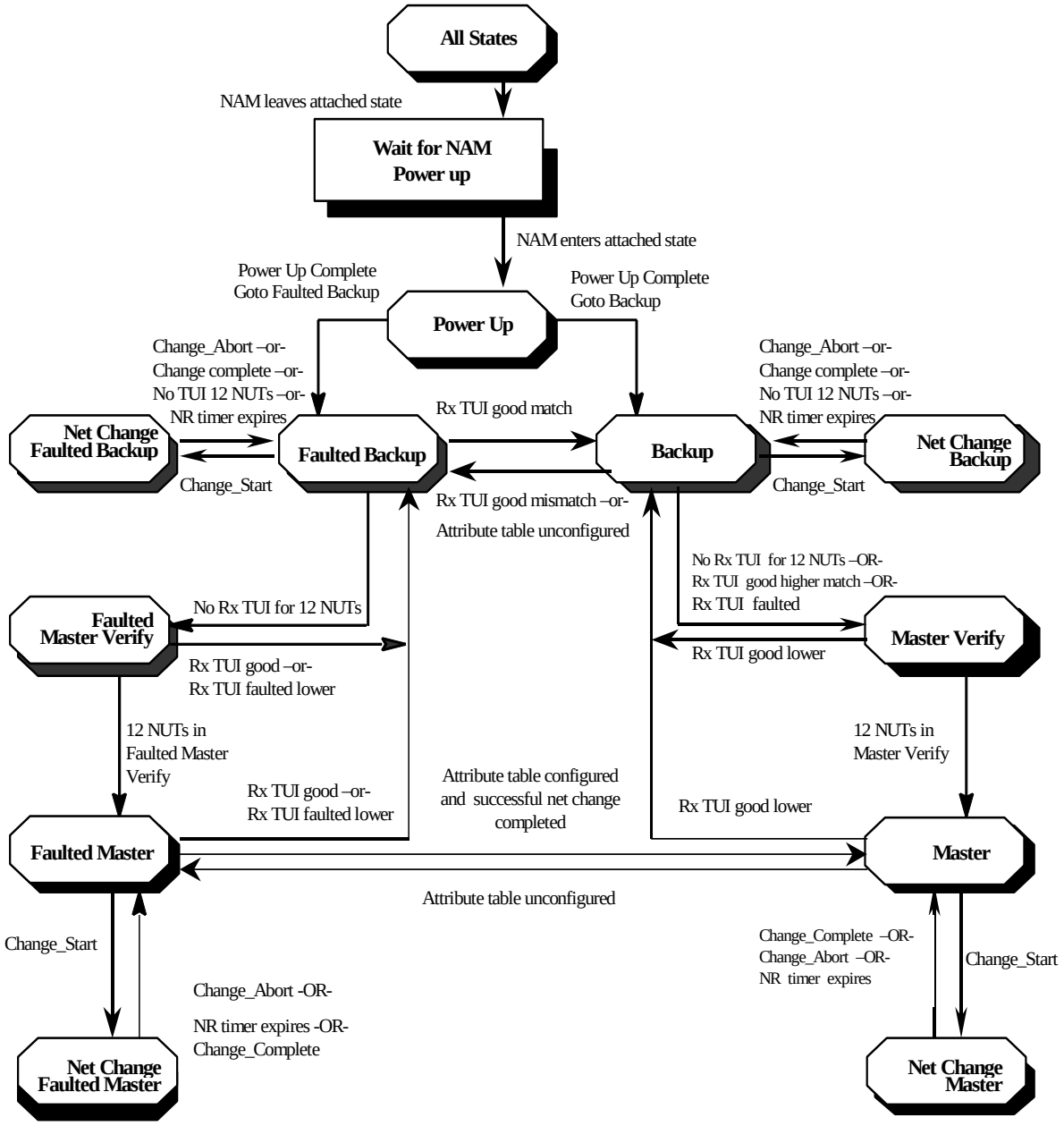
c) Si aucune réponse n'est obtenue d'un nœud interrogé et que UMAX a été atteint, l'interrogation est annulée et l'objet Keeper doit passer à l'état d'activation de l'objet Keeper, étant donné qu'au moins un autre nœud d'un réseau non défaillant est présent mais incapable de répondre. Cela permettra de répéter le processus d'interrogation d'activation tant qu'un nœud n'est pas en mesure de répondre.

d) Si une réponse a été reçue d'un nœud interrogé et qu'une valeur a été attribuée au bit d'état de l'objet Keeper dans le TUI status_flag renvoyé, le nœud interrogé a été configuré par un objet Keeper à l'état de fonctionnement Maître. Comparer la valeur du TUI unique_id renvoyée par le nœud interrogé à celle de la table d'attributs de cet objet Keeper. Si elles correspondent, l'interrogation est annulée et l'objet Keeper doit passer à l'état de fonctionnement de sauvegarde. Si elles ne correspondent pas, l'interrogation est annulée et l'objet Keeper doit passer à l'état de fonctionnement d'échec de la sauvegarde.

e) Si une réponse a été reçue d'un nœud interrogé et qu'une valeur n'a pas été attribuée au bit d'état de l'objet Keeper dans le TUI status_flag renvoyé, le nœud interrogé n'a pas été configuré par un objet Keeper à l'état de fonctionnement Maître. Dans ce cas, les informations sont écartées et l'interrogation se poursuit. Si tous les nœuds répondent de cette manière, indiquant qu'aucun n'a été configuré par un objet Keeper à l'état de fonctionnement maître, l'interrogation se termine et l'objet Keeper doit passer à l'état de fonctionnement de sauvegarde, puisque que dans ce cas, tous les nœuds sont nouveaux ou ont perdu leur configuration stockée.

#### 7.3.10.3 Etats de fonctionnement

Les états de fonctionnement de l'objet Keeper doivent être définis conformément à la Figure 19, au Tableau 35 et aux Paragraphes 7.3.10.3 à 7.3.10.9.




**Légende**

| Anglais | Français |
|---|---|
| All states | Tous les états |
| NAM leaves attached state | Le NAM quitte l'état associé |
| Wait for NAM power up | Attente de l'activation du NAM |
| NAM enters attached state | Le NAM passe à l'état associé |
| Power up complete | Activation réalisée |
| Goto faulted backup | aller à l'état échec de sauvegarde |
| Goto backup | Aller à l'état de sauvegarde |
| -or- | -ou- |
| No TUI 12 NUTs | Pas de TUI de 12 NUT |
| NR timer expires | Temporisateur NR écoulé |
| Rx TUI good match | Bonne correspondance Rx TUI |

| Anglais | Français |
|---|---|
| Net change | Variation nette |
| faulted backup | échec de sauvegarde |
| Backup | Sauvegarde |
| Rx TUI good mismatch | Non correspondance Rx TUI correcte |
| Attribute table unconfigured | Table d'attributs non configurée |
| No Rx TUI for 12 NUTs | Pas de Rx TUI pendant 12 NUT |
| Rx TUI good higher match | Bonne correspondance supérieure Rx TUI |
| Rx TUI faulted | Echec Rx TUI |
| Faulted | Echec |
| Master verify | Vérification maître |
| Rx TUI good | Rx TUI correcte |
| Rx TUI faulted lower | Echec Rx TUI inférieure |
| Rx TUI good lower | Rx TUI inférieure correcte |
| 12 NUTs in faulted master verify | 12 NUT au cours de l'échec de vérification maître |
| 12 NUTs in master verify | 12 NUT au cours de la vérification maître |
| Rx TUI good | Rx TUI correcte |
| Faulted master | Echec maître |
| Master | Maître |
| Attribute table configured and successful net change completed | Table d'attributs configurée et réussite de la réalisation de la variation nette |
| Attribute table unconfigured | Table d'attributs non configurée |
| Net change faulted master | Variation nette – échec maître |
| Net change master | Variation nette – maître |

**Figure 19 – Diagramme d'états de fonctionnement de l'objet Keeper**

**Tableau 35 – Matrice d'événement d'état de l'objet Keeper**

| Evénement | Etat | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acti-vation | Sauve-garde | Vérifi-cation du maître | Maître | Echec de la sauve-garde | Echec de la vérification du maître | Echec du maître | Variation nette (sauve-garde) | Variation nette (Maître) | Variation nette (Echec de la sauve-garde) | Variation nette (Echec du maître) |
| Rx TUI provenant d'un nœud correct | | | | | | Aller à Echec de la sauve-garde | Aller à Echec de la sauve-garde | | | | |
| Rx TUI provenant d'un nœud inférieur correct | | | Aller à Sauvegarde | | | | | | | | |
| Rx TUI provenant d'un nœud supérieur correct (correspondance) | | Aller à Vérifi-cation du maître | | | | | | | | | |
| Rx TUI provenant d'un nœud correct (non correspondance) | | Aller à Echec de la sauve-garde | | | | | | | | | |
| Rx TUI provenant d'un nœud correct (correspondance) | | | | | Aller à Sauve-garde | | | | | | |

| Evénement | Etat | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Rx TUI provenant d'un nœud défaillant | | Aller à Vérification du maître | | | | | | | | | |
| Rx TUI provenant d'un nœud inférieur défaillant | | | | | | Aller à Echec de la sauvegarde | | | | | |
| Pas de TUI Rx pendant 12*NUT | | Aller à Vérification du maître | | | Aller à Echec de la vérification du maître | | | Annuler modification. Aller à Sauvegarde | | Annuler modification. Aller à Echec de la sauvegarde | |
| 12 Nut écoulées | | | Aller à Maître | | | Aller à Echec du maître | | | | | |
| Change_Start | | Aller à Variation nette (Sauvegarde) | | Aller à Variation nette (Maître) | Aller à Variation nette (Echec de la sauvegarde) | | Aller à Variation nette (Echec du maître) | | | | |
| Change_Complete | | | | | | | | Aller à Sauvegarde | Aller à Maître | Aller à Sauvegarde | Aller à Vérification du maître |
| Change_Abort | | | | | | | | Annuler modification. Aller à Sauvegarde | Annuler modification. Aller à Maître | Annuler modification. Aller à Echec de la sauvegarde | Annuler modification. Aller à Echec du maître |
| Le temporisateur de ressource réseau 60 s a expiré | | | | | | | | Annuler modification. Aller à Sauvegarde | Annuler modification. Aller à Maître | Annuler modification. Aller à Echec de la sauvegarde | Annuler modification. Aller à Echec du maître |
| Le moniteur d'accès au réseau a quitté l'état associé | Attendre l'activation du moniteur d'accès au réseau | | | | | | | | | | |
| Activation terminée pour Sauvegarde | Aller à Sauvegarde | | | | | | | | | | |
| Activation terminée pour Echec de la sauvegarde | Aller à Echec de la sauvegarde | | | | | | | | | | |

NOTE Les cellules vierges indiquent que cet événement ne peut pas se produire ou que, s'il se produit, aucune action ne doit être entreprise par l'objet Keeper.

### 7.3.10.4 Ressource réseau

L'accès aux objets Keeper de modification doit être contrôlé à l'aide de la ressource réseau. Un seul appareil peut acquérir la ressource réseau pour un accès exclusif à la fois.

La ressource réseau doit être gérée par l'objet Keeper maître pour le nœud qui procède aux mises à jour de l'attribut. Si l'objet Keeper maître n'écoute pas le nœud qui procède à la mise à jour au moins une fois toutes les 60 s, il doit automatiquement libérer la ressource réseau, annuler la modification en cours et reprendre un fonctionnement normal.

#### 7.3.10.5 Keeper défaillant

Un objet Keeper dont l'attribut *TUI unique_id* ne correspond pas à celui de la diffusion TUI provenant d'un objet Keeper doit passer à l'état d'anomalie, sauf si une opération de variation nette est en cours.

#### 7.3.10.6 Le MAC ID choisit l'objet Keeper maître

L'objet Keeper doit utiliser le MAC ID du nœud Keeper inclus dans le message TUI pour déterminer l'objet Keeper d'une liaison qui est le maître.

#### 7.3.10.7 Vérification du maître

L'objet Keeper doit attendre 12*NUT après le début de la première diffusion TUI avant de supposer l'état de l'objet Keeper maître. Cette durée doit être suffisante pour permettre à un objet Keeper doté d'un MAC ID inférieur de devenir le maître. Les autres objets Keeper peuvent devenir des objets Keeper de sauvegarde.

#### 7.3.10.8 Suspect

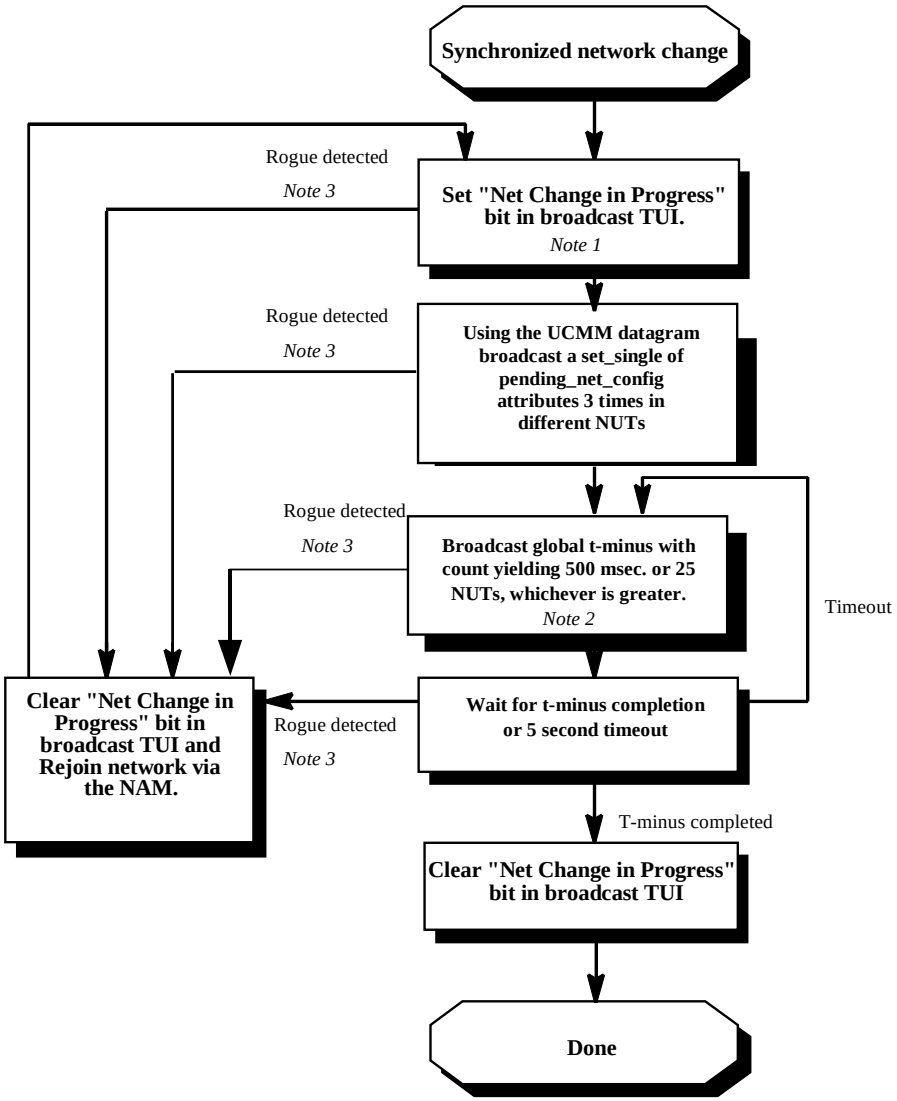
Si le nœud contenant l'objet Keeper devient suspect pendant une séquence de modification de réseau synchronisée, le bit de variation nette en cours doit être supprimé, et le contrôle doit être renvoyé au début du traitement de la modification de réseau synchronisée immédiatement après que le moniteur d'accès au réseau a remis le nœud en ligne. La séquence normale d'activation de l'objet Keeper doit être ignorée.

#### 7.3.10.9 Algorithme de traitement de la modification de réseau synchronisée

Bien que la liaison utilise le modérateur pour distribuer les paramètres de fonctionnement, c'est l'objet Keeper qui est tenu de modifier les paramètres de liaison.

Toutes les modifications de paramètre de liaison doivent être réalisées à l'aide d'un algorithme de modification de réseau synchronisé. Cela doit permettre à tous les nœuds du réseau de modifier simultanément leurs paramètres de liaison en cours sans interrompre le fonctionnement du réseau.

L'algorithme de modification de réseau synchronisée doit être tel que défini dans la Figure 20.




NOTE 1 La diffusion d'un TUI avec le bit de modification de réseau en cours oblige les nœuds qui viennent d'être activés à suspendre la mise en ligne tant que la modification de réseau n'est pas terminée, et que le bit de modification de réseau en cours de la diffusion TUI n'est pas supprimé. Cela permet de limiter le risque d'événement suspect.

NOTE 2 Le délai tMinus assure l'immunité au bruit et permet à tous les nœuds d'entendre au moins un modérateur avec le nouveau comptage tMinus. Il fournit également aux nœuds une durée de traitement raisonnable des paquets Set_Single avec de nouveaux attributs de réseau. Le délai est déterminé en prenant la valeur maximale 25 ou en divisant 500 ms par la durée NUT en cours (en ms).

NOTE 3 La condition suspecte se produira uniquement si le nœud modérateur et au moins un autre nœud n'ont pas reçu l'un des trois paquets de demande de service de l'attribut de réseau de définition de diffusion. Cela se produit uniquement en cas d'événement bruyant sérieux (durant au moins 3 NUT).

**Légende**

| Anglais | Français |
|---|---|
| Synchronized network change | Modification de réseau synchronisée |
| Rogue detected | Evénement suspect détecté |
| Set « Net change in progress » bit in broadcast TUI | Définir le bit modification de réseau en cours dans le TUI de diffusion |
| Using the UCMM datagram broadcast as set_single of pending_net_config attributes 3 times in different NUTs | A l'aide du datagramme UCMM, diffuser un set_single des attributs pending_net_config 3 fois pendant différentes NUT |
| Broadcast global t-minus with count yielding 500 msec. Or 25 NUTS, whichever is greater | Diffuser t-minus global à un rythme de 500 msec. ou 25 NUT, selon la plus grande valeur |
| Timeout | Temporisation |
| Clear "net change in progress" bit in broadcast | Effacer le bit " modification de réseau en cours" |

| Anglais | Français |
|---|---|
| TUI | dans le TUI de diffusion |
| And rejoin network via the NAM | Et joindre le réseau via le NAM |
| Wait for t-minus completion or 5 second timeout | Attendre la réalisation complète de t-minus ou une temporisation de 5 s |
| t-minus completed | t-minus écoulé |
| Done | Fin réalisation |

**Figure 20 – Traitement de la modification du réseau synchronisée**

## 7.4 Objet de planification

### 7.4.1 Présentation

L'objet de planification doit exister dans les appareils à l'origine de la connexion (CO). L'objet de planification doit être utilisé par un logiciel de planification de liaison (LSS) pour

a) lire les informations de connexion planifiée;

b) écrire les informations de planification;

c) offrir un mot de passe (clé logicielle) à l'auteur de la connexion, qui doit être utilisée pour authentifier l'accès de l'appareil CO à une liaison particulière.

Une session LSS avec l'objet de planification doit permettre de planifier une liaison particulière. Si un appareil CO détient des connexions sur plusieurs liaisons, plusieurs sessions LSS doivent être nécessaires pour planifier la totalité des connexions. Les résultats de la session de planification doivent être sauvegardés par l'appareil CO (y compris le mot de passe CO calculé par le LSS).

Etant donné que le LSS peut être intégré au logiciel de programmation d'un appareil CO, l'objet de planification doit fournir les commandes de prise en charge d'un logiciel de programmation intégré ou séparé. De manière spécifique, lorsque le LSS écrit des données de planification dans l'objet de planification, il peut le faire sous condition ou forcer l'écriture. Les données de planification doivent être écrites sous condition si le logiciel de programmation et le LSS n'ont pas été intégrés, et l'objet de planification doit uniquement utiliser les données de planification fournies si un autre logiciel de programmation n'a pas modifié les informations de connexion pendant la session LSS. Une écriture forcée peut être utilisée avec un LSS/logiciel de programmation intégré si l'outil intégré peut garantir qu'aucun autre logiciel de programmation n'a modifié les informations de connexion depuis la dernière lecture pendant la session de planification en cours.

Toutes les communications adressées à l'objet de planification doivent utiliser le service fixe UCMM (Unconnected Message Manager) ou un service de connexion à balise générique actif vers l'objet Routeur de message.

### 7.4.2 Attributs de classe

L'objet Scheduling doit prendre en charge les attributs de classe tel que spécifié dans le Tableau 36. Le service Get_Attribute_All doit renvoyer les attributs de niveau de classe dans l'ordre numérique croissant.

**Tableau 36 – Attributs de classe de l'objet de planification**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut |
|---|---|---|---|---|---|
| 0x01 | Obligatoire | Get | Revision | UINT | Première révision, valeur = 1 |
| 0x02 | Obligatoire | Get | MaxInstances | UDINT | Nombre maximal d'objets de planification pris en charge |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut |
|---|---|---|---|---|---|
| 0x03 | Obligatoire | Get | NumInstances | UDINT | Nombre d'objets de planification instanciés |

### 7.4.3 Attributs d'instance

L'objet Scheduling doit prendre en charge les attributs d'instance tel que spécifié dans le Tableau 37. Le service Get_Attribute_All doit renvoyer les attributs de niveau d'attribut dans l'ordre numérique croissant.

**Tableau 37 – Attributs d'instance de l'objet de planification**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut |
|---|---|---|---|---|---|
| 0x01 | Obligatoire | Get/Set | Route | STRUCT de taille variable | Route entre le CO et la liaison en cours de planification |
| | | | NumSegments | UINT | Nombre de segments du chemin |
| | | | Segments | MATRICE d'UINT | Matrice de segments |
| 0x02 | Obligatoire | Get/Set | TimeOut | UDINT | Délai d'attente du chien de garde (µs)<br>(valeur par défaut = 60 s) |
| 0x03 | Facultatif | Get/Set | Etat du contrôleur | UINT | bit 0 (exécution/programme)<br>= 0, peut être programmé<br>= 1, contrôle le bit d'E/S<br>bit 1 (distant/local)<br>= 0, le bit 0 ne doit pas être modifié à distance<br>= 1, le bit 0 peut être modifié à distance<br>bits 2 à 15 réservés et doivent être nuls |

### 7.4.4 Services communs

#### 7.4.4.1 Généralités

Outre les services décrits dans en 7.4.4, l'objet de planification doit prendre en charge le Get_Attribute_All standard adressé à l'instance de classe (instance zéro). La mise en œuvre d'un objet de planification particulier peut éventuellement prendre en charge d'autres attributs de service standard.

L'objet de planification doit prendre en charge les services communs spécifiés dans le Tableau 38.

**Tableau 38 – Services communs de l'objet de planification**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x01 | Obligatoire | Facultatif | Get_Attribute_All | Renvoie une liste prédéfinie des attributs de cet objet. Modifie tous les attributs définissables |
| 0x02 | N/A | Facultatif | Set_Attribute_All | Modifie tous les attributs définissables |

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x03 | Facul-tatif | Facul-tatif | Get_Attribute_List | Renvoie une liste d'attributs spécifiés |
| 0x04 | N/A | Facul-tatif | Set_Attribute_List | Modifie une liste d'attributs définissables |
| 0x08 | Obliga-toire | Facul-tatif | Create | Permet d'instancier un objet de planification |
| 0x09 | N/A | Obliga-toire | Delete | Permet de mettre fin à une session de planification LSS |
| 0x0E | Facul-tatif | Facul-tatif | Get_Attribute_Single | Renvoie le contenu de l'attribut spécifié |
| 0x10 | N/A | Facul-tatif | Set_Attribute_Single | Modifie un seul attribut |

Une mise en œuvre peut prendre en charge la définition de l'attribut au niveau de l'instance de la route, mais si ce n'est pas le cas, il ne doit pas être possible de distinguer le comportement interne d'un service delete suivi d'un service create contenant la nouvelle route.

#### 7.4.4.2 Create

Le service Create doit être utilisé pour instancier un objet de planification sur le CO. Dans le message Create, le client (LSS) doit inclure une route pointant du CO vers la liaison à planifier. La route vers la liaison doit être composée de segments de port et ne doit pas contenir de segments de réseau. L'objet de planification doit répondre par un numéro d'instance auquel toutes les communications ultérieures doivent être adressées pendant cette session de planification, et par les mots de passe et chemin CO déjà enregistrés.

La nouvelle instance doit se supprimer elle-même si les communications avec son client sont coupées. Les communications avec une instance doivent être surveillées à l'aide d'un temporisateur dont la valeur de délai d'attente (par défaut = 60 s) peut être attribuée par l'un des services d'attribut de définition facultatifs. Le temporisateur d'une instance doit être réinitialisé lorsqu'il reçoit un message. Si le temporisateur expire, l'instance doit être supprimée en annulant toutes les modifications en attente. La valeur de délai d'attente doit être un entier de 32 bits exprimé en µs.

Si le service Create s'adresse au niveau de classe (instance zéro), le numéro d'instance de la nouvelle instance créée doit être choisi automatiquement par l'objet de planification. Pour les besoins du débogage, l'objet de planification peut éventuellement permettre au service Create d'être adressé à un numéro d'instance particulier.

L'objet de planification doit prendre en charge au moins une instance. Si l'objet de planification admet l'existence de plusieurs instances, il doit s'assurer de l'intégrité de chacune d'elles. La valeur de l'attribut d'instance 0x01 doit être fournie avec le service Create.

```
class Scheduling_Create_Request: public MR_Request
{
   UINT   number_of_attributes;     // set to 1
   InstanceAttribute1 attribute1;
}

class Scheduling_Create_Response: public MR_Response
{
   UDINT  instance_number;
   UDINT  cop;             // Connection Originator Password (COP)
   UINT   path_size;          // in words, range 0 to 255
   UINT   path[];             // array of port segments from the scheduled link
                           // to the connection originator
                           // no key segments, no network segments
                           // include the MAC ID of the link hop
};
```

La réponse doit contenir l'un des codes d'état figurant dans le Tableau 39.

**Tableau 39 – Description de l'erreur d'état de Create**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x02 | 0x0001 | Ressources insuffisantes — le nombre maximal d'instances d'objet de planification existe déjà |
| | 0x0002 | Ressources insuffisantes — mémoire insuffisante sur l'appareil CO |
| 0x04 | N/A | Erreur de syntaxe du chemin |
| 0x08 | N/A | Service non mis en œuvre (les instances non nulles étant facultatives) |
| 0x10 | N/A | Impossible d'ouvrir une autre instance suite à des conflits internes |
| 0x13 | N/A | Données de demande insuffisantes — la demande est trop courte ou tronquée |
| 0x0D | N/A | L'objet existe déjà (lorsqu'une instance non nulle est spécifiée) |
| 0x1C | N/A | Pénurie de listes d'attributs — il manque l'attribut requis |
| 0xDO | N/A | Aucune connexion planifiée sur cette liaison |

#### 7.4.4.3 Delete

Le service Delete doit être utilisé pour mettre fin à une session de planification du LSS. Il permet de désallouer toutes les ressources d'une instance spécifiée de l'objet de planification. Toutes les modifications en attente qui n'ont pas encore été engagées doivent être perdues. Les connexions rompues par des commandes antérieures dans la session LSS ne doivent pas nécessairement être restaurées. Le service Delete ne doit pas être pris en charge au niveau de la classe. La réponse de ce service doit être spécifiée conformément au Tableau 40.

**Tableau 40 – Description d'erreur d'état pour Delete et Kick_Timer**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Erreur de syntaxe du chemin |
| 0x05 | N/A | Instance non définie |
| 0x08 | N/A | Service non mis en œuvre (si dirigé vers l'instance 0) |

### 7.4.5 Services spécifiques à la classe

#### 7.4.5.1 Généralités

L'objet Scheduling doit prendre en charge les attributs de classe suivant tel que spécifié dans le Tableau 41.

**Tableau 41 – Services spécifiques à la classe de l'objet de planification**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x4B | N/A | Obligatoire | Kick_Timer | Réinitialisation du temporisateur dans l'objet de planification |
| 0x4C | N/A | Obligatoire | Read | Lit les informations relatives aux connexions planifiées |
| 0x4D | N/A | Obligatoire | Conditional_Write | Réécrit les informations de planification sur l'appareil CO |
| 0x4E | N/A | Facultatif | Forced Write | Réécrit les informations de planification sur l'appareil CO |
| 0x4F | N/A | Obligatoire | Change Start | Lance une modification des données de planification |
| 0x50 | N/A | Obligatoire | Break Connections | Interrompt toutes les connexions affectées après de nouvelles données de planification |
| 0x51 | N/A | Obligatoire | Change Complete | Indique que toutes les connexions planifiées ont été interrompues |
| 0x52 | Obligatoire | N/A | Restart Connections | Interrompt toutes les connexions qui utilisent une liaison particulière |

#### 7.4.5.2 Kick_Timer

Le service kick_timer doit réinitialiser le temporisateur dans une instance de l'objet de planification sans affecter l'état de cet objet. Le LSS doit émettre quatre commandes par période de délai d'attente. Par exemple, le LSS doit émettre une demande de service toutes les 15 s si la valeur de délai d'attente est de 60 s par défaut. Tous les autres services de l'objet de planification, dirigés vers cette instance de planification, doivent également réinitialiser le temporisateur de manière à utiliser le service Kick_Timer uniquement si le client (LSS) souhaite conserver une instance de l'objet de planification active alors qu'il n'a rien à dire.

Le temporisateur d'une instance doit être réinitialisé lorsqu'il reçoit un message. Toutefois, son compte à rebours ne doit pas démarrer tant que la réponse correspondante n'a pas été envoyée. Ceci ne doit permettre la mise en œuvre que d'une seule tâche. La réponse de ce service doit être spécifiée conformément au Tableau 40.

#### 7.4.5.3 Read

Le LSS doit utiliser le service Read pour lire les informations relatives aux connexions planifiées que l'appareil CO souhaite créer sur une liaison particulière. Les informations de connexion doivent provenir d'un objet d'application dans l'appareil CO.

Une demande de service Read doit contenir un paramètre UDINT requis qui spécifie l'indice de connexion sur lequel démarrer la lecture. L'objet de planification doit répondre à un service Read avec autant d'informations de connexion que ne peut en contenir un Lpacket de réponse. Les indices de connexion d'un Lpacket doivent être classés dans l'ordre croissant (les intervalles doivent être admis). Le champ d'état étendu de la réponse doit indiquer s'il

existe des connexions dotées d'indices plus élevés dans l'appareil CO. Un client (LSS) doit continuer à soumettre des demandes read avec des indices de démarrage plus élevés tant qu'il n'a pas lu toutes les informations de connexion de la liaison choisie. Un état étendu 0 doit indiquer qu'il existe plus de connexions dans l'appareil. Un état étendu 1 doit indiquer que toutes les connexions ont été lues.

La réponse doit être composée d'un UINT indiquant le nombre de connexions décrites dans cette réponse, suivi d'une série d'entrées décrivant chacune une connexion répondant aux critères de demande de connexion (la route vers la liaison). Chaque entrée doit être associée à un indice de données de planification unique, choisi et devant être utilisé principalement par l'appareil CO. Le dernier de ces indices peut être incrémenté par le LSS pour poursuivre la demande si elle est trop longue pour une réponse.

Les autres paramètres fournis dans l'entrée de connexion doivent décrire les tailles de connexion, les types de connexion (multidiffusion/point à point) et les vitesses de mise à jour des connexions (RPI). Les informations relatives à la route d'entrée de connexion doivent être composées de la route entre la liaison et le module cible pour la connexion comprenant les points de connexion du module cible. Les informations relatives au port peuvent ne pas être réellement utilisées par le LSS.

Cette route doit également contenir les valeurs en cours attribuées à la connexion dans les segments de réseau. Les segments de réseau doivent contenir les informations relatives à l'API (intervalle Lpacket) et à la NUT de départ.

```
class Scheduling_Read_Request: public MR_Request
{
   UDINT  first_connection_index;
}

class Scheduling_Read_Response: public MR_Response
{
   UINT   number_of_connections;
   Scheduling_Connection_Description desc[];
}

class Scheduling_Connection_Description
{
   UDINT  connection_index;
   UINT   O2T_parameters;
   UINT   T2O_parameters;
   UDINT  O2T_RPI;
   UDINT  T2O_RPI;
   UINT   path_size;      // in words, range 0 to 255
   UINT   path[];         // array of path segments
                     // first segment is always O=>T schedule segment
                     // second is always port segment onto the link
                     // third is always T=>O schedule segment
                     // remaining path including logical segments
                     // and other port segments if multihop connection
};
```

L'outil de planification LSS doit utiliser les segments logiques du chemin cible afin de déterminer si plusieurs demandes de connexion utilisent réellement un producteur multipoint. La règle concernant tous les objets (outre l'objet Assembly) doit stipuler qu'une correspondance de classe, d'instance, de point de connexion T⇒O et de granularité plus fine (attribut ou membre, par exemple) constitue un producteur unique et doit être planifiée une seule fois pour les deux connexions. Etant donné que l'objet Assembly compare une instance et un point de connexion, une correspondance d'une instance de chemin de connexion ne doit pas déterminer le caractère unique du producteur.

Par exemple, ces deux chemins de connexion spécifient le même producteur (instance 0x88 de l'objet Assembly):

"20 04 24 03 2C 99 2C 88";

"20 04 24 AA 2C 77 2C 88".

Ces deux chemins de connexion spécifient des producteurs différents, étant donné que le segment de classe logique ne correspond pas:

"20 77 24 03 2C 99 2C 88";

"20 77 24 AA 2C 99 2C 88".

La réponse d'état de ce service doit être spécifiée conformément au Tableau 42.

**Tableau 42 – Description de l'erreur d'état de Read**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Erreur de syntaxe du chemin |
| 0x05 | N/A | Instance non définie |
| 0x08 | N/A | Service non mis en œuvre (si dirigé vers l'instance 0) |
| 0x13 | N/A | Données de demande insuffisantes — la demande est trop courte ou tronquée |

#### 7.4.5.4 Conditional_Write

Le service Conditional_Write doit être utilisé par le LSS pour réécrire les informations de planification dans l'appareil CO. La réception de ce message doit indiquer que le LSS planifie de nouveau des indices particuliers des informations de connexion. Un appareil CO peut commencer par interrompre les connexions en cours associées aux indices de connexion ou attendre la réception d'une demande de service d'interruption des connexions consécutive.

Si la nouvelle planification fournie par le LSS correspond exactement pour une connexion donnée, l'appareil CO peut ignorer cette partie de la demande d'écriture. Une mise en œuvre de l'objet de planification peut répondre par une erreur de ressource indisponible si elle reçoit plus de mises à jour que demandées pour plusieurs connexions (par l'intermédiaire de demandes Read). Une mise en œuvre peut également vérifier les demandes d'écriture en double et donc autoriser les nouvelles tentatives (plus de demandes d'écriture que de lecture). Les indices de connexion peuvent ne pas être dans l'ordre.

La différence entre le service d'écriture conditionnelle et le service d'écriture forcée doit être que, dans le premier cas, l'appareil CO doit être requis pour vérifier la fraîcheur des informations de connexion préalablement envoyées au LSS. Si les données ne sont pas fraîches, l'appareil CO ne doit pas utiliser les données de planification pour cet indice de connexion. Dans le cas du service d'écriture forcée, l'appareil CO peut ne pas procéder à une vérification. Informations de connexion « fraîches » s'entend qu'elles n'ont pas été modifiées depuis leur envoi au LSS à l'aide de la commande Read.

La demande doit être une matrice de planifications de connexion.

```
class Scheduling_Write_Request: public MR_Request
{
   UINT   number_of_connection_schedules;
   struct {
      UDINT  connection_index;
      USINT  O2T_schedule;
      USINT  T2O_schedule;
   } schedules[];
};
```

L'état de réponse de ce service doit être spécifié conformément au Tableau 43.

**Tableau 43 – Descriptions de l'erreur d'état de Conditional_Write**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | N/A | Success |
| 0x03 | N/A | Valeur non valide — La requête contient un indice incorrect |
| 0x04 | N/A | Erreur de syntaxe du chemin |
| 0x05 | N/A | Instance non définie |
| 0x08 | N/A | Service non mis en œuvre (si dirigé vers l'instance 0) |
| 0x0C | N/A | Etat incorrect (***change start*** non reçu) |
| 0x10 | N/A | Conflit d'état de l'appareil (l'appareil ne peut pas interrompre la connexion) |
| 0x13 | N/A | Données de demande insuffisantes — la demande est trop courte ou tronquée |
| 0x15 | N/A | L'instance n'est pas en mesure de recevoir d'autres demandes d'écriture |

#### 7.4.5.5 Forced_Write

Le service Forced_Write doit être analogue au service Conditional_Write, sauf que le LSS doit garantir avoir fourni des informations de connexion valides. Les paramètres de demande doivent être identiques à ceux du service Conditional_Write. L'état de réponse de ce service doit être spécifié conformément au Tableau 44.

**Tableau 44 – Description de l'erreur d'état pour Forced_Write**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Erreur de syntaxe du chemin |
| 0x05 | N/A | Instance non définie |
| 0x08 | N/A | Service non mis en œuvre (si dirigé vers l'instance 0) |
| 0x0C | N/A | Etat incorrect (***change start*** non reçu) |
| 0x10 | N/A | Conflit d'état de l'appareil (l'appareil ne peut pas interrompre la connexion) |
| 0x13 | N/A | Données de demande insuffisantes — la demande est trop courte ou tronquée |
| 0x15 | N/A | L'instance n'est pas en mesure de recevoir d'autres demandes d'écriture |

#### 7.4.5.6 Change_Start

Le service Change_Start doit être utilisé pour lancer une modification de données de planification dans le contrôleur. Le contrôleur doit allouer suffisamment de mémoire pour les nouvelles données, le mot de passe et le chemin CO. Le service ***conditional write*** ou ***forced write*** doit être utilisé pour écrire les données dans le contrôleur.

Le mot de passe CO doit être utilisé lorsqu'un contrôleur est activé, puis commence à rétablir les connexions planifiées. Le CO doit s'assurer que le chemin associé au mot de passe CO (voir réponse Create ou demande Change_Complete) correspond à celui du chemin entre le réseau et l'appareil CO. S'il correspond, le CO doit s'assurer que ce mot de passe CO correspond à celui stocké dans l'objet Keeper de la liaison, avant d'établir des connexions planifiées sur ladite liaison.

La demande Change_Start ne doit comporter qu'un seul paramètre. Le paramètre doit être un UINT compris entre 0 et 255 qui spécifie la taille du mot de passe et du chemin CO à fournir dans le service Change_Complete. Les mises en œuvre peuvent l'utiliser pour allouer une mémoire tampon de réception pour le mot de passe et le chemin CO.

L'état de réponse de ce service doit être spécifié conformément au Tableau 45.

**Tableau 45 – Description de l'erreur d'état de Change_Start**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | N/A | Success |
| 0x02 | 0x0002 | Ressources insuffisantes — mémoire insuffisante sur l'appareil CO |
| 0x04 | N/A | Erreur de syntaxe du chemin |
| 0x05 | N/A | Instance non définie |
| 0x08 | N/A | Service non mis en œuvre (si dirigé vers l'instance 0) |
| 0x10 | N/A | Conflit d'état de l'appareil (l'appareil ne peut pas allouer suffisamment de mémoire, par exemple) |
| 0x13 | N/A | Données de demande insuffisantes — la demande est trop courte ou tronquée |

#### 7.4.5.7 Break_Connections

Le service Break_Connections doit être utilisé après avoir écrit les nouvelles données de planification dans le contrôleur afin d'interrompre toutes les connexions concernées. Le service doit répondre uniquement après l'interruption de toutes les connexions. Si des connexions sont en cours, elles doivent également être interrompues avant de répondre à ce service.

Le service Break_Connections de l'objet de planification doit interrompre toutes les connexions spécifiées dans les services d'écriture de données de planification. Un LSS peut toujours écrire des données de planification pour toutes les connexions (interrompant toutes les connexions sur la liaison), mais un autre LSS peut être optimisé pour limiter le nombre de connexions interrompues.

Lors de la réception du service Break_Connections de l'objet de planification, outre l'interruption des connexions modifiées, le CO doit attribuer une valeur nulle (zéro) aux données planifiées actives pour les connexions qu'il interrompt, afin éviter de les rétablir tant que le service Change_Complete des données de planification n'a pas été reçu. Si le service Change_Complete n'est jamais reçu (les communications avec le LSS ayant été interrompues), ces connexions ne doivent pas pouvoir être ouvertes tant qu'elles n'ont pas été de nouveau planifiées. La réponse de ce service doit être spécifiée conformément au Tableau 46.

**Tableau 46 – Descriptions de l'erreur d'état de Break_Connections**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Erreur de syntaxe du chemin |
| 0x05 | N/A | Instance non définie |
| 0x08 | N/A | Service non mis en œuvre (si dirigé vers l'instance 0) |
| 0x0C | N/A | Conflit d'état d'objet |
| 0x10 | N/A | Conflit d'état de l'appareil (l'appareil ne peut pas interrompre les connexions) |
| 0x13 | N/A | Données de demande insuffisantes — la demande est trop courte ou tronquée |

### 7.4.5.8 Change_Complete

Le service Change_Complete doit être utilisé pour indiquer que toutes les connexions planifiées ont été interrompues et qu'il est admis d'établir les nouvelles connexions planifiées. La demande contient le nouveau mot de passe et le nouveau chemin CO utilisés pour la liaison qui a été planifiée. Le CO doit s'assurer que le chemin associé au mot de passe CO (voir réponse Create ou demande Change_Complete) correspond à celui du chemin entre le réseau et l'appareil CO. S'il correspond, le CO doit s'assurer que ce mot de passe CO correspond à celui stocké dans l'objet Keeper de la liaison, avant d'établir des connexions planifiées sur ladite liaison.

```
class Scheduling_Change_Complete_Request: public MR_Request
{
   UDINT  cop;             // Connection Originator Password (COP)
   UINT   path_size;          // in words, range 0 to 255
   UINT   path[];             // array of port segments from the scheduled link
                           // to the connection originator
                           // no key segments, no network segments
                           // include the MAC ID of the link hop
};
```

NOTE Grâce au chemin inverse de cette demande, le LSS peut détecter les modifications dans la position de réseau du CO planifié pendant les dernières sessions de planification.

L'état de réponse de ce service doit être spécifié conformément au Tableau 47.

**Tableau 47 – Descriptions de l'erreur d'état de Change_Complete**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | N/A | Success |
| 0x03 | N/A | Valeur non valide (les nouveaux mots de passe et chemin CO ne sont pas correctement mis en forme) |
| 0x04 | N/A | Erreur de syntaxe du chemin |
| 0x05 | N/A | Instance non définie |
| 0x08 | N/A | Service non mis en œuvre (si dirigé vers l'instance 0) |
| 0x10 | N/A | Conflit d'état de l'appareil |
| 0x13 | N/A | Données de demande insuffisantes — la demande est trop courte ou tronquée |
| 0x0C | N/A | Etat incorrect |

### 7.4.5.9 Restart_Connections

Le service Restart_Connections peut uniquement être envoyé à l'instance de classe de l'objet de planification (instance zéro). Il doit être utilisé pour interrompre toutes les connexions qui utilisent une liaison particulière. Suite à l'interruption de toutes les connexions, l'appareil CO doit tenter de les rétablir. Le paramètre de demande de ce service doit être structuré comme l'attribut d'instance 0x01. L'état de réponse de ce service doit être spécifié conformément au Tableau 48.

**Tableau 48 – Descriptions de l'erreur d'état de Restart_Connections**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | N/A | Success |
| 0x04 | N/A | Erreur de syntaxe du chemin |
| 0x08 | N/A | Service non mis en œuvre (s'il est adressé à une instance non nulle) |

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x10 | N/A | Conflit d'état de l'appareil (impossible d'interrompre les connexions) |
| 0x13 | N/A | Données de demande insuffisantes — la demande est trop courte ou tronquée |
| 0xDO | N/A | Aucune connexion planifiée sur cette liaison |

### 7.4.6 Session de planification classique

Une session de planification classique doit utiliser ces services:

a) create;

b) read;

c) write;

d) change_start;

e) break_connections;

f) conditional_write;

g) forced_write;

h) change_complete.

La procédure ci-dessous décrit l'utilisation des services dans une session de planification classique:

1) Le LSS doit commencer à envoyer un message ***create*** à une instance zéro de l'objet de planification. Ce message doit contenir un chemin entre le CO et la liaison. Ce chemin doit déterminer les connexions présentant un intérêt pour le LSS.

2) L'objet de planification doit créer une instance de lui-même au service de ce LSS et répondre par le numéro d'instance auquel toutes les communications ultérieures doivent être envoyées.

3) L'instance qui vient d'être créée doit réinitialiser un temporisateur de 60 s dès la réception d'un message. Si ce temporisateur expire, l'instance doit se supprimer elle-même en annulant toutes les modifications en cours. En fonctionnement normal, le LSS doit envoyer une commande à l'objet de planification au moins une fois toutes les 15 s, en permettant de supprimer quelques messages sans supprimer inutilement une instance.

4) Le LSS peut éventuellement envoyer des commandes ***read*** à l'instance qui vient d'être créée afin de transférer les informations de connexion depuis l'appareil CO. Les informations de connexion transmises doivent contenir les indices utilisés pour identifier de manière unique les connexions en interne dans l'appareil CO.

5) Le LSS doit refléter ces indices sur des commandes ***write*** consécutives, de sorte que le CO puisse déterminer les connexions auxquelles s'applique la commande ***write***. L'émission de commandes ***read*** doit être facultative, de manière à pouvoir obtenir par d'autres moyens les informations de connexion pour le LSS. Par exemple, dans un LSS/PS intégré, le PS doit fournir ces informations au LSS.

6) Après avoir obtenu les informations de connexion auprès de chaque CO souhaitant planifier des données sur la liaison, le LSS doit calculer une planification provisoire.

7) Le LSS doit alors émettre la séquence de commandes suivante vers chaque appareil CO: ***change start***, un ou plusieurs ***writes*** et ***break connections***. Cette séquence peut être réalisée parallèlement à chacun des appareils CO. Le type de commande ***write*** (forcée ou conditionnelle) doit dépendre de la manière dont les informations de connexion ont été obtenues. Si le LSS a indépendamment assuré que les informations de connexion sont valides, il peut utiliser la commande ***forced write***. Sinon, il doit utiliser une commande ***conditional write***. Le LSS peut savoir que les informations de connexion sont valides en obtenant la ressource d'édition du CO avant les informations de connexion.

8) Dès la réception de la commande ***conditional write***, l'objet de planification doit vérifier que les paramètres de la connexion n'ont pas été modifiés depuis la dernière lecture. Si la

connexion a été modifiée, l'objet de planification ne doit pas utiliser les informations de planification pour l'indice dont les paramètres de connexion ont été modifiés. L'objet de planification du CO peut le faire par tous les moyens dont il dispose.

9) L'objet de planification doit être déchargé de toute responsabilité de vérification de la validité des commandes ***forced writes***. La responsabilité doit être assumée par le LSS.

10) Lorsque les commandes ***Break_Connection*** destinées à chacun des appareils CO ont abouti, le LSS doit émettre une commande ***Change_Complete*** destinée à chaque appareil CO en indiquant que la planification préalablement transmise via les commandes ***writes*** est à présent valide. La commande ***Change_Complete*** doit inclure les nouveaux mots de passe et chemin CO.

11) La session doit se terminer par la suppression de l'instance de l'objet de planification.

## 7.5 Objet d'interface TCP/IP

### 7.5.1 Présentation

L'objet d'interface TCP/IP fournit le mécanisme de configuration de l'interface TCP/IP d'un appareil.

NOTE 1 Des exemples d'éléments configurables incluent l'adresse IP de l'appareil, le masque de réseau et l'adresse de passerelle.

Le port physique associé à l'objet d'interface TCP/IP doit être un port prenant en charge le protocole TCP/IP.

NOTE 2 Par exemple, un objet d'interface TCP/IP peut être associé à: un port Ethernet ISO/CEI 8802-3, un port ATM, un port série exécutant SLIP, un port série exécutant PPP.

L'objet d'interface TCP/IP offre un attribut qui identifie l'objet spécifique à la liaison pour le port physique associé. En règle générale, l'objet spécifique à la liaison est censé fournir des compteurs et des attributs de configuration spécifiques à la liaison.

Chaque appareil doit prendre en charge exactement une instance de l'objet d'interface TCP/IP pour chaque port TCP/IP qu'il contient.

### 7.5.2 Historique de révision

Le Tableau 49 montre l'historique de révision pour l'objet d'interface TCP/IP.

**Tableau 49 – Historique de révision**

| Revision | Motif de la mise à jour de la définition de l'objet |
|---|---|
| 1 | Révision initiale de la définition de cet objet |
| 2 | Addition des attributs 10 (SelectACD) et 11 (LastConflictDetected) pour l'instance ACD<br>Addition de l'attribut 12, QuickConnect, pour l'instance<br>Addition des bits 5 (Interface Configuration Pending) et 6 (AcdStatus) à l'attribut 1, Etat<br>Addition des bits 6 (Interface Configuration Change Requires Reset) et 7 (AcdCapable) à l'attribut 2, Configuration Capability |
| 3 | Ajout du bit 7 (AcdFault) à l'attribut 1, Etat |

### 7.5.3 Attributs de classe

L'objet TCP/IP Interface doit prendre en charge les attributs de classe tel que spécifié dans le Tableau 50.

**Tableau 50 – Attributs de classe de l'objet d'interface TCP/IP**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x01** | Obligatoire | Get | NV | Revision | UINT | Révision de cet objet | Troisième révision, valeur = 3 |
| **0x02** | Sous condition[a] | Get | NV | Max instance | UINT | Nombre maximal d'instances d'un objet créées dans ce niveau de classe de l'appareil | Nombre d'instances le plus élevé d'un objet créé à ce niveau hiérarchique de classe. |
| **0x03** | Sous condition[a] | Get | NV | Nombre d'instances | UINT | Nombre d'instances d'objets actuellement créées à ce niveau de classe dans l'appareil | Le nombre d'instances d'objet à ce niveau de hiérarchie de classe |
| **0X04 – 0x07** | Ces attributs de classe sont facultatifs | | | | | | |
| [a] Nécessaire si le nombre d'instances est supérieur à 1. | | | | | | | |

### 7.5.4 Attributs d'instance

#### 7.5.4.1 Généralités

L'objet TCP/IP Interface doit prendre en charge les attributs d'instance tel que spécifié dans le Tableau 51.

**Tableau 51 – Attributs d'instance de l'objet d'interface TCP/IP**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x01** | Obligatoire | Get | V | Etat | DWORD | Etat de l'interface | Voir 7.5.4.2 |
| **0x02** | Obligatoire | Get | NV | Capacité de configuration | DWORD | Balises de capacité d'interface | Matriciel de balises de capacité. Voir 7.5.4.3 |
| **0x03** | Obligatoire | Get<br>Set est condi-tionnel[a] | NV | Contrôle de configuration | DWORD | Balises de contrôle d'interface | Matriciel de balises de contrôle. Voir 7.5.4.4 |
| **0x04** | Obligatoire | Get | NV | Objet de liaison physique | STRUCT de taille variable | Chemin vers l'objet de liaison physique | Voir 7.5.4.5 |
| | | | | Taille du chemin | UINT | Taille du chemin | Nombre d'UINT dans le chemin |
| | | | | Chemin | EPATH rempli | Segments logiques identifiant l'objet de liaison physique | Le chemin est limité à un segment de classe logique et à un segment d'instance logique. La taille maximale est de 12 octets. |
| **0x05** | Obligatoire | Get<br>Set est recom-mandé | NV lorsque la méthode de configuration est 0.<br>V lorsqu'elle est obtenue via BOOTP ou DHCP | Configuration d'interface | STRUCT de taille variable | Configuration d'interface réseau TCP/IP | Voir 7.5.4.6 |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| | | | | Adresse IP | UDINT | Adresse IP de l'appareil | Une valeur 0 indique qu'aucune adresse IP n'a été configurée. Sinon, l'adresse IP doit être définie sur une adresse de classe A, B ou C valide et ne doit pas l'être sur l'adresse en boucle (127.0.0.1) |
| | | | | Masque de réseau | UDINT | Masque de réseau de l'appareil | Une valeur 0 indique qu'aucun masque de réseau n'a été configuré |
| | | | | Adresse de la passerelle | UDINT | adresse de la passerelle | Une valeur 0 indique qu'aucune adresse IP n'a été configurée. Sinon, l'adresse IP doit être définie sur une adresse de classe A, B ou C valide et ne doit pas l'être sur l'adresse en boucle (127.0.0.1) |
| | | | | Nom du serveur | UDINT | Serveur de noms principal | Une valeur 0 indique qu'aucune adresse de serveur de noms n'a été configurée. Sinon, l'adresse du serveur de nom doit être définie sur une adresse de classe A, B ou C valide |
| | | | | Serveur de noms 2 | UDINT | Serveur de noms secondaire | Une valeur 0 indique qu'aucune adresse de serveur de noms secondaire n'a été configurée. Sinon, l'adresse du serveur de nom doit être définie sur une adresse de classe A, B ou C valide |
| | | | | Nom de domaine | STRING | Nom de domaine par défaut | Caractères ASCII La longueur maximale est de 48 caractères. Doit être rempli jusqu'à un nombre pair de caractères (remplissage exclu de la longueur). Une longueur de 0 doit indiquer qu'aucun nom de domaine n'est configuré |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x06** | Obligatoire | Get<br>Set est facul-tatif[b] | NV | Nom d'hôte | STRING | Nom d'hôte | Caractères ASCII La longueur maximale est de 64 caractères. Doit être rempli jusqu'à un nombre pair de caractères (remplissage exclu de la longueur). Une longueur de 0 doit indiquer qu'aucun nom d'hôte n'est configuré.<br>Voir 7.5.4.6.2 |
| **0x07** | Sous condition[c] | | | Numéro de réseau de sécurité | USINT[6] | | Voir la CEI 61784-3-2 |
| **0x08** | Sous condition[d] | Get<br>Set est condi-tionnel[e] | NV | Valeur TTL | USINT | Valeur TTL de paquets multidiffusion CP 2/2 | Valeur de durée de vie des paquets multidiffusion IP. La valeur par défaut est 1, la valeur minimale est 1 et la valeur maximale est de 255. Voir 7.5.4.8 |
| **0x09** | Sous condition[d] | Get<br>Set est condi-tionnel[e] | NV | Config Mcast | STRUCT de 8 octets | Configuration de l'adresse multidiffusion IP | Voir 7.5.4.9 |
| | | | | Alloc control | USINT | Mot de contrôle d'allocation d'adresse multidiffusion. Détermine la manière d'allouer les adresses | Détermine si des adresses multidiffusion sont générées via un algorithme ou explicitement définies. Voir 7.5.4.9 pour plus de détails |
| | | | | Réservé | USINT | Réservé pour une utilisation future | Doit être égal à 0 |
| | | | | Num mcast | UINT | Nombre d'adresses multidiffusion IP à allouer pour CP 2/2 | Nombre d'adresses multidiffusion IP allouées, commençant à "Mcast start addr".<br>La valeur maximale est spécifique à l'appareil. Toutefois, elle ne doit pas dépasser le nombre de connexions multidiffusion CP 2/2 pris en charge par l'appareil. |
| | | | | Mcast start addr | UDINT | Démarrage de l'adresse multidiffusion à partir de laquelle commencer l'allocation | Adresse multidiffusion IP (classe D). Un bloc d'adresses « Num mcast » alloué commençant avec cette adresse |
| **0x0A** | Sous condition[f] | Set | NV | SelectAcd | BOOL | Active l'utilisation de l'ACD | Activer l'ACD (1, par défaut), désactiver l'ACD (0).<br>Voir 7.5.4.10 |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x0B** | Sous condition[f] | Set | NV | LastConflictDetected | STRUCT de 35 octets | Structure contenant les informations relatives au dernier conflit détecté | Paramètres de diagnostic ACD<br>Voir 7.5.4.11 |
| | | | | AcdActivity | USINT | Etat d'activité ACD lors de la détection du dernier conflit | Activité ACD<br>Par défaut = 0 |
| | | | | RemoteMAC | USINT[6] | Adresse MAC du nœud distant de l'unité PDU ARP où un conflit a été détecté | MAC de l'en-tête du paquet Ethernet<br>Par défaut = 0 |
| | | | | ArpPdu | USINT[28] | Copie de l'unité PDU ARP brute dans laquelle un conflit a été détecté. | PDU ARP<br>Par défaut = 0 |
| **0x0C** | Facultatif | Set | NV | QuickConnect | BOOL | Active/Désactive la fonction QuickConnect | 0 = Désactivée (par défaut)<br>1 = Activée<br>Voir 7.5.4.12 |

NOTE L'ACD est spécifié en 7.5.7.

[a] Set est requis à moins que la méthode de configuration ne soit exclusivement sélectionnée à l'aide des paramètres matériels.

[b] Set est facultatif lorsque l'attribut de configuration d'interface ne peut pas être défini.

[c] Cet attribut est obligatoire pour les appareils de sécurité CPF 2. Les appareils non relatifs à la sécurité ne doivent pas mettre en œuvre cet attribut.

[d] Si « valeur TTL » ou « Mcast config » est mis en œuvre, les deux doivent l'être.

[e] Si « valeur TTL » ou « Mcast config » est mis en œuvre comme étant définissable, les deux doivent l'être en tant que tel.

[f] Obligatoire si l'appareil met en œuvre l'ACD.

### 7.5.4.2 Etat

L'attribut Etat est un matriciel indiquant l'état de l'interface réseau TCP/IP (voir Tableau 52). Consulter le diagramme d'états à la Figure 21 pour obtenir une description des états d'objet en fonction de leur attribut Etat.

**Tableau 52 – Bits Etat**

| Bit(s) | Nom | Définition |
|---|---|---|
| **0-3** | Interface configuration status | Indique l'état de l'attribut de configuration de l'interface.<br>0 = L'attribut de configuration de l'interface n'a pas été configuré<br>1 = L'attribut de configuration de l'interface contient une configuration valide obtenue auprès de BOOTP, DHCP ou de la mémoire non volatile<br>2 = L'attribut de configuration de l'interface contient une configuration valide, obtenue auprès des paramètres matériels (poussoir, molette,...)<br>3-15 = Réservé pour une utilisation future |
| **4** | Mcast pending | Indique une modification de configuration en attente dans la valeur TTL et/ou les attributs de configuration Mcast. Une valeur doit être attribuée à ce bit lorsque la valeur TTL ou l'attribut de configuration Mcast est défini, ce bit doit être supprimé au prochain démarrage de l'appareil |

| Bit(s) | Nom | Définition |
|---|---|---|
| **5** | Interface configuration pending | Indique une modification de la configuration en attente dans l'attribut de configuration de l'interface. Ce bit doit avoir la valeur 1 (TRUE) lorsque les attributs de configuration de l'interface sont définis et que l'appareil nécessite une réinitialisation afin que la modification de la configuration prenne effet (comme cela est indiqué dans l'attribut de capacité de configuration).<br>L'objectif du bit de configuration d'interface en attente est de permettre au logiciel client de détecter si la configuration IP d'un appareil est modifiée, avec une prise d'effet uniquement en cas de réinitialisation de l'appareil. |
| **6** | AcdStatus | Indique lorsqu'un conflit d'adresses IP a été détecté par l'ACD.<br>Ce bit doit être mis par défaut à 0 (FALSE) au démarrage. Si l'ACD est pris en charge et activé, ce bit doit être mis à 1 (TRUE) chaque fois qu'un conflit d'adresses est détecté tel que défini par les transitions [ConflictDetected] dans la Figure 23 (Comportement ACD). |
| **7** | AcdFault | Indique lorsqu'un conflit d'adresses IP a été détecté par l'ACD ou que la défense a échoué, et que la configuration d'interface actuelle ne peut pas être utilisée en raison de ce conflit. Ce bit DOIT être à 1 (TRUE) si un conflit d'adresses a été détecté et que cette interface est actuellement à l'état ACD Notification & FaultAction ou AcquireNewIpv4Parameters tel que défini en 7.5.7. Sinon, il DOIT être à 0 (FALSE).<br>Noter que, lorsque ce bit est défini, ce port CPF 2 ne sera pas utilisable. Néanmoins, pour les appareils avec des ports multiples, ce bit permet de déterminer si le port a un défaut ACD et ne peut donc pas être utilisé. |
| **8-31** | Reserved | Réservé pour une utilisation future et doit être défini sur zéro |

### 7.5.4.3 Capacité de configuration

L'attribut Capacité de configuration est un matriciel qui indique le support de l'appareil pour une capacité de configuration facultative du réseau. Les éléments possibles de capacité de configuration figurent dans le Tableau 53. Il n'est pas indispensable que les appareils prennent en charge un élément particulier. Toutefois, ils doivent prendre en charge au moins une méthode permettant d'obtenir une adresse IP initiale.

**Tableau 53 – Bits de capacité de configuration**

| Bit | Nom | Définition |
|---|---|---|
| **0** | BOOTP client | 1 (TRUE) doit indiquer que l'appareil est en mesure d'obtenir sa configuration de réseau par l'intermédiaire de BOOTP |
| **1** | DNS client | 1 (TRUE) doit indiquer que l'appareil est en mesure de résoudre les noms d'hôte en interrogeant un serveur DNS |
| **2** | DHCP client | 1 (TRUE) doit indiquer que l'appareil est en mesure d'obtenir sa configuration de réseau par l'intermédiaire de DHCP |
| **3** | DHCP-DNS update | Doit avoir la valeur 0, comportement à définir dans une future version de la présente norme |
| **4** | Configuration settable | 1 (TRUE) doit indiquer qu'une valeur peut être allouée à l'attribut de configuration de l'interface. Certains appareils (un PC ou un poste de travail, par exemple) peuvent ne pas permettre de définir la configuration d'interface par l'intermédiaire de l'objet d'interface TCP/IP |
| **5** | Hardware configurable | 1 (TRUE) doit indiquer que le membre adresse IP de l'attribut de configuration d'interface peut être obtenu à partir des paramètres matériels (par exemple, poussoir, molette, etc.).<br>Si ce bit a la valeur FALSE l'attribut d'instance Etat (0x01), la valeur du champ d'état de la configuration d'interface ne doit jamais avoir la valeur 2 (l'attribut de configuration d'interface contient une configuration valide, obtenue depuis les paramètres matériels) |

| Bit | Nom | Définition |
|---|---|---|
| **6** | La modification de la configuration d'interface nécessite une réinitialisation | 1 (TRUE) doit indiquer que l'appareil nécessite un redémarrage afin que la modification de l'attribut de configuration d'interface prenne effet.<br>Si ce bit a la valeur FALSE, une modification de l'attribut de configuration d'interface prendra effet immédiatement. |
| **7** | AcdCapable | (1) TRUE doit indiquer que l'appareil est compatible ACD |
| **8-31** | Réservé | Réservé pour une utilisation future et doit être défini sur zéro |

#### 7.5.4.4 Contrôle de configuration

##### 7.5.4.4.1 Structure du contrôle de configuration

L'attribut Contrôle de configuration est un matriciel permettant de contrôler les options de configuration de réseau ( voir Tableau 54).

**Tableau 54 – Bits de contrôle de configuration**

| Bit | Nom | Définition | |
|---|---|---|---|
| **0-3** | Méthode de configuration | Détermine comment l'appareil doit obtenir sa configuration relative à l'IP | 0 = L'appareil doit utiliser des valeurs de configuration IP affectées de façon statique<br>1 = L'appareil doit obtenir ses valeurs de configuration d'interface par l'intermédiaire de BOOTP.<br>2 = L'appareil doit obtenir ses valeurs de configuration d'interface par l'intermédiaire de DHCP.<br>3-15 = Réservé pour une utilisation future |
| **4** | DNS Enable | Si 1 (TRUE), l'appareil doit résoudre les noms d'hôte en interrogeant un serveur DNS | |
| **5-31** | Réservé | Réservé pour une utilisation future et doit être défini sur zéro | |

##### 7.5.4.4.2 Méthode de configuration

La méthode de configuration détermine comment l'appareil doit obtenir sa configuration relative à l'IP.

- Si la méthode de configuration est 0, l'appareil doit utiliser la configuration IP affectée de façon statique contenue dans l'attribut de configuration d'interface (ou affectée via des méthodes autres que celles de Type 2, comme indiqué ci-dessous).
- Si la méthode de configuration est 1, l'appareil doit obtenir sa configuration IP via BOOTP. Le comportement du client BOOTP doit être conforme à ce qui est défini dans les normes IETF RFC (IETF RFC 951, IETF RFC 1542, IETF RFC 2132 ou leurs successeurs).
- Si la méthode de configuration est 2, l'appareil doit obtenir sa configuration IP via DHCP. Le comportement du client DHCP doit être conforme à ce qui est défini dans les normes IETF RFC (IETF RFC 2131, IETF RFC 2132 ou leurs successeurs).
- Les appareils qui fournissent en option des moyens matériels (par exemple, commutateur rotatif) pour la configuration du comportement d'adressage IP doivent configurer la méthode de configuration afin de refléter la configuration définie via le matériel: 0 si une adresse IP statique a été configurée, 1 si BOOTP a été configuré, 2 si DHCP a été configuré.

Si un appareil a été configuré de façon à obtenir sa configuration via BOOTP ou DHCP, il doit continuer à envoyer des demandes jusqu'à la réception d'une réponse venant du serveur. Les appareils qui choisissent d'utiliser une configuration IP par défaut en cas d'absence de réponse du serveur doivent poursuivre l'émission de demandes jusqu'à réception d'une

réponse ou jusqu'à ce que la méthode de configuration soit modifiée en faveur de la méthode statique.

Lorsque l'appareil reçoit une réponse du serveur, il doit cesser l'envoi de demandes clients BOOTP/DHCP (les clients DHCP doivent suivre le comportement de renouvellement de bail conformément à l'IETF RFC). Il est recommandé que les appareils mettent en œuvre des moyens de détection de liaisons ascendantes et, en cas de détection de liaison ascendante, qu'ils redémarrent la séquence BOOTP ou DHCP initiale. Pour les appareils multiports, le redémarrage de la séquence BOOTP ou DHCP initiale doit uniquement être déclenché si toutes les liaisons externes ont été coupées et lorsque la première liaison ascendante est détectée.

Définir la méthode de configuration sur 0 (adresse statique) doit entraîner la sauvegarde de la configuration d'interface dans la mémoire non volatile.

Il est recommandé de définir la méthode de configuration sur 1 (BOOTP) ou sur 2 (DHCP) pour que l'appareil démarre le client BOOTP/HDCP afin d'obtenir la nouvelle configuration de l'adresse IP. Si l'appareil requiert une réinitialisation afin de démarrer le client BOOTP/DHCP, il doit définir le bit de configuration d'interface en attente et, lors de la réinitialisation de l'appareil, démarrer le client BOOTP/DHCP.

#### 7.5.4.4.3 DNS Enable

Pour les appareils émetteurs qui prennent en charge la résolution des noms d'hôtes cibles via serveur DNS, le bit DNS Enable doit activer (1) et désactiver (0) le client DNS.

### 7.5.4.5 Objet de liaison physique

Cet attribut permet d'identifier l'objet associé au port physique sous-jacent. Il existe deux composants pour cet attribut: une taille de chemin (en UINT) et un chemin. Le chemin doit contenir un segment de classe et un segment d'instance identifiant l'objet de port physique. La taille de chemin maximale est de 6 (dans le cas d'un segment logique de 32 bits pour chacune des classes de l'instance). Un exemple est montré dans le Tableau 55.

**Tableau 55 – Exemple de chemin**

| Chemin | Signification |
|---|---|
| **[20][F6][24][01]** | [20] = type de segment de classe à 8 bits;<br>[F6] = classe d'objet de liaison Ethernet;<br>[24] = type de segment d'instance à 8 bits;<br>[01] = instance 1 |

D'une manière générale, l'objet de liaison physique lui-même gère les compteurs et les attributs de configuration spécifiques à la liaison. Si le port associé à l'objet d'interface TCP/IP a une couche physique Ethernet, cet attribut doit pointer vers une instance de l'objet de liaison Ethernet (voir 7.6). En cas d'interfaces physiques multiples qui correspondent à l'interface TCP/IP, cet attribut doit soit contenir une taille de chemin de 0, soit contenir un chemin vers l'objet représentant une interface de communication interne (souvent utilisée avec un commutateur intégré).

### 7.5.4.6 Configuration d'interface

#### 7.5.4.6.1 Contenu de la configuration d'interface

Cet attribut contient les paramètres de configuration requis pour fonctionner comme un nœud TCP/IP.

Le contenu de l'attribut de configuration d'interface doit dépendre de la configuration de l'appareil pour l'obtention de ses paramètres IP.

- S'il est configuré pour utiliser une adresse IP statique (la valeur de la méthode de configuration est 0), les valeurs de la configuration d'interface doivent être celles qui ont été affectées de manière statique et stockée dans la mémoire non volatile.
- S'il est configuré pour utiliser BOOTP ou DHCP (la valeur de la méthode de configuration est 1 ou 2), les valeurs de configuration d'interface doivent contenir la configuration obtenue du serveur BOOTP ou DHCP. La valeur de l'attribut de configuration d'interface doit être 0 tant que la réponse BOOTP ou DHCP n'a pas été reçue.
- Certains appareils offrent en option des mécanismes non CIP supplémentaires pour le paramétrage de la configuration relative à l'IP (par exemple, une interface de serveur web, un commutateur rotatif pour la configuration de l'adresse IP, etc.). Lorsqu'un tel mécanisme est utilisé, l'attribut de configuration d'interface doit refléter les valeurs de la configuration IP utilisée.

Les composants des attributs de configuration d'interface sont présentés dans le Tableau 56.

**Tableau 56 – Composants de configuration d'interface**

| Nom | Signification |
|---|---|
| **Adresse IP** | L'adresse IP de l'appareil |
| **Masque de réseau** | Masque de réseau de l'appareil. Le masque de réseau est utilisé lorsque le réseau IP a été partitionné en sous-réseaux. Le masque de réseau permet de déterminer si une adresse IP se trouve dans un autre sous-réseau |
| **Adresse de la passerelle** | Adresse IP de la passerelle par défaut de l'appareil. Si une adresse IP de destination se trouve sur un autre sous-réseau, les paquets sont transférés vers la passerelle par défaut pour être acheminés vers le sous-réseau de destination |
| **Nom du serveur** | Adresse IP du serveur de noms principal. Le serveur de noms permet de résoudre les noms d'hôte. Par exemple, ils peuvent se trouver dans un chemin de connexion de bus de terrain |
| **Serveur de noms 2** | Adresse IP du serveur de noms secondaire. Le serveur de noms secondaire est utilisé lorsque le serveur de noms principal n'est pas disponible ou qu'il n'est pas en mesure de résoudre un nom d'hôte |
| **Nom de domaine** | Nom de domaine par défaut. Le nom de domaine par défaut est utilisé pour résoudre les noms d'hôte qui ne sont pas totalement qualifiés. Par exemple, si le nom de domaine par défaut est « network.org » et que l'appareil souhaite résoudre un nom d'hôte « plc », l'appareil tentera de résoudre le nom d'hôte en « plc.network.org » |

#### 7.5.4.6.2 Paramétrage du comportement des attributs

Pour éviter les configurations incomplètes ou incompatibles, les paramètres composant l'attribut de configuration d'interface ne peuvent pas être définis individuellement. Pour modifier l'attribut de configuration d'interface, il convient que l'utilisateur l'extraie en premier lieu, modifie les paramètres de son choix, puis le définisse.

Une tentative pour définir les paramètres de l'attribut de configuration d'interface avec des valeurs non valides (voir le Tableau 51) doit résulter en une réponse d'erreur avec le code de statut 0x09 « Valeur d'attribut non valide » retourné. Dans ce scénario, tous les paramètres de l'attribut de configuration d'interface conservent les valeurs qui existaient avant l'utilisation du service de paramétrage.

Si l'appareil dispose d'une connexion E/S active, il est recommandé que l'appareil rejette la demande de définition des attributs en retournant une réponse d'erreur avec le code de statut 0x10 « Conflit d'état de l'appareil ».

Lorsque la valeur de la méthode de configuration (attribut de contrôle de configuration) est 0, le service de définition d'attribut doit stocker les nouvelles valeurs de configuration d'interface dans la mémoire non volatile. Si l'appareil nécessite une réinitialisation pour la prise d'effet des nouveaux paramètres (bit 6 de la capacité de configuration), l'appareil doit définir le bit de configuration d'interface en attente (bit 5 de l'attribut d'état).

Après avoir stocké les nouvelles valeurs, l'appareil doit envoyer une réponse à l'aide de son adresse IP actuelle (c'est-à-dire l'adresse IP à laquelle a été envoyée la demande de définition des attributs).

Si l'appareil ne nécessite pas de réinitialisation (bit 6 de la capacité de configuration) pour la prise d'effet de la nouvelle configuration IP, après avoir répondu au service de définition, l'appareil doit initier l'application des nouveaux paramètres IP. Si elle est dépendante de la mise en œuvre, cette activité est en règle générale initiée par le service de définition de l'objet d'interface TCP/IP, puis terminée de façon asynchrone par la pile TCP/IP.

Pour avoir un comportement de configuration d'appareil cohérent, il est recommandé que les appareils prennent en charge la définition de l'attribut de configuration d'interface et l'application des nouvelles valeurs d'attribut sans nécessiter de réinitialisation des appareils. Si un appareil ne prend pas en charge la définition de l'attribut de configuration d'interface, l'appareil doit retourner une réponse d'erreur avec le code de statut 0x0E « Attribut non définissable ».

#### 7.5.4.6.3 Application des nouveaux paramètres de configuration

Si l'appareil ne nécessite pas de réinitialisation (bit 6 de la capacité de configuration), après avoir répondu au service de définition, l'appareil doit initier l'application des nouveaux paramètres IP. Lors de l'application de la nouvelle configuration IP, l'appareil doit maintenir la cohérence du contexte de configuration IP dans lequel il fonctionne. Par exemple, l'appareil ne doit pas mélanger d'anciennes et de nouvelles valeurs pour l'adresse IP/le masque réseau/l'adresse de passerelle, et ne doit pas utiliser la nouvelle configuration IP sur des connexions de Type 2 établies à l'aide de la précédente adresse IP.

Lorsqu'une pile TCP/IP est configurée pour utiliser un ensemble spécifique de paramètres IP, un contexte est créé autour de ces paramètres IP. Ce contexte inclut des relations avec la pile TCP/IP, l'environnement de communication de Type 2 et l'application de contrôle parmi les autres entités. Pour permettre l'utilisation d'un autre ensemble de paramètres IP, un nouveau contexte IP doit être créé. L'appareil doit administrer correctement son contexte IP et veiller à la cohérence. Par exemple, une nouvelle adresse IP ne doit pas être utilisée avec un ancien masque de réseau ou une ancienne adresse de passerelle; une ancienne adresse IP ne doit pas être utilisée pour la communication de Type 2 ou autre une fois que la nouvelle est appliquée.

### 7.5.4.7 Nom d'hôte

L'attribut Nom d'hôte contient le nom d'hôte de l'appareil, qui peut être utilisé à titre informatif. L'accès de définition est facultatif lorsque l'attribut de configuration d'interface ne peut pas être défini.

### 7.5.4.8 Valeur TTL

La valeur TTL est celle que l'appareil doit utiliser pour le champ de durée de vie de l'en-tête IP lors de l'envoi de paquets CP 2/2 par multidiffusion IP. Par défaut, la valeur TTL doit être 1. La valeur maximale de TTL est 255. Les paquets à diffusion individuelle doivent utiliser la valeur TTL configurée pour la pile TCP/IP, et pas celle configurée dans cet attribut.

Une fois définie, la valeur TTL doit être sauvegardée dans la mémoire non volatile. Si un appareil ne prend pas en charge l'application de la valeur TTL immédiatement, le bit Mcast pending dans l'attribut Etat de l'interface doit être défini, en indiquant qu'une configuration est en attente. Pour les appareils prenant en charge la valeur TTL immédiatement, s'il existe des connexions multipoints, une erreur de conflit d'état d'objet (0xC) doit être retournée et le bit Mcast pending ne doit pas être défini. Si une nouvelle valeur TTL est en attente, la demande Get_Attribute_Single ou Get_Attribute_All doit renvoyer la valeur en attente. Le bit Mcast pending doit être supprimé au démarrage suivant de l'appareil.

Il convient que les utilisateurs soient très prudents lorsqu'ils attribuent une valeur supérieure à 1 à TTL, afin d'éviter qu'un trafic multidiffusion indésirable ne se propage sur le réseau.

NOTE La CEI 61158-6-2 contient une discussion relative aux considérations de l'utilisateur sur la multidiffusion.

#### 7.5.4.9 Config Mcast

L'attribut Mcast config contient la configuration des adresses IP de multidiffusion de l'appareil à utiliser pour les paquets de multidiffusion CP 2/2. Trois éléments sont constitutifs de la structure de Mcast config: Alloc control, Num mcast et Mcast start addr.

- Alloc control permet de déterminer la manière dont l'appareil doit attribuer des adresses IP de multidiffusion (par un algorithme, une définition explicite, etc.). Le Tableau 57 montre le détail pour alloc control.

**Tableau 57 – Valeurs d'alloc control**

| Valeur | Définition |
|---|---|
| 0 | Les adresses multidiffusion doivent être générées à l'aide de l'algorithme d'allocation par défaut spécifié dans la CEI 61158-6-2. Si cette valeur est attribuée à Set_Attribute_Single ou Set_Attribute_All, les valeurs de Num mcast et Mcast start addr de la demande Set_Attribute doivent être égales à 0 |
| 1 | Les adresses multidiffusion doivent être attribuées en fonction des valeurs spécifiées dans Num mcast et Mcast start addr |
| 2 | Réservé |

- Num mcast est le nombre d'adresses multidiffusion IP attribuées. Le nombre maximal d'adresses multidiffusion est spécifique à l'appareil. Toutefois, il ne doit pas dépasser le nombre de connexions multidiffusion CP 2/2 pris en charge par l'appareil.
- Mcast start addr est l'adresse multidiffusion de départ à partir de laquelle les adresses Num mcast sont attribuées.

Une fois définie, l'attribut Mcast config doit être sauvegardé dans la mémoire non volatile. Si un appareil ne prend pas en charge l'application de l'attribut Mcast config immédiatement, le bit Mcast pending dans l'attribut Etat de l'interface doit être défini, en indiquant qu'une configuration est en attente. Pour les appareils prenant en charge l'application de l'attribut Mcast config immédiatement, s'il existe des connexions multipoints, une erreur de conflit d'état d'objet (0xC) doit être retournée et le bit Mcast pending ne doit pas être défini. Si une nouvelle valeur Mcast Config est en attente, la demande Get_Attribute_Single ou Get_Attribute_All doit renvoyer la valeur en attente. Le bit Mcast pending doit être supprimé au démarrage suivant de l'appareil.

Si les adresses multidiffusion sont générées à l'aide de l'algorithme par défaut, Num mcast et Mcast start addr doivent renvoyer les valeurs générées par l'algorithme.

#### 7.5.4.10 SelectAcd

SelectAcd est un attribut utilisé pour Activer/Désactiver l'ACD.

Si SelectAcd est défini sur 0, l'ACD est désactivé. Si SelectAcd est défini sur 1, l'ACD est activé.

La valeur par défaut de SelectAcd doit être 1, indiquant que l'ACD est activé.

Lorsque la valeur de SelectAcd est modifiée par un service Set_Attribute service, la nouvelle valeur de SelectAcd ne doit pas être appliquée jusqu'à ce que l'appareil ne redémarre.

#### 7.5.4.11 LastConflictDetected

L'attribut LastConflictDetected est un attribut de diagnostic présentant les informations concernant l'état de l'ACD lors de la détection du dernier conflit d'adresses IP. Cet attribut doit être mis à jour par l'appareil chaque fois qu'il reçoit un paquet ARP entrant, qui constitue un conflit avec l'adresse IP de l'appareil tel que décrit dans l'IETF RFC 5227.

Pour réinitialiser cet attribut, le service Set_Attribute_Single est appelé avec une valeur d'attribut de 0. Les valeurs autres que 0 doivent résulter en une réponse d'erreur (code de statut 0x09, Valeur d'attribut non valide).

Trois éléments sont constitutifs de la structure de LastConflictDetected: AcdActivity, RemoteMac et ArpPdu.

- AcdActivity contient le statut de l'algorithme ACD lors de la détection du dernier conflit d'adresse IP. Les activités de l'ACD sont définies dans le Tableau 58.

**Tableau 58 – Valeurs de AcdActivity**

| Valeur | AcdMode | Description |
|---|---|---|
| 0 | NoConflictDetected (par défaut) | Aucun conflit n'a été détecté, car cet attribut a été supprimé dernièrement |
| 1 | ProbeIpv4Address | Dernier conflit détecté à l'état ProbeIpv4Address |
| 2 | OngoingDetection | Dernier conflit détecté à l'état OngoingDetection ou à l'état DefendWithPolicyB ultérieur |
| 3 | SemiActiveProbe | Dernier conflit détecté à l'état SemiActiveProbe ou à l'état DefendWithPolicyB ultérieur |

- RemoteMac contient l'adresse MAC source ISO/CEI 8802-3 de l'en-tête du paquet Ethernet reçu qui a été envoyé par un appareil faisant état d'un conflit.
- ArpPdu contient l'unité PDU de la réponse ARD au format binaire. ArpPdu doit être une copie du message ARP ayant causé le conflit d'adresses. Ce doit être une copie brute du message ARP tel qu'il apparaît sur le réseau Ethernet, c'est-à-dire: ArpPdu[1] contient le premier octet de l'ArpPdu reçu (voir Tableau 59).

**Tableau 59 – ArpPdu – Unité PDU de la réponse ARP au format binaire**

| Taille du champ (en octets) | Description du champ | Valeur du champ |
|---|---|---|
| 2 | Type d'adresse du matériel | 1 pour Ethernet H/W |
| 2 | Type d'adresse du protocole | 0x800 pour IP |
| 1 | HADDR LEN | 6 pour Ethernet H/W |
| 1 | PADDR LEN | 4 pour IP |
| 2 | OPERATION | 1 pour Demande ou 2 pour Réponse |
| 6 | SENDER HADDR | Adresse H/W de l'expéditeur |
| 4 | SENDER PADDR | Adresse proto de l'expéditeur |
| 6 | TARGET HADDR | Adresse H/W de la cible |
| 4 | TARGET PADDR | Adresse proto de la cible |

#### 7.5.4.12 QuickConnect™ [4]

L'attribut QuickConnect doit activer (1) ou désactiver (0) la fonction QuickConnect. La valeur par défaut de l'attribut doit être 0.

NOTE Le profil de mise en œuvre QuickConnect™ est spécifié par ODVA, Inc (voir <www.odva.org>).

### 7.5.5 Services communs

#### 7.5.5.1 Généralités

L'objet Interface TCP/IP doit prendre en charge les services communs tel que spécifié dans le Tableau 60.

**Tableau 60 – Services communs de l'objet d'interface TCP/IP**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| **0x01** | Facul-tatif | Facul-tatif | Get_Attribute_All | Renvoie une liste prédéfinie des attributs de cet objet (voir la définition de réponse Get_Attribute_All en 7.5.5.2) |
| **0x02** | N/A | Facul-tatif | Set_Attribute_All | Modifie tous les attributs définissables |
| **0x0E** | Obli-gatoire | Obli-gatoire | Get_Attribute_Single | Renvoie le contenu de l'attribut spécifié |
| **0x10** | N/A | Obli-gatoire | Set_Attribute_Single | Modifie un seul attribut |

#### 7.5.5.2 Réponse Get_Attribute_All

Au niveau de classe, la réponse Get_Attributes_All doit contenir les attributs de classe dans l'ordre numérique, jusqu'au dernier attribut mis en œuvre. Tous les attributs non mis en œuvre dans la réponse doivent utiliser les valeurs d'attribut par défaut.

Pour les attributs d'instance, les attributs doivent être renvoyés dans l'ordre numérique jusqu'au dernier attribut mis en œuvre. La réponse Get_Attribute_All doit être telle que spécifiée dans le Tableau 61.

**Tableau 61 – Format de réponse Get_Attribute_All**

| ID attribut | Taille (octets) | Sommaire |
|---|---|---|
| **1** | 4 | Etat |
| **2** | 4 | Capacité de configuration |
| **3** | 4 | Contrôle de configuration |
| **4** | 2 | Objet de liaison physique, taille de chemin |
| | Variable, 12 octets max | Objet de liaison physique, chemin (si la taille de chemin est non nulle) |

[4] QuickConnect™ est une appellation commerciale d'ODVA, Inc. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que la CEI approuve ou recommande le détenteur de l'appellation commerciale ou l'un quelconque de ses produits. La conformité à ce profil n'exige pas l'utilisation de l'appellation commerciale QuickConnect™. L'utilisation de l'appellation commerciale QuickConnect™ exige l'obtention d'une autorisation auprès d'ODVA, Inc.

| ID attribut | Taille (octets) | Sommaire |
|---|---|---|
| **5** | 4 | Adresse IP |
| | 4 | Masque de réseau |
| | 4 | Adresse de la passerelle |
| | 4 | Nom du serveur |
| | 4 | Serveur de noms secondaire |
| | 2 | Longueur du nom de domaine |
| | Variable, égale à la longueur du nom de domaine | Nom de domaine |
| | 1 | Octet de remplissage uniquement si la longueur du nom de domaine est impaire |
| **6** | 2 | Longueur du nom d'hôte |
| | Variable, égale à la longueur du nom d'hôte | Nom d'hôte |
| | 1 | Octet de remplissage uniquement si la longueur du nom d'hôte est impaire |
| **7** | 6 | Voir la CEI 61784-3-2.<br>Absent si l'attribut 7 n'est pas mis en œuvre. Doit être égal à 0 si des attributs supplémentaires supérieurs à l'ID attribut 7 sont inclus |
| **8** | 1 | Valeur TTL.<br>Absent si l'attribut 8 n'est pas mis en œuvre. Doit être égal à 0 si des attributs supplémentaires supérieurs à l'ID attribut 8 sont inclus |
| **9** | 8 | Config Mcast.<br>Absent si l'attribut 9 n'est pas mis en œuvre. Doit être égal à 0 si des attributs supplémentaires supérieurs à l'ID attribut 9 sont inclus |
| **10** | 1 octet | Valeur SelectACD.<br>Absent si l'attribut 10 n'est pas mis en œuvre. Doit être égal à 0 si des attributs supplémentaires supérieurs à l'ID attribut 10 sont inclus. |
| **11** | 1 octet | AcdActivity<br>Absent si l'attribut 11 n'est pas mis en œuvre. Doit être égal à 0 si des attributs supplémentaires supérieurs à l'ID attribut 11 sont inclus. |
| | 6 octets | RemoteMAC<br>Absent si l'attribut 11 n'est pas mis en œuvre. Doit être égal à 0 si des attributs supplémentaires supérieurs à l'ID attribut 11 sont inclus. |
| | 28 octets | ArpPdu<br>Absent si l'attribut 11 n'est pas mis en œuvre. Doit être égal à 0 si des attributs supplémentaires supérieurs à l'ID attribut 11 sont inclus. |
| **12** | 1 octet | QuickConnect<br>Absent si l'attribut 12 n'est pas mis en œuvre. |

Les longueurs de chemin d'objet de liaison physique, de nom de domaine et de nom d'hôte sont inconnues avant d'émettre la demande de service Get_Attribute_All. Les implémenteurs doivent se préparer à accepter une réponse contenant les tailles maximales de chemin d'objet de liaison physique (6 UINT), de nom de domaine (48 USINT) et de nom d'hôte (64 USINT).

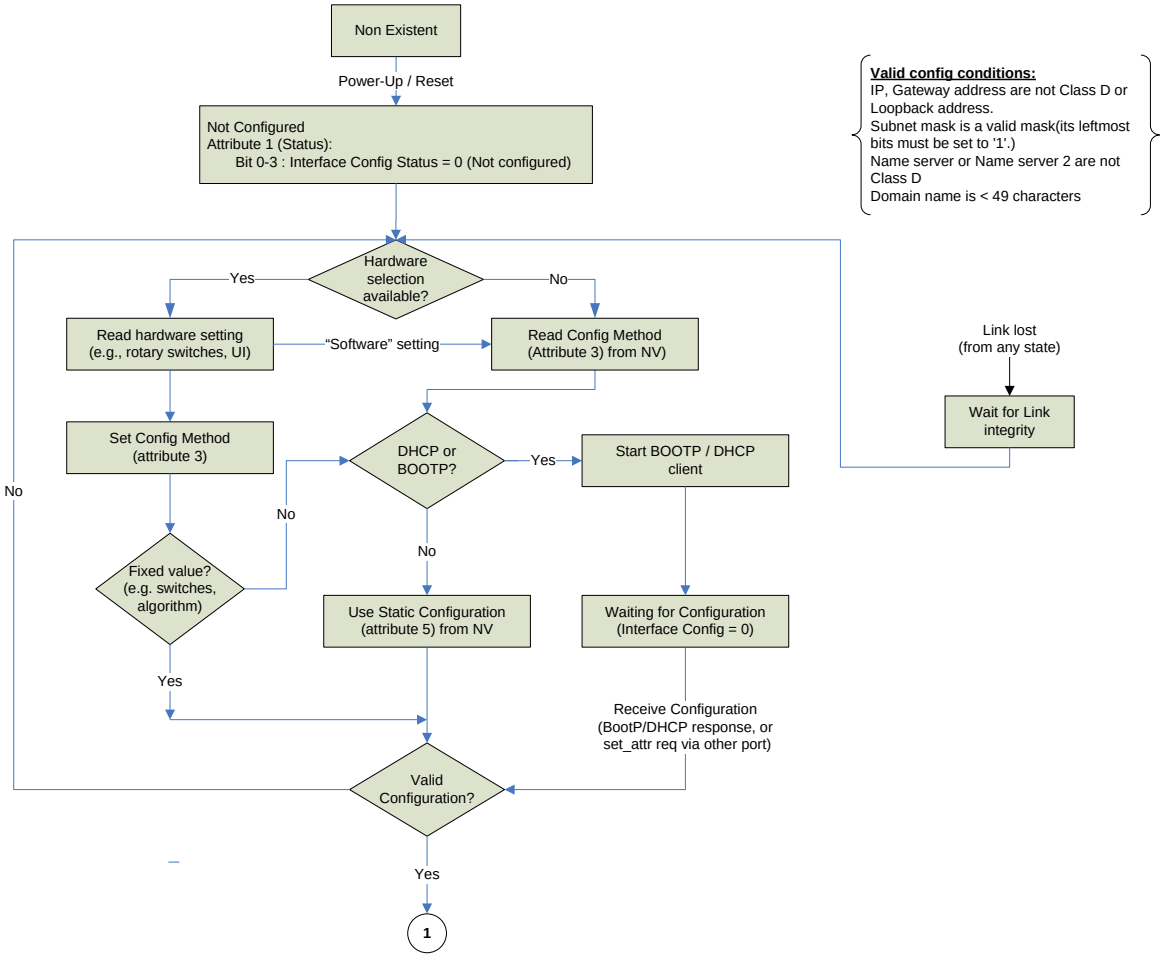
### 7.5.5.3 Demande Set_Attribute_All

La demande Set_Attribute_All de l'instance doit contenir l'attribut Contrôle de configuration, suivi de l'attribut de configuration d'interface.

## 7.5.6 Comportement

Le comportement de l'objet d'interface TCP/IP doit être tel qu'illustré dans le diagramme de transition d'états (voir Figure 21 et Figure 22).

A noter que l'obtention d'une image exécutable initiale par l'intermédiaire de BOOTP/TFTP ne doit pas être considérée dans le cadre du comportement de l'objet d'interface TCP/IP. Les appareils sont libres de mettre en œuvre ce type de comportement. Toutefois, il doit être considéré comme s'étant produit à l'état « inexistant ».

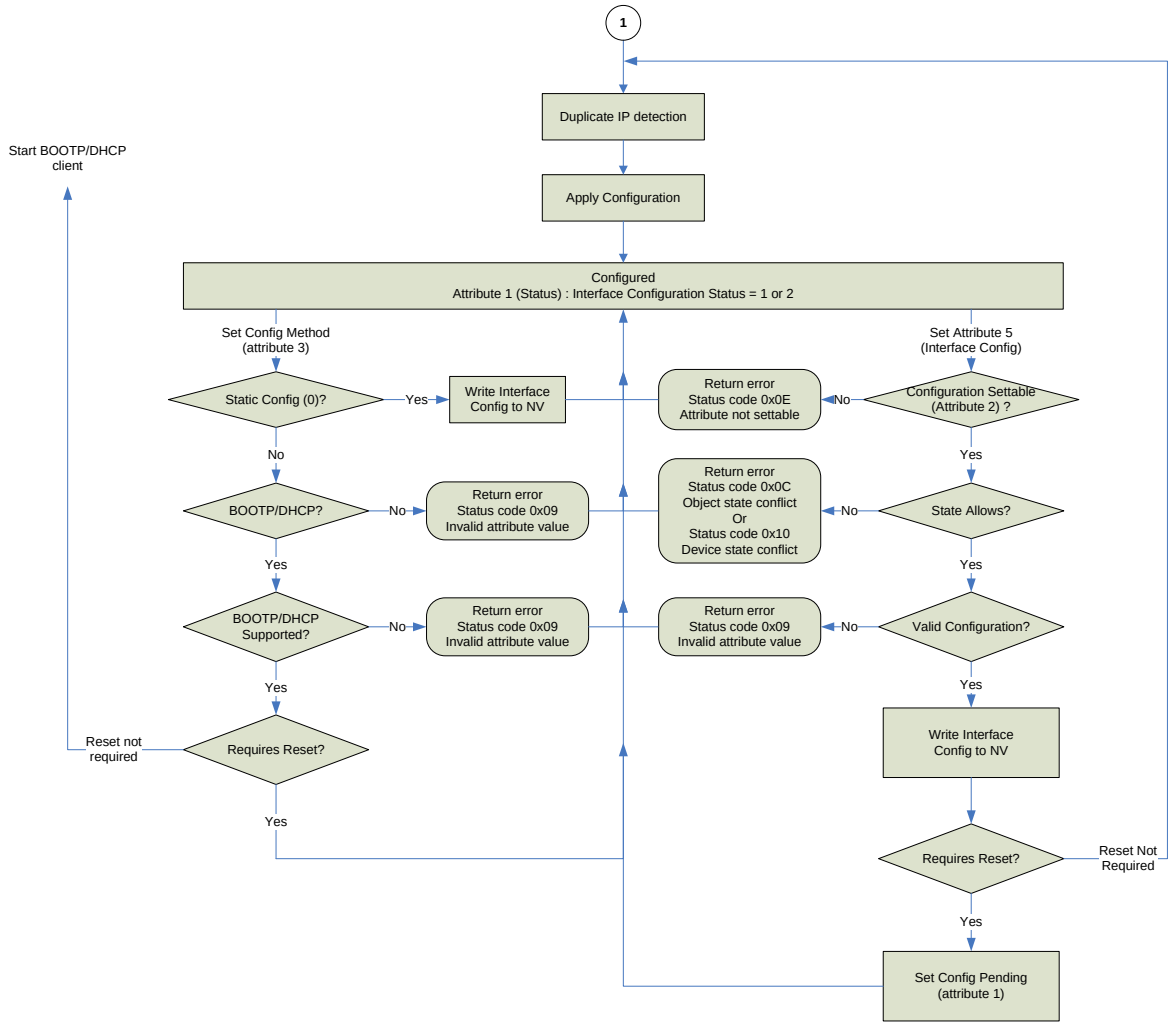



**Légende**

| Anglais | Français |
|---|---|
| Non-existent | Non existant |
| Powerup/reset | Mise sous tension/réinitialisation |
| Not configured | Non configuré |
| Attribute 1 (Status): | Attribut 1 (statut): |
| Bit 0-3: Interface Config Status = 0 (Not configured) | Bit 0-3: statut de configuration d'interface = 0 (non configuré) |
| Hardware selection available? | Sélection du matériel disponible? |
| Yes / No | Oui / Non |

| Anglais | Français |
|---|---|
| Read hardware setting (e.g. rotary switches, UI) | Lecture des paramètres matériels (ex: commutateurs rotatifs, interface utilisateur) |
| « Software » setting | Paramétrage « logiciel » |
| Read Config Method (Attribute 3) from NV | Lecture méthode configuration (attribut 3) de NV |
| Set Config Method (attribute 3) | Lecture méthode configuration (attribut 3) |
| DHCP or BOOTP? | DHCP ou BOOTP? |
| Start BOOTP / DHCP client | Démarrer le client BOOTP / DHCP |
| Link lost (from any state) | Liaison perdue (tout état) |
| Wait for Link Integrity | Attendre l'intégrité de la liaison |
| Fixed values?(e.g. switches, algorithm) | Valeurs fixes? (commutateurs, algorithme) |
| Use static configuration (attribute 5) from NV | Utiliser configuration statique (attribut 5) de NV |
| Waiting for configuration (Interface Config = 0) | Attente de la configuration (config. Interface = 0) |
| Receive configuration (BOOTP/DHCP response, or set_attr req via other port) | Réception de la configuration (réponse BOOTP/DHCP ou demande set_attr via un autre port) |
| Valid configuration? | Configuration valide? |
| Valid config conditions: | Conditions de configuration valide: |
| IP, Gateway address are not Class D or Loopback address. | Les adresses IP et de la passerelle ne sont pas de la classe D ou ne sont pas des adresses en boucle. |
| Subnet mask is a valid mask (its leftmost bits must be set to 1) | Le masque de sous-réseau est un masque valide (ses bits de gauche doivent être définis sur 1) |
| Name server or name server 2 are not class D | Serveur de noms et serveur de noms 2 ne sont pas de classe D |
| Domain name is < 49 characters | Nom de domaine < 49 caractères |

**Figure 21 – Diagramme de transition d'états de l'objet d'interface TCP/IP**




**Légende**

| Anglais | Français |
|---|---|
| Duplicate IP detection | Dupliquer la détection IP |
| Start BOOTP / HDCP client | Démarrer le client BOOTP / HDCP |
| Apply Configuration | Appliquer la configuration |
| Configured<br>Attribute 1 (Status): Interface Configuration Status = 1 or 2 | Configuré<br>Attribut 1 (Status): Interface Configuration Status = 1 ou 2 |
| Set config method (attribute 3) | Définir méthode de configuration (attribut 3) |
| Set Attribute 5 (interface config) | Définir attribut 5 (configuration d'interface) |
| Static Config (0)? | Configuration statique (0)? |
| Yes / No | Oui / Non |
| Write interface config to VN | Ecrire la configuration d'interface dans le NV |
| Return error<br>Status code 0x0E<br>Attribute not settable | Erreur de retour<br>Code statut 0x0E<br>Attribut non définissable |
| Configuration settable (attribute 2)? | Configuration définissable (attribut 2)? |
| Return error<br>Status code 0x09<br>Invalid attribute value | Erreur de retour<br>Code statut 0x09<br>Valeur d'attribut non valide |

| Anglais | Français |
|---|---|
| Return error<br>Status code 0x0C<br>Object state conflict or<br>Status Code 0x10<br>Device state conflict | Erreur de retour<br>Code statut 0x0C<br>Conflit d'état d'objet ou<br>Code statut 0x10<br>Conflit d'état d'appareil |
| State allows? | Autorisation par l'état? |
| BOOTP/DHCP supported? | Prise en charge BOOTP/DHCP? |
| Valid configuration? | Configuration valide? |
| Reset not required | Réinitialisation non requise |
| Requires reset? | Réinitialisation requise? |
| Set config pending (attribute 1) | Définition de la configuration en attente (attribut 1) |

**Figure 22 – Diagramme de transition d'états de l'objet d'interface TCP/IP**

### 7.5.7 Détection de conflit d'adresses (ACD)

#### 7.5.7.1 Généralités

##### 7.5.7.1.1 Exigences ACD pour les appareils de Type 2

La détection de conflit d'adresses (ACD) est un mécanisme que peuvent utiliser les appareils pour détecter et agir sur les conflits d'adresses IPv4.

Le mécanisme ACD spécifié dans le présent Paragraphe 7.5.7 est conforme à l'IETF RFC 5227. Les exigences spécifiées dans l'IETF RFC 5227 sont incluses par référence, sauf en cas de remplacement par des exigences normatives en 7.5.7. Le présent Paragraphe 7.5.7 spécifie des exigences supplémentaires pour les appareils de Type 2 relatifs au mécanisme ACD IPv4.

Les exigences ACD suivantes s'appliquent aux appareils de Type 2:

- ACD est recommandé;
- un appareil mettant en œuvre l'ACD doit être conforme au mécanisme ACD spécifié dans l'IETF RFC 5227, sauf en cas de remplacement par des exigences normatives dans la présente Annexe;
- un appareil exécutant l'ACD doit exécuter le mécanisme ACD indépendamment de la méthode employée par l'appareil pour l'obtention de ses paramètres IP;
- un appareil exécutant l'ACD, et n'exécutant pas l'algorithme QuickConnect, doit exécuter le mécanisme ACD avant d'utiliser un ensemble de paramètres IP;
- les appareils doivent répondre aux sondes ARP.

#### 7.5.7.1.2 Aperçu du mécanisme ACD dans l'IETF RFC 5227

Le mécanisme IPv4 ACD décrit dans l'IETF RFC 5227 implique les activités suivantes:

- échantillonnage d'adresse initiale et détection de conflit – avant d'utiliser une adresse IP, l'appareil émet des sondes ARP pour détecter si l'adresse est utilisée par un autre appareil;
- annonce de l'adresse – un paquet de demande ARP est envoyé après avoir déterminé qu'il n'existe aucun autre conflit d'adresse IP;
- détection de conflit continue – processus continu en vigueur tant que l'appareil utilise une adresse IP;
- défense d'adresse – procédure utilisée pour résoudre un conflit d'adresses.

### 7.5.7.2 Comportement ACD

Les principes de fonctionnement du mécanisme ACD sont spécifiés dans l'IETF RFC 5227.

En particulier, l'IETF RFC 5227, 2.1, exige qu'« un hôte mettant en œuvre la présente spécification doit effectuer des tests afin de vérifier si l'adresse est déjà utilisée lorsqu'une modification de l'état d'une liaison signale qu'un câble Ethernet a été connecté ».

Par ailleurs, le présent Paragraphe 7.5.7 détaille davantage encore le comportement ACD comme suit.

– Les activités de l'ACD sont regroupées en régions, notamment:
  - la phase active;
  - la phase passive;
  - la phase semi-active.
– Un ensemble plus complet de transitions link_up est inclus dans le diagramme.
– Les appareils à port simple et à ports multiples sont compatibles.

Un appareil doit mettre en œuvre le mécanisme ACD avec le comportement spécifié dans la Figure 23.

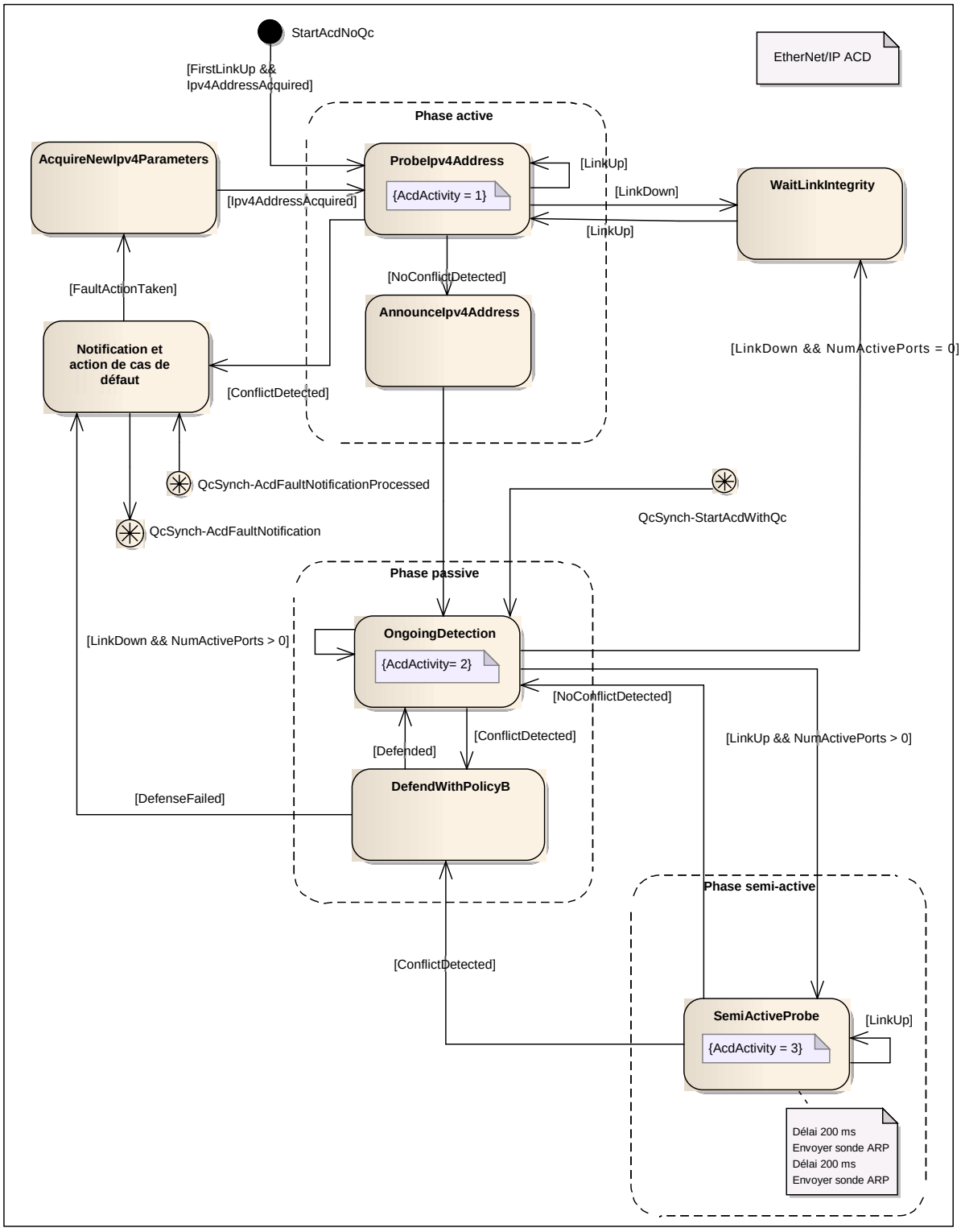



**Figure 23 – Comportement ACD**

#### 7.5.7.3 Détails ACD

##### 7.5.7.3.1 Contraintes temporelles IPv4 ACD

L'IETF RFC 5227, 1.1 spécifie un nombre de contraintes temporelles. Certaines de ces contraintes ne sont pas optimales pour les applications et réseaux de Type 2. Plus particulièrement, les délais de démarrage causés par les intervalles des sondes ARP seraient inacceptables dans un contexte industriel. L'adaptation des valeurs alternatives spécifiées est cohérente avec l'IETF RFC 5227, 1.3 concernant l'applicabilité.

Les appareils dotés de l'ACD doivent mettre en œuvre les valeurs alternatives suivantes pour les paramètres définis dans l'IETF RFC 5227, 1.1.

- PROBE_WAIT: 200 ms (délai aléatoire initial)
- PROBE_NUM: 4 (nombre de paquets de sondes)
- PROBE_MIN: 200 ms (délai minimal jusqu'à la sonde répétée)
- PROBE_MAX: 200 ms (délai minimal jusqu'à la sonde répétée)
- ANNOUNCE_WAIT: 200 ms (délai avant annonce)
- ANNOUNCE_INTERVAL: 2 s (délai entre les paquets d'annonce)
- DEFEND_INTERVAL: 2 s (intervalle minimal entre les ARP défensives)

Les appareils doivent se conformer à la temporisation indiquée dans l'IETF RFC 5227 telle que modifiée ci-dessus à ±10 %. Noter que les paramètres PROBE_MIN et PROBE_MAX définis sur la même valeur illustrent la recommandation CPF 2 visant à utiliser un intervalle fixe entre les sondes plutôt que la recommandation de l'IETF RFC 5227, 2.1.1 (Détails des sondes) selon laquelle les sondes ultérieures après la sonde initiale sont rendues aléatoires entre les valeurs PROBE_MIN et PROBE_MAX.

##### 7.5.7.3.2 ProbeIpv4Address

Voir l'IETF RFC 5227, 2.1 et 2.1.1.

##### 7.5.7.3.3 AnnounceIpv4Address

Voir l'IETF RFC 5227, 2.3.

Une fois que l'appareil a envoyé sa première annonce ARP, il doit passer à l'état OngoingDetection. Si aucun conflit n'est détecté au bout de ANNOUNCE_INTERVAL secondes, l'appareil doit effectuer l'envoi de sa deuxième annonce ARP de manière asynchrone avec les autres opérations réseau.

##### 7.5.7.3.4 WaitLinkIntegrity

L'activité WaitLinkIntegrity attend l'occurrence d'un événement LinkUp depuis un port Ethernet.

Cette activité est initiée dans deux scénarios:

- premièrement lorsqu'un événement LinkDown se produit sur un appareil à port unique;
- deuxièmement, lorsqu'un événement LinkDown se produit sur un appareil à ports multiples et que tous ses autres ports Ethernet sont également inutilisables.

Lorsqu'un événement LinkUp se produit, l'activité ProbeIpv4Address est initiée.

Lorsqu'un appareil détecte que l'intégrité de la liaison Ethernet a été perdue puis retrouvée (par exemple, le câble réseau a été retiré puis remis), l'appareil doit redémarrer l'activité ProbeIpv4Address initiale.

#### 7.5.7.3.5 Notification et action de cas de défaut

Voir l'IETF RFC 5227, Article 1.

Outre le comportement de l'IETF RFC 5227, lorsqu'un conflit d'adresses est détecté, les appareils de Type 2 doivent prendre les actions de cas de défaut suivantes:

- définir l'état de l'appareil sur Recoverable Fault (défaut récupérable) (voyant DEL d'état du module rouge clignotant);
- définir le voyant DEL d'état du réseau sur rouge fixe.

Il existe 2 transitions de synchronisation entre l'activité ACD Notification & FaultAction et le diagramme de comportement de QuickConnect.

QcSynch-AcdFaultNotification est une transition de synchronisation vers le diagramme de comportement de QuickConnect qui se produit lorsque le mécanisme ACD contient un défaut ConflictDetected provenant de l'activité ProbeIpAddress ou un défaut DefenseFailed provenant de l'activité DefendWithPolicyB.

QcSynch-AcdFaultNotificationProcessed est une transition de synchronisation du diagramme de comportement QuickConnect vers l'activité Notification & FaultAction qui se produit après que QuickConnect a traité la précédente AcdFaultNotification.

#### 7.5.7.3.6 AcquireNewIpv4Parameters

Voir l'IETF RFC 5227, Article 1.

Après notification de la détection d'un conflit d'adresses IPv4, l'appareil peut être conçu pour obtenir ou utiliser une adresse IPv4 alternative. La conception de cette méthode et la sélection qui en résulte pour les paramètres IPv4 sont propres au fournisseur.

Si un événement LinkDown se produit sur le dernier port actif alors que l'activité est AcquireNewIpv4Parameter, le processus ACD illustré dans la Figure 23 (Comportement ACD) doit s'achever. Le processus ACD doit être redémarré après qu'au moins une liaison ait été rétablie et qu'une configuration IPv4 valide ait été acquise conformément à la Figure 21 et la Figure 22 qui illustrent le comportement de l'objet TCP/IP.

Noter également que les appareils CP 2/2 doivent limiter la fréquence à laquelle ils sondent de nouvelles adresses candidates en utilisant les paramètres MAX_CONFLICTS et RATE_LIMIT_INTERVAL comme défini dans l'IETF RFC 5227, 2.1.1.

#### 7.5.7.3.7 OngoingDetection

Voir l'IETF RFC 5227, 2.4.

QcSynch-StartAcdWithQc est une transition de synchronisation avec le diagramme de comportement de QuickConnect. Si QuickConnect est activé sur l'appareil, l'activité de l'ACD démarre à ce stade.

Le mécanisme ACD tel que spécifié dans l'IETF RFC 5227, 2.1 déclare qu'« un hôte ne doit pas effectuer cette vérification périodiquement », indiquant qu'il n'est pas nécessaire que les sondes ARP périodiques soient envoyées dans le cadre de la détection continue. Toutefois, il a été observé que les sondes ARP périodiques permettent au module de détecter des conflits avec des appareils qui peuvent ne pas avoir été connectés au réseau lors du sondage initial,

ou lorsque les commutateurs ont perdu les sondes ARP initiales en raison du délai de transmission initial. Par conséquent, il convient que les appareils de Type 2 qui fournissent l'algorithme ACD émettent des sondes ARP périodiques durant la détection continue.

Les appareils de Type 2 prenant en charge les sondes ARP périodiques doivent utiliser un délai de transmission dans la plage comprise entre ONGOING_PROBE_MIN et ONGOING_PROBE_MAX selon les valeurs ci-dessous:

- ONGOING_PROBE_MIN: 90 s (intervalle minimal pour les sondes continues);
- ONGOING_PROBE_MAX: 150 s (intervalle maximal pour les sondes continues).

Ces valeurs ne sont pas définies dans l'IETF RFC 5227. Les appareils ne doivent pas dépasser cette plage spécifiée de plus de 10 % aux deux extrémités.

Il est recommandé que le délai de transmission initial pour la première sonde ARP périodique ait une valeur pseudo-aléatoire dans la plage de ONGOING_PROBE_MIN et ONGOING_PROBE_MAX. (rendue "aléatoire" à l'aide de l'adresse MAC Ethernet comme cela est montré par exemple en 7.5.7.3.8.2). Le délai de transmission pour les sondes ARP périodiques ultérieures peut ne pas être rendu aléatoire et peut, par exemple, sélectionner la valeur centrale (120 s).

#### 7.5.7.3.8 DefendWithPolicyB

##### 7.5.7.3.8.1 Utilisation

Voir l'IETF RFC 5227, 2.4.

L'IETF RFC 5227, 2.4 déclare que « la détection de conflit d'adresses ne se limite pas uniquement au moment de la configuration d'interface initiale, lorsqu'un hôte envoie des sondes ARP. La détection de conflit d'adresses est un processus continu en vigueur tant qu'un hôte utilise une adresse. »

Après sondage réussi et utilisation d'une adresse IP, un appareil doit effectuer une détection de conflit et une défense continues, conformément à l'IETF RFC 5227.

En cas de détection d'un conflit d'adresses, l'appareil doit défendre son adresse conformément à l'alternative (b) indiquée dans l'IETF RFC 5227, 2.4.

Si un conflit perdure (comme défini par l'IETF RFC 5227), l'appareil doit suivre le même comportement que celui adopté lors de la détection d'un conflit pendant le sondage initial (état de l'appareil, DEL, comportement DHCP, etc.) et exécuter la notification et les actions de cas de défauts spécifiées.

##### 7.5.7.3.8.2 Exemple d'algorithme de délai pseudo-aléatoire (informative)

NOTE L'algorithme Xorshift Random Number Generator suivant est fondé sur le travail de George Marsaglia de la Florida State University.

Ce générateur de nombres aléatoires (RNG) est utilisé pour produire un délai initial pseudo-aléatoire avant d'envoyer la première sonde ARP périodique.

L'objectif est de produire un nombre aléatoire compris dans la plage de PROBE_WAIT (entre 180 et 200) en calculant un décalage aléatoire à partir du début de la plage.

PROBE_WAIT est calculé sur la base de la formule suivante:

$$\text{PROBE\_WAIT} = \text{PROBE\_WAIT\_MIN} + \Delta rn$$

∆rn représente le décalage aléatoire compris entre (0 .. et (PROBE_WAIT_MAX – PROBE_WAIT_MIN) et calculé à l'aide de la formule suivante:

∆rn = (PROBE_WAIT_MAX – PROBE_WAIT_MIN) * R256/256.

R256 est un nombre aléatoire compris dans la plage (0 .. 255) et calculé à l'aide de l'algorithme ci-dessous.

L'algorithme RNG utilise une adresse MAC IEEE 802.3, par exemple 12-34-56-78-9A-BC, ici écrite en notation canonique. La notation canonique représente l'ordre dans lequel les octets de l'adresse MAC sont transmis, de gauche à droite, sur le câble; c'est -à-dire 0x12 envoyé en premier, 0x34 envoyé ensuite, etc. L'adresse MAC est mappée avec une matrice, EnetAddr[], de 6 octets comme suit, à l'aide de l'adresse MAC d'exemple:

EnetAddr[0] = 0x12;

EnetAddr[1] = 0x34;

EnetAddr[2] = 0x56;

EnetAddr[3] = 0x78;

EnetAddr[4] = 0x9A;

EnetAddr[5] = 0xBC;

R256 est ensuite calculé à l'aide du RNG Xorshift suivant.

R256 = EnetAddr[0];

R256 = (R256 << 1 ) ^ EnetAddr[1];

R256 = (R256 << 1 ) ^ EnetAddr[2];

R256 = (R256 << 1 ) ^ EnetAddr[3];

R256 = (R256 << 1 ) ^ EnetAddr[4];

R256 = (R256 << 1 ) ^ EnetAddr[5];

R256 = R256 % 256;

Où << est l'opérateur de décalage gauche, ^ l'opérateur OR exclusif, et % est l'opérateur de base arithmétique.

#### 7.5.7.3.9 SemiActiveProbe

L'activité SemiActiveProbe est initiée lorsqu'un appareil à ports multiples reçoit un événement LinkUp pendant que les autres ports sont toujours actifs.

L'activité SemiActiveProbe consiste en une activité de sondage modifiée dans laquelle seules deux sondes ARP sont envoyées à l'aide de délais de sondage de 200 ms.

L'activité SemiActiveProbe est définie pour réduire le volume de trafic de diffusion ARP qui sera initié depuis l'activité LinkUp sur les appareils à ports multiples.

### 7.6 Objet de liaison Ethernet

#### 7.6.1 Présentation

L'objet de liaison Ethernet gère les compteurs spécifiques à la liaison et les informations d'état d'un port Ethernet physique ISO/CEI 8802-3. Chaque appareil doit prendre en charge exactement une instance de l'objet de liaison Ethernet pour chaque port Ethernet dans le module. Les appareils peuvent utiliser une instance d'objet de liaison Ethernet pour une interface accessible en interne, telle qu'un port interne pour un commutateur intégré.

### 7.6.2 Historique de révision

L'historique de révision de l'objet Ethernet Link est décrit dans le Tableau 62.

**Tableau 62 – Historique de révision de l'objet de liaison Ethernet**

| Révision de la classe | Description des modifications |
|---|---|
| **01** | Emission initiale |
| **02** | Emission pour la série CEI 61158. |
| **03** | Ajout des nouveaux attributs d'instance 7-10 offrant une prise en charge des appareils à ports Ethernet multiples |

### 7.6.3 Attributs de classe

L'objet de liaison Ethernet doit prendre en charge les attributs de classe spécifiés dans le Tableau 63.

**Tableau 63 – Attributs de classe de l'objet de liaison Ethernet**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x01** | Obligatoire si la valeur de révision est supérieure à 1 | Get | NV | Revision | UINT | Révision de cet objet | La valeur minimale doit être 1.<br>Doit être au moins 2 si l'attribut d'instance 6 est mis en œuvre.<br>Doit être 3 si l'un des attributs d'instance 7 à 10 est mis en œuvre.<br>La valeur maximale doit être 3. |
| **0x02** | Nécessaire si le nombre d'instances est supérieur à 1. | Get | NV | Max Instance | UINT | Nombre maximal d'instances d'un objet créées dans ce niveau de classe de l'appareil | Nombre d'instances le plus élevé d'un objet créé à ce niveau hiérarchique de classe |
| **0x03** | Nécessaire si le nombre d'instances est supérieur à 1. | Get | NV | Nombre d'instances | UINT | Nombre d'instances d'objets actuellement créées à ce niveau de classe dans l'appareil | Le nombre d'instances d'objet à ce niveau de hiérarchie de classe |
| **0x04 à 0x07** | Ces attributs de classe sont facultatifs et sont décrits dans la CEI 61158-5-2. | | | | | | |

Une erreur lors de la lecture de l'attribut de révision de classe implique qu'il s'agit d'une mise en œuvre de révision 1 uniquement.

### 7.6.4 Attributs d'instance

#### 7.6.4.1 Généralités

L'objet Ethernet Link doit prendre en charge les attributs d'instance tel que spécifié dans le Tableau 64.

**Tableau 64 – Attributs d'instance de l'objet de liaison Ethernet**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x01** | Obligatoire | Get | V | Vitesse d'interface | UDINT | Vitesse d'interface utilisée | Vitesse en Mbit/s (0, 10, 100, 1 000, par exemple). Voir 7.6.4.2 |
| **0x02** | Obligatoire | Get | V | Balises d'interface | DWORD | Balises d'état de l'interface | Matriciel de balises d'interface. Voir 7.6.4.3 |
| **0x03** | Obligatoire | Get | NV | Adresse physique | USINT[6] | Adresse de la couche MAC | Voir 7.6.4.4 |
| **0x04** | Sous condition[a] | Get | V | Compteurs d'interface | STRUCT de 44 octets | | Voir 7.6.4.5 |
| | | | | En octets | UDINT | Octets reçus sur l'interface | |
| | | | | In Ucast Packets | UDINT | Paquets à diffusion individuelle reçus sur l'interface | |
| | | | | In NUcast Packets | UDINT | Paquets à diffusion non individuelle reçus sur l'interface | |
| | | | | In Discards | UDINT | Paquets entrants reçus sur l'interface mais écartés | |
| | | | | In Errors | UDINT | Paquets entrants contenant des erreurs (n'inclut pas In Discards) | |
| | | | | In Unknown Protos | UDINT | Paquets entrants avec protocole inconnu | |
| | | | | Out Octets | UDINT | Octets envoyés sur l'interface | |
| | | | | Out Ucast Packets | UDINT | Paquets à diffusion individuelle envoyés sur l'interface | |
| | | | | Out NUcast Packets | UDINT | Paquets à diffusion non individuelle envoyés sur l'interface | |
| | | | | Out Discards | UDINT | Paquets sortants écartés | |
| | | | | Out Errors | UDINT | Paquets sortants contenant des erreurs | |
| **0x05** | Facultatif | Get | V | Compteurs de support | STRUCT de 48 octets | Compteurs spécifiques au support | Voir 7.6.4.6 |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| | | | | Erreurs d'alignement | UDINT | Trames reçues dont la longueur n'est pas un nombre d'octets intégral | |
| | | | | Erreurs FCS | UDINT | DLPDU reçues n'ayant pas satisfait au contrôle FCS | |
| | | | | Collisions uniques | UDINT | DLPDU transmises avec succès ayant fait l'objet d'une seule collision | |
| | | | | Collisions multiples | UDINT | DLPDU transmises avec succès ayant fait l'objet de plusieurs collisions | |
| | | | | Erreurs d'essai SQE | UDINT | Nombre de fois qu'un message d'erreur d'essai SQE a été généré | |
| | | | | Transmissions différées | UDINT | DLPDU dont la première tentative de transmission a été retardée car le support est occupé | |
| | | | | Collisions tardives | UDINT | Nombre de fois qu'une collision est détectée après 512 temps bits dans la transmission d'un paquet | |
| | | | | Collisions excessives | UDINT | DLPDU dont la transmission n'a pas abouti en raison de collisions excessives | |
| | | | | Erreurs de transmission MAC | UDINT | DLPDU dont la transmission n'aboutit pas n raison d'une erreur de transmission de la sous-couche MAC | |
| | | | | Erreurs de sens de porteuse | UDINT | Nombre de fois que le sens de la porteuse a été perdu ou n'a jamais été évaluée lors de la tentative de transmission d'une trame | |
| | | | | Trame trop longue | UDINT | DLPDU reçues ayant dépassé la taille DLPDU maximale admise | |
| | | | | Erreurs de réception MAC | UDINT | DLPDU dont la réception sur une interface n'a pas abouti en raison d'une erreur de réception de la sous-couche MAC interne | |
| **0x06** | Facultatif | Get/Set | V | Contrôle d'interface | STRUCT de | Configuration de l'interface physique | Voir 7.6.4.7 |
| | | | | Bits de contrôle | WORD | Bits de contrôle d'interface | |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| | | | | Vitesse d'interface forcée | UINT | Vitesse à laquelle l'interface doit être forcée pour fonctionner | Vitesse en Mbit/s (10, 100, 1 0 00, par exemple). |
| **0x07** | Facultatif | Get | NV | Type d'interface | USINT | Type d'interface (paire torsadée, fibre, interne, par exemple) | Voir 7.6.4.8 |
| **0x08** | Facultatif | Get | V | Etat d'interface | USINT | Etat actuel de l'interface (opérationnelle, désactivée, par exemple) | Voir 7.6.4.9 |
| **0x09** | Facultatif | Set | NV | Etat Admin | USINT | Etat administratif: activé, désactivé | Voir 7.6.4.10 |
| **0x0A** | Sous condition[b] | Get | NV | Libellé d'interface | SHORT_STRING_ | Identification lisible par l'homme | Voir 7.6.4.11 |

[a] L'attribut Compteurs d'interface est obligatoire si l'attribut Compteurs de support est mis en œuvre.

[b] Obligatoire si le nombre d'instances est supérieur à 1.

### 7.6.4.2 Vitesse d'interface

L'attribut Vitesse d'interface doit indiquer la vitesse de fonctionnement de l'interface (10 Mbit/s, 100 Mbit/s, 1 Gbit/s ou d'autres vitesses). Une valeur égale à 0 doit être utilisée pour indiquer que la vitesse de l'interface est indéterminée.

L'échelle de l'attribut étant exprimée en Mbit/s, l'interface fonctionne à 100 Mbit/s. Par conséquent, la valeur de l'attribut Vitesse d'interface doit être égale à 100. La vitesse de l'interface a pour objet de représenter la durée de transmission de liaison disponible. L'attribut ne doit pas être indiqué deux fois si l'interface fonctionne en mode bidirectionnel simultané.

#### 7.6.4.3 Balises d'interface

L'attribut Balises d'interface contient les informations d'état et de configuration relatives à l'interface physique (voir Tableau 65).

**Tableau 65 – Bits des balises d'interface**

| Bit | Nom | Définition |
|---|---|---|
| **0** | Etat de la liaison | Indique si le port est connecté ou pas à un réseau actif:<br>– 0 indique une liaison inactive<br>– 1 indique une liaison active<br>La détermination de l'état de la liaison est spécifique à la mise en œuvre. Dans certains cas, les appareils peuvent dire si la liaison est active par l'intermédiaire d'un support matériel/pilote. Dans d'autres cas, l'appareil peut uniquement être en mesure de dire si la liaison est active en présence de paquets entrants |
| **1** | Bidirectionnel non simultané/simultané | Indique le mode duplex utilisé:<br>– 0 indique que le port fonctionne en mode duplex non simultané<br>– 1 indique le mode duplex simultané<br>Si la balise d'état de la liaison est 0, la valeur de la balise bidirectionnel non simultané/simultané est indéterminée. |
| **2-4** | Etat de négociation | Indique l'état de l'auto-négociation de la liaison:<br>0 = Auto-négociation en cours<br>1 = Echec de l'auto-négociation et de la détection de la vitesse. Utilisation des valeurs par défaut pour la vitesse et le duplex. Les valeurs par défaut recommandées sont 10 Mbit/s et bidirectionnel non simultané<br>2 = L'auto-négociation n'a pas abouti mais la vitesse a été détectée. Duplex prend la valeur par défaut. La valeur par défaut dépend du produit. La valeur recommandée est bidirectionnel non simultané<br>3 = Vitesse et duplex négociés<br>4 = Auto-négociation non tentée. Vitesse et duplex forcés. |
| **5** | Définition manuelle requiert réinitialisation | 0 indique que l'interface peut activer automatiquement les modifications apportées aux paramètres de liaison (auto-négociation, mode duplex, vitesse d'interface)<br>1 indique que l'appareil implique d'émettre un service Reset pour son objet d'identification afin que les modifications entrent en vigueur |
| **6** | Défaut matériel local | 0 indique que l'interface ne détecte aucun défaut matériel<br>1 indique qu'un défaut matériel local est détecté<br><br>Sa signification est spécifique au produit. Par exemple: une interface AUI/MII ne détecte aucun émetteur-récepteur associé ou le modem cellulaire ne détecte aucune antenne associée. A l'inverse, la nature auto-correctrice de l'état de la liaison étant inactive, elle est considérée comme un incident machine impliquant l'intervention de l'utilisateur |
| **7-31** | Réservé | Réservé et une valeur nulle doit être attribuée |

#### 7.6.4.4 Adresse physique

L'attribut Adresse physique contient l'adresse de la couche MAC de l'interface. L'adresse physique est une matrice d'octets. Le format d'affichage recommandé est « XX-XX-XX-XX-XX-XX », en commençant par le premier octet. L'adresse physique n'est pas un attribut définissable: l'adresse Ethernet doit être attribuée par le fabricant et doit être unique conformément aux exigences de l'ISO/CEI 8802-3.

Les appareils avec des ports multiples mais une seule interface MAC (par exemple un appareil avec une technologie de commutateur intégrée) peuvent utiliser la même valeur pour cet attribut dans chaque instance de l'objet de liaison Ethernet. L'exigence principale est que la valeur de cet attribut doit être l'adresse MAC utilisée pour les paquets à destination et en provenance de l'interface MAC de l'appareil sur ce port physique.

#### 7.6.4.5 Compteurs d'interface

L'attribut Compteurs d'interface contient tous les compteurs ayant rapport à la réception des paquets sur l'interface. Ces compteurs doivent être définis conformément à l'IETF RFC 1213. L'attribut Compteurs d'interface est conditionnel. Il doit être mis en œuvre si l'attribut Compteurs de support l'est également.

Les appareils à ports multiples avec unique interface MAC (par exemple, appareil avec commutateur intégré) doivent maintenir les valeurs des compteurs de trois façon possibles.

- Dans chaque instance, compter les trames MAC envoyées/reçues par l'appareil lui-même sur le port représenté par cette instance (c'est-à-dire chaque interface physique compte les trames MAC envoyées/reçues sur cette interface). Cette solution est privilégiée.
- Utiliser les valeurs compteurs égales à 0 pour ces instances qui correspondent aux ports de commutateurs externes; compter les trames MAC dans l'instance qui correspond à l'interface de l'appareil interne.
- Utiliser les mêmes valeurs de compteurs pour toutes les instances, en comptant les trames MAC envoyées/reçues par l'appareil lui-même.

#### 7.6.4.6 Compteurs de support

L'attribut Compteurs de support contient les compteurs spécifiques au support Ethernet. Ces compteurs doivent être définis conformément à l'IETF RFC 1643. Si cet attribut est mis en œuvre, l'attribut Compteurs d'interface doit l'être également. Les instances qui font référence aux interfaces internes peuvent définir les valeurs des Compteurs d'interface sur 0.

Certaines mises en œuvre matérielles ou logicielles sous-jacentes peuvent ne pas proposer tous les compteurs de support. Dans le cas d'un support par fibre, certains compteurs ne s'appliquent pas (par exemple, compteurs de collision). Les appareils doivent utiliser des valeurs égales à 0 pour les compteurs non mis en œuvre.

#### 7.6.4.7 Contrôle d'interface

##### 7.6.4.7.1 Présentation

L'attribut Contrôle d'interface est une structure composée de bits de contrôle et d'une vitesse d'interface forcée, et doit être conforme aux spécifications des Paragraphes 7.6.4.7.2 à 7.6.4.7.3.

##### 7.6.4.7.2 Bits de contrôle

L'attribut Bits de contrôle est spécifié dans le Tableau 66.

**Tableau 66 – Bits de contrôle**

| Bit | Nom | Définition |
|---|---|---|
| **0** | Auto-négociation | 0 indique que l'auto-négociation de liaison ISO/CEI 8802-3 est désactivée<br>1 indique que l'auto-négociation est activée<br><br>Si l'auto-négociation est désactivée, l'appareil doit utiliser les paramètres indiqués par le mode duplex forcé et les bits de vitesse d'interface forcée |

| Bit | Nom | Définition |
|---|---|---|
| **1** | Mode duplex forcé | Si la valeur 0 est attribuée au bit d'auto-négociation, le bit de mode duplex forcé indique si l'interface doit fonctionner en mode duplex simultané ou non simultané.<br>0 indique qu'il convient que l'interface fonctionne en mode duplex non simultané<br>1 indique qu'il convient que l'interface fonctionne en mode duplex simultané<br>Les interfaces qui ne prennent pas en charge le mode duplex demandé doivent retourner un code d'erreur 0x09 (valeur d'attribut non valide). Si l'auto-négociation est activée, la tentative de définition du mode duplex forcé doit donner un code d'erreur 0x0C (conflit d'état de l'objet). |
| **2-15** | Réservé | Doit être défini à zéro |

#### 7.6.4.7.3 Vitesse d'interface forcée

Si la valeur 0 est attribuée au bit d'auto-négociation, les bits de vitesse d'interface forcée indiquent la vitesse à laquelle l'interface doit fonctionner. La vitesse est exprimée en Mbit/s (par exemple, pour 10 Mbit/s Ethernet, la vitesse d'interface doit être 10). Il convient que les interfaces qui ne prennent pas en charge la vitesse demandée retournent un code d'erreur 0x09 (valeur d'attribut non valide).

Si l'auto-négociation est activée, la tentative de définition de la vitesse d'interface forcée doit donner un code d'erreur 0x0C (conflit d'état de l'objet).

### 7.6.4.8 Type d'interface

L'attribut Type d'interface indique le type d'interface physique. Le Tableau 67 spécifie les valeurs de type d'interface. Cet attribut doit être stocké dans la mémoire non volatile.

**Tableau 67 – Type d'interface**

| Valeur | Type d'interface |
|---|---|
| 0 | Type d'interface inconnu |
| 1 | Il s'agit d'une interface interne à l'appareil (dans le cas d'un commutateur intégré, par exemple) |
| 2 | Paire torsadée (10Base-T, 100Base-TX, 1000Base-T, par exemple) |
| 3 | Fibre optique (100Base-FX, par exemple) |
| 4-255 | Réservé |

### 7.6.4.9 Etat d'interface

L'attribut Etat d'interface indique l'état opérationnel en cours de l'interface. Le Tableau 68 spécifie les valeurs d'état d'interface. Cet attribut doit être stocké dans la mémoire volatile.

**Tableau 68 – Etat d'interface**

| Valeur | Etat d'interface |
|---|---|
| 0 | Etat d'interface inconnu |
| 1 | L'interface est activée et prête à envoyer et recevoir des données |
| 2 | L'interface est désactivée |
| 3 | L'interface est en cours d'essai |
| 4-255 | Réservé |

#### 7.6.4.10 Etat Admin

L'attribut Etat Admin permet de définir les paramètres administratifs de l'état d'interface. Le Tableau 69 spécifie les valeurs d'Etat Admin. Cet attribut doit être stocké dans la mémoire non volatile.

**Tableau 69 – Etat Admin**

| Valeur | Etat Admin |
|---|---|
| 0 | Réservé |
| 1 | Activer l'interface |
| 2 | Désactiver l'interface. S'il s'agit de la seule interface de communication CPF2, une demande de désactivation de l'interface doit donner une erreur (code d'état 0x09) |
| 3-255 | Réservé |

Les appareils dont seul le port de communication est un port CP 2/2 avec une seule instance Ethernet Link doivent retourner le code d'état général 0x09 (valeur d'attribut non valide) si une demande de désactivation de son interface est reçue. Les appareils dotés de plusieurs ports (n'importe quelle combinaison d'instances Ethernet Link multiples et/ou d'autres ports de communication) doivent retourner le code d'état général 0x10 (conflit d'états d'appareils) si l'exécution d'une demande de désactivation d'une interface entraînait la désactivation de l'ensemble des ports de communication de l'appareil.

#### 7.6.4.11 Libellé d'interface

L'attribut Interface label est une chaîne de texte décrivant l'interface. Le contenu de la chaîne est spécifique au fournisseur. Pour les interfaces internes, il convient que la chaîne de texte contienne le terme « interne ». Cette chaîne peut contenir au maximum 64 caractères. Cet attribut doit être stocké dans la mémoire non volatile.

### 7.6.5 Services communs

#### 7.6.5.1 Généralités

L'objet de liaison Ethernet doit prendre en charge les services communs spécifiés dans le Tableau 70.

**Tableau 70 – Services communs de l'objet de liaison Ethernet**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| **0x01** | Facultatif | Facultatif | Get_Attribute_All | Renvoie une liste prédéfinie des attributs de cet objet (voir la définition de réponse Get_Attribute_All en 7.6.5.2) |
| **0x0E** | Sous condition | Obligatoire | Get_Attribute_Single | Renvoie le contenu de l'attribut spécifié |
| **0x10** | N/A | Sous condition | Set_Attribute_Single | Modifie un seul attribut |

Le service Get_Attribute_Single doit être mis en œuvre pour l'attribut de classe si ce dernier est mis en œuvre.

Le service Set_Attribute_Single doit être mis en œuvre si l'attribut de contrôle d'interface est mis en œuvre.

#### 7.6.5.2 Réponse Get_Attribute_All

Au niveau de classe, la réponse Get_Attributes_All doit contenir les attributs de classe dans l'ordre numérique, jusqu'au dernier attribut mis en œuvre. Tous les attributs non mis en œuvre dans la réponse doivent utiliser les valeurs d'attribut par défaut.

Pour les attributs d'instance, les attributs doivent être renvoyés dans l'ordre numérique, jusqu'au dernier attribut mis en œuvre. La réponse Get_Attribute_All doit être telle que spécifiée dans le Tableau 71.

**Tableau 71 – Format de réponse Get_Attribute_All**

| ID attribut | Taille (octets) | Sommaire |
|---|---|---|
| **1** | 4 | Vitesse d'interface |
| **2** | 4 | Balises d'interface |
| **3** | 6 | Adresse physique |
| **4** | 44 | Compteurs d'interface<br>La valeur par défaut est égale à 0 s'il n'y a pas de mise en œuvre |
| **5** | 48 | Compteurs de support<br>La valeur par défaut est égale à 0 s'il n'y a pas de mise en œuvre |
| **6** | 4 | Contrôle d'interface<br>La valeur par défaut est égale à 0 s'il n'y a pas de mise en œuvre<br>Une valeur de 0 (qui indiquerait littéralement auto-négociation désactivée, duplex non simultané et vitesse de 0) doit indiquer que l'attribut n'est pas mis en œuvre. |
| **7** | 1 | Type d'interface<br>La valeur par défaut est égale à 0 s'il n'y a pas de mise en œuvre |
| **8** | 1 | Etat d'interface<br>La valeur par défaut est égale à 0 s'il n'y a pas de mise en œuvre |
| **9** | 1 | Etat Admin<br>La valeur par défaut est égale à 0 s'il n'y a pas de mise en œuvre |
| **10** | 1 | Longueur du libellé d'interface |
| | Variable, égale à la longueur du libellé de l'interface | Libellé d'interface |

La longueur du libellé d'interface est inconnue avant l'émission de la demande de service Get_Attribute_All. Les implémenteurs doivent se préparer à accepter une réponse contenant la taille maximale du libellé d'interface (65 USINT).

### 7.6.6 Services spécifiques à la classe

#### 7.6.6.1 Généralités

L'objet de liaison Ethernet doit prendre en charge les services spécifiques à la classe spécifiés dans le Tableau 72.

**Tableau 72 – Services spécifiques à la classe de l'objet de liaison Ethernet**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| **0x4C** | N/A | Sous condition | Get_and_Clear | Permet d'obtenir, puis de supprimer, l'attribut spécifié (Compteurs d'interface ou Compteurs de support) |

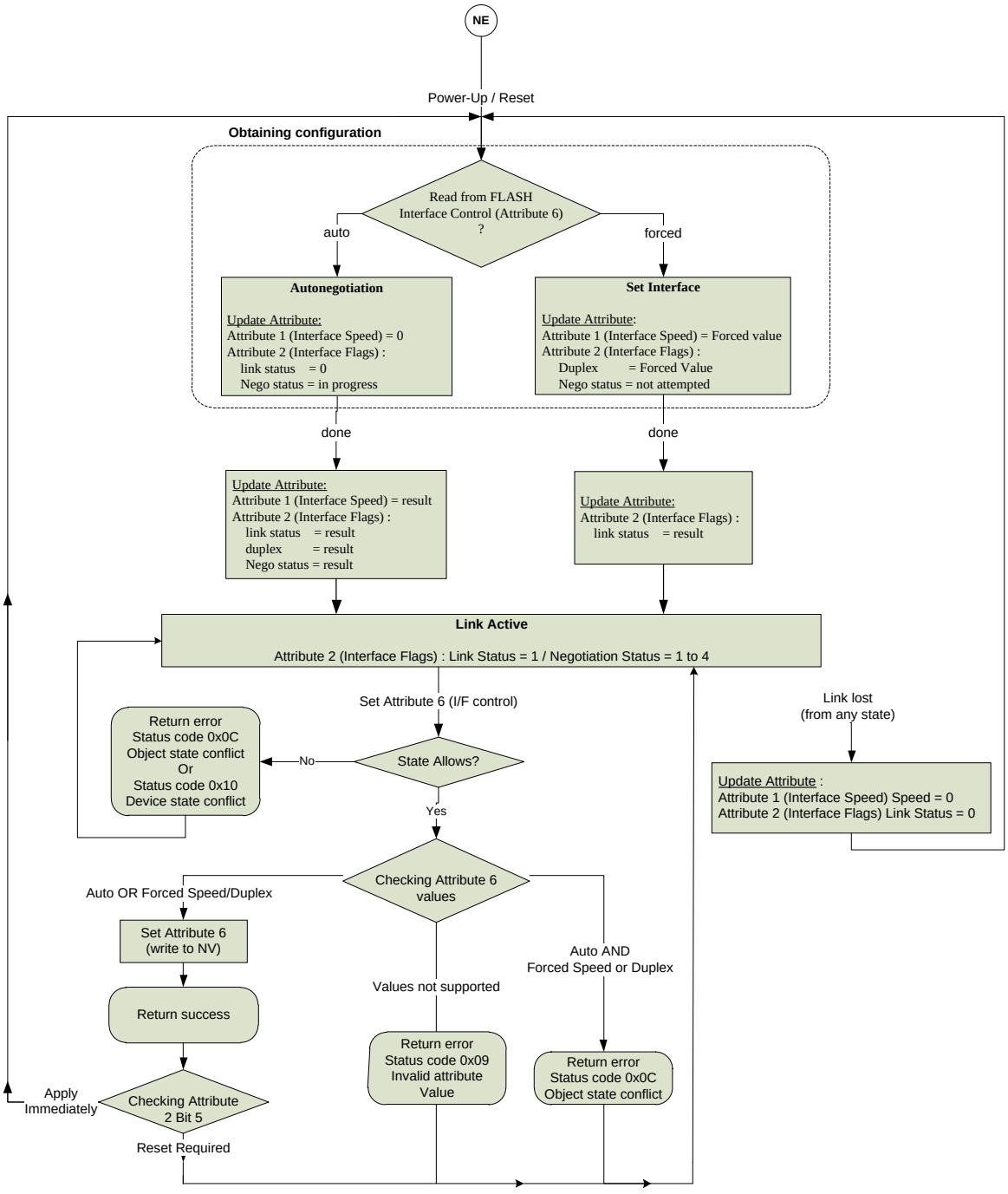
Le service Get_and_Clear doit uniquement être mis en œuvre si les attributs Compteurs d'interface et Compteurs de support le sont.

#### 7.6.6.2 Service Get_and_Clear

Le service Get_and_Clear est un service spécifique à la classe. Il est uniquement pris en charge pour les attributs Compteurs d'interface et Compteurs de support. La réponse Get_and_Clear doit être identique à la réponse Get_Attribute_Single de l'attribut spécifié. Suite à la génération de la réponse, la valeur de l'attribut doit être nulle.

### 7.6.7 Comportement

Le comportement de l'objet d'interface TCP/IP doit être tel qu'illustré dans le diagramme de transition d'états (voir Figure 24).




**Légende**

| Anglais | Français |
|---|---|
| Obtaining configuration | Obtention de la configuration |
| Read from FLASH Interface Control (Attribute 6)? | Lecture du contrôle d'interface FLASH (attribut 6)? |
| Forced | Forcé |
| Autonegociation<br>Update attribute<br>Attribute1 (interface speed) = 0<br>Attribute 2 (interface flags):<br>Link status = 0 | Auto-négociation<br>Mise à jour de l'attribut<br>Attribut 1 (vitesse d'interface) = 0<br>Attribut 2 (balises d'interface):<br>Etat de la liaison = 0 |

| Anglais | Français |
|---|---|
| Nego status = in progress | Etat de la négociation = en cours |
| Set Interface<br>Update attribute:<br>Attribute 1 (interface speed) = forced value<br>Attribute 2 (interface flags):<br>Duplex = forced value<br>Nego status = not attempted | Définition de l'interface<br>Mise à jour de l'attribut<br>Attribut 1 (vitesse d'interface) = valeur forcée<br>Attribut 2 (balises d'interface):<br>Duplex = forced value<br>Etat de la négociation = pas de tentative |
| Done | Effectué |
| Update attribute:<br>Attribute 1 (interface speed) = result<br>Attribute 2 (interface flags):<br>Link status = result<br>Duplex = result<br>Nego status = result | Mise à jour de l'attribut:<br>Attribut 1 (vitesse d'interface) = résultat<br>Attribut 2 (balises d'interface):<br>Etat de la liaison = résultat<br>Duplex = résultat<br>Etat de la négociation = résultat |
| Update attribute:<br>Attribute 2 (interface flags):<br>Link status = result | Mise à jour de l'attribut:<br>Attribut 2 (balises d'interface):<br>Etat de la liaison = résultat |
| Link active<br>Attribute 2 (interface flags): link status = 1 / Negociation status = 1 to 4 | Liaison active<br>Attribut 2 (balises d'interface): état de la liaison = 1 / Etat de la négociation = 1 à 4 |
| Return error<br>Status code 0x0C<br>Object state conflict or<br>Status code 0x10<br>Device state conflict | Erreur de retour<br>Code statut 0x0C<br>Conflit d'état d'objet Ou<br>Code statut 0x10<br>Conflit d'état d'appareil |
| Yes / No | Oui / Non |
| Set attribute 6 (I/F control) | Définition de l'attribut 6 (contrôle I/F) |
| Link lost (from any state) | Liaison perdue (tout état) |
| Checking attribute 6 values | Vérification des 6 valeurs de l'attribut |
| Auto OR forced speed/duplex | Vitesse/duplex auto OU forcés |
| Set Attribute 6 (write to NV) | Définition de l'attribut 6 (écriture vers NV) |
| Return success | Réussite du retour |
| Apply immediately | Application immédiate |
| Checking attribute 2 bit 5 | Vérification du bit 5 de l'attribut 2 |
| Reset required | Réinitialisation requise |
| Values not supported | Valeurs non prises en charge |
| Auto AND forced speed/duplex | Vitesse/duplex auto ET forcés |
| Return error<br>Status code 0x09<br>Invalid attribute value | Erreur de retour<br>Code statut 0x09<br>Valeur d'attribut non valide |
| Return error<br>Status code 0x0C<br>Object state conflict | Erreur de retour<br>Code statut 0x0C<br>Conflit d'état d'objet |

**Figure 24 – Diagramme de transition d'états de l'objet de liaison Ethernet**

## 7.7 Objet DeviceNet

### 7.7.1 Présentation

L'objet DeviceNet doit fournir une interface cohérente de gestion de station aux couches de liaison physique et de données. Cet objet doit mettre les informations de diagnostic de ces couches à la disposition des applications client. Chaque nœud doit prendre en charge un objet DeviceNet par liaison.

Un appareil peut contenir plusieurs nœuds.

### 7.7.2 Historique de révision

L'historique de révision de l'objet DeviceNet est décrit dans le Tableau 73.

**Tableau 73 – Historique de révision de l'objet DeviceNet**

| Révision de la classe | Description des modifications |
|---|---|
| **01** | Emission initiale |
| **02** | Emission pour la série CEI 61158 |

### 7.7.3 Attributs de classe

L'objet DeviceNet doit prendre en charge les attributs de classe tel que spécifié dans le Tableau 74.

**Tableau 74 – Attributs de classe de l'objet DeviceNet**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x01** | Obligatoire | Get | NV | Revision | UINT | révision de cet objet | 2 (révision 2) |

### 7.7.4 Attributs d'instance

#### 7.7.4.1 Généralités

L'objet DeviceNet doit prendre en charge les attributs d'instance spécifiés dans le Tableau 75.

**Tableau 75 – Attributs d'instance de l'objet DeviceNet**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x01** | Facultatif | Get/Set | NV | MAC ID | USINT | adresse de nœud | Voir 7.7.4.2 |
| **0x02** | Facultatif | Get/Set | NV | Vitesse en bits | USINT | vitesse en bits | Voir 7.7.4.3 |
| **0x03** | Facultatif | Get/Set | NV | BOI | BOOL | Interruption de désactivation de bus | Voir 7.7.4.4 |
| **0x04** | Facultatif | Get/Set | V | Bus-Off Counter | USINT | Nombre de fois que CAN passe à l'état de désactivation de bus | Voir 7.7.4.5 |
| **0x05** | Facultatif | Get | V | Allocation Information | STRUCT of 2 octets | | Voir 7.7.4.6 |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| | | | | Allocation choice | SWORD | | Voir 7.7.4.6.2 |
| | | | | Master's MAC ID | USINT | MAC ID du maître (de l'allocation) | Voir 7.7.4.6.3 |
| **0x06** | Sous condition[a] | Get | V | MAC ID switch changed | BOOL | Le/Les commutateur(s) de l'adresse de nœud a/ont changé depuis la dernière mise sous tension/réinitialisation | 0 = pas de modification<br>1 = modification depuis la dernière réinitialisation |
| **0x07** | Sous condition[b] | Get | V | Bit rate switch changed | BOOL | Le/Les commutateur(s) de vitesse en bits a/ont changé depuis la dernière mise sous tension/réinitialisation | 0 = pas de modification<br>1 = modification depuis la dernière réinitialisation |
| **0x08** | Sous condition[a] | Get | V | MAC ID switch value | USINT | Valeur réelle du/des commutateur(s) d'adresse de nœud | 0 à 99 |
| **0x09** | Sous condition[b] | Get | V | Bit rate switch value | USINT | Valeur réelle du/des commutateur(s) de vitesse en bits | 0 à 9 |
| **0x0A** | Facultatif[c] | Get/Set | NV | QuickConnect | BOOL | Activation/Désactivation de la fonction de connexion rapide | 0 = désactivation (valeur par défaut)<br>1 = activation |
| **0x0B** | Sous condition[d] | | | Numéro de réseau de sécurité | SWORD[6] | | Voir la CEI 61784-3-2 |
| **0x0C** | Facultatif | Get | V | Compteurs de diagnostic | STRUCT of 34 octets | Liste des compteurs de diagnostic ISO 11898 | Voir 7.7.4.7 |
| | | | | Descripteur de compteurs de diagnostic | WORD | Indique les compteurs de diagnostic pris en charge | |
| | | | | Arbitration loss count | UINT | Voir ISO 11898 | Plage = 0 à 65 535<br>Valeur par défaut = 0 |
| | | | | Overload count | UINT | Voir ISO 11898 | |
| | | | | Bit error count | UINT | Voir ISO 11898 | |
| | | | | Stuff error count | UINT | Voir ISO 11898 | |
| | | | | Ack error count | UINT | Voir ISO 11898 | |
| | | | | Form error count | UINT | Voir ISO 11898 | |
| | | | | CRC error count | UINT | Voir ISO 11898 | |
| | | | | Rx message loss count | UINT | Le sous-système ISO 11898 a détecté un message reçu perdu | |
| | | | | Warning error count | UINT | Voir ISO 11898 | |
| | | | | Rx error counter | UINT | Recevoir compteur d'erreurs (voir ISO 11898) | Plage = 0 à 256 |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| | | | | Tx error counter | UINT | Transmettre compteur d'erreurs (voir ISO 11898) | Plage = 0 à 256 |
| | | | | | UINT[5] | Réservé | Valeur par défaut = 0 |
| **0x0D** | Sous condition[e] | Get | V | Active Node Table | BOOL[64] | Identifie les nœuds en ligne sur le réseau local, en fonction de l'adresse de nœud | Un bit pour chaque numéro de nœud. Le numéro de nœud est l'indice dans la matrice de bits.<br>0 = Inactif<br>1 = Actif |

[a] Ces attributs sont obligatoires si une valeur différente de l'appareil actuellement en ligne peut être attribuée au commutateur MAC ID.

[b] Ces attributs sont obligatoires si une valeur différente de l'appareil actuellement en ligne peut être attribuée au commutateur de vitesse en bits.

[c] Cet attribut doit être stocké dans la mémoire non volatile.

[d] Cet attribut est obligatoire pour les appareils de sécurité CPF 2. Les appareils non relatifs à la sécurité ne doivent pas mettre en œuvre cet attribut.

[e] Cet attribut est obligatoire si l'appareil prend en charge l'acheminement entre au moins deux ports CPF2.

#### 7.7.4.2 MAC ID

Cet attribut contient le MAC ID de cet appareil. La plage de valeurs est comprise entre 0 et 63 décimales.

Un appareil qui utilise un commutateur pour définir le MAC ID doit renvoyer une réponse d'erreur, avec le code d'erreur général 0x0E (l'attribut ne peut pas être défini), en réponse à une demande Set_Attribute_Single lorsque le paramètre de commutation est compris entre 0 et 63 et que le commutateur MAC ID est activé par d'autres paramètres. Si le paramètre de commutation MAC ID est supérieur à 63 (désactivé) ou si le commutateur MAC ID est désactivé par d'autres paramètres, l'appareil peut prendre en charge le paramètre de l'attribut MAC ID par la demande de service Set_Attribute_Single. L'appareil doit fournir une indication visuelle (commutateur physique, DEL, par exemple) pour signaler que le commutateur MAC ID a été remplacé. Une panne récupérable mineure doit être déclarée lorsque le commutateur MAC ID est activé et indique un MAC ID valide, mais ne correspondant pas à l'adresse en ligne en cours de l'appareil. Pendant la mise sous tension ou la réinitialisation, il convient qu'un matériel passe à l'état de défaut de communication si un paramètre non valide est attribué au commutateur MAC ID et si le paramètre de l'attribut MAC ID n'est pas pris en charge.

L'attribut MAC ID est considéré comme étant non volatil lorsque, une fois configuré, l'attribut doit être rappelé après un cycle de puissance ou la réinitialisation de l'appareil. La valeur MAC ID 63 doit être attribuée par défaut à la configuration « prête à l'emploi ». Les scénarii suivants de détermination MAC ID s'appliquent uniquement aux appareils qui utilisent la mémoire non volatile pour stocker le MAC ID de l'appareil.

1. Lorsque le commutateur MAC ID est valide et activé, sa valeur est chargée dans la mémoire non volatile lors de la mise sous tension ou de la réinitialisation de l'appareil et avant toute tentative de passage en ligne. La valeur du commutateur MAC ID n'est à aucun autre moment chargée dans la mémoire non volatile.

2. Les appareils doivent tenter de passer en ligne avec la valeur MAC ID stockée dans la mémoire non volatile ou, en cas de configuration « prête à l'emploi », avec une valeur MAC ID de 63.
3. Si un appareil dont le commutateur MAC ID est désactivé reçoit une demande Set_Attribute_Single avec un MAC ID valide, le MAC ID non volatil doit être chargé avec le nouveau MAC ID.
4. Si un appareil dont le commutateur MAC ID est activé reçoit une demande Set_Attribute_Single avec un MAC ID valide, le MAC ID non volatil ne doit pas changer et une réponse d'erreur portant le code d'erreur général 0x0E (l'attribut ne peut pas être défini) doit être renvoyée.

La modification de MAC ID demande à l'appareil de supprimer tous les objets de connexion et d'exécuter de nouveau le diagramme d'états d'accès à la liaison défini dans la CEI 62026-3:2008, 5.4.

#### 7.7.4.3 Vitesse en bits

L'attribut Bit Rate indique la vitesse en bits sélectionnée. Les valeurs sont définies dans le Tableau 76.

**Tableau 76 – Valeurs de l'attribut Bit rate**

| Valeur | Signification |
|---|---|
| 00 | 125 Kbit/s |
| 01 | 250 Kbit/s |
| 02 | 500 Kbit/s |

Un appareil qui utilise un commutateur pour définir la vitesse en bits doit renvoyer une réponse d'erreur, avec le code d'erreur général 0x0E (l'attribut ne peut pas être défini), en réponse à une demande Set_Attribute_Single lorsqu'une valeur valide est attribuée au commutateur de vitesse en bits, lequel n'est pas désactivé par d'autres paramètres. Si une valeur valide n'est pas attribuée au commutateur de vitesse en bits (désactivé) et/ou que le commutateur de vitesse en bits est désactivé par d'autres paramètres, l'appareil peut prendre en charge le paramètre de l'attribut Bit Rate par la demande de service Set_Attribute_Single. L'appareil doit fournir une indication visuelle (commutateur physique, DEL, par exemple) pour signaler que le commutateur de vitesse en bits a été remplacé. Une panne récupérable mineure doit être déclarée lorsque le commutateur de vitesse en bits est activé et indique une vitesse en bits valide, mais ne correspondant pas à la vitesse en bits en ligne en cours de l'appareil. Pendant la mise sous tension ou la réinitialisation, il convient qu'un appareil passe à l'état de défaut de communication si un paramètre non valide est attribué au commutateur de vitesse en bits et que le paramètre de l'attribut Bit Rate n'est pas pris en charge.

L'attribut Bit Rate est considéré comme étant non volatil lorsque, une fois configuré, l'attribut doit être rappelé après un cycle de puissance ou la réinitialisation de l'appareil. La configuration « prête à l'emploi » doit être celle par défaut, de sorte que l'appareil soit en mesure de passer en ligne à 125 kbit/s. Les scénarii suivant de détermination de la vitesse en bits s'appliquent uniquement aux appareils utilisant une mémoire non volatile pour stocker la vitesse en bits de l'appareil.

1. Lorsque le commutateur de vitesse en bits est valide et activé, sa valeur est chargée dans la mémoire non volatile lors de la mise sous tension ou de la réinitialisation de l'appareil, et avant toute tentative de passage en ligne. La valeur du commutateur de vitesse en bits n'est à aucun autre moment chargée dans la mémoire non volatile.
2. Les appareils doivent tenter de passer en ligne avec la valeur de vitesse en bits stockée dans la mémoire non volatile ou, en cas de configuration « prête à l'emploi », être en mesure de passer en ligne à 125 kbit/s.

3. Si un appareil dont le commutateur de vitesse en bits est désactivé reçoit une demande Set_Attribute_Single avec une vitesse en bits valide, l'attribut Bit Rate non volatil doit être chargé avec la nouvelle vitesse en bits.
4. Si un appareil dont le commutateur de vitesse en bits est activé reçoit une demande Set_Attribute_Single avec une vitesse en bits valide, l'attribut Bit Rate non volatil ne doit pas changer et une réponse d'erreur portant le code d'erreur général 0x0E (l'attribut ne peut pas être défini) doit être renvoyée.

La modification de l'attribut Bit Rate ne prendra pas effet tant que l'appareil n'est pas physiquement réinitialisé (un cycle de puissance ou un commutateur de réinitialisation, par exemple) ou réinitialisé le service Reset avec l'objet d'identité. Pendant cette période, la valeur de l'attribut Bit Rate ne correspondra pas à la vitesse en bits réelle du réseau.

#### 7.7.4.4 BOI (Bus-off interrupt)

L'attribut BOI est composé d'un bit qui définit la manière dont un appareil CAN traite les interruptions de désactivation de bus. L'attribut BOI se trouve en position binaire 0 dans un octet pour les services Get_Attribute_Single/Set_Attribute_Single. Le reste des bits de l'octet doit être nul.

**Tableau 77 – Valeurs de l'attribut BOI**

| Valeur | Signification |
|---|---|
| 00 | Maintient la puce CAN à son état de désactivation de bus (réinitialisation) en cas de détection d'une indication de désactivation de bus |
| 01 | Si possible, réinitialiser totalement la puce CAN et poursuivre la communication en cas de détection d'une indication de désactivation de bus |

Si la valeur FALSE est affectée à l'attribut BOI (valeur nulle) et en cas de détection d'un événement de désactivation de bus de la puce CAN ISO 11898, suivre la procédure ci-dessous:

- la puce CAN est maintenue à l'état de réinitialisation/désactivation de bus;
- l'appareil passe à l'état de défaut de communication (voir CEI 62026-3:2008, 5.4).

Si la valeur TRUE est affectée à l'attribut BOI (définie sur un) et qu'un événement de désactivation de bus d'une puce CAN a été détecté, il peut être possible de renvoyer la puce CAN à son mode de fonctionnement normal et de poursuivre la communication en fonction du diagramme d'états d'accès à la liaison défini dans la CEI 62026-3:2008, 5.4.

Si la valeur un est affectée à l'attribut BOI, l'appareil doit s'assurer qu'il ne se réinitialise pas indéfiniment et ne continue pas à produire des paquets corrompus sur le bus. Si ce n'est pas le cas, l'appareil interrompra toutes les communications sur le bus.

Les connexions ne sont pas nécessairement affectées lorsqu'un événement de désactivation de bus est détecté, et l'appareil peut poursuivre la communication. Les connexions préalablement établies peuvent être conservées ou supprimées (réinitialisation à chaud). Quoi qu'il en soit, l'algorithme de détection de MAC ID en double doit être de nouveau exécuté en fonction du diagramme d'états d'accès à la liaison.

Comme indiqué dans le Tableau 75, la prise en charge de l'attribut BOI est facultative. S'il n'est pas pris en charge, un appareil doit mettre en œuvre le comportement indiqué par la valeur d'attribut zéro (0) du Tableau 77.

#### 7.7.4.5 Bus-off counter

L'attribut Bus-off Counter permet de compter le nombre de fois que la puce CAN est passé à l'état de déconnexion du bus (nombre d'interruptions de désactivation de bus). Le compteur comporte des valeurs comprises entre 0 et 255 décimales.

L'attribut Bus-off Counter est remis à zéro à la mise sous tension ou à l'initialisation de l'appareil.

L'attribut Bus-off Counter arrête le comptage lorsqu'il atteint le nombre maximal. Le compteur ne passe pas à zéro. Le compteur reste au nombre maximal tant qu'un service Set_Attribute_Single n'est pas exécuté.

L'objet DeviceNet remet l'attribut Bus-off Counter à zéro (0) à chaque fois qu'il reçoit une demande Set_Attribute_Single spécifiant l'attribut Bus-Off Counter. Les données Set_Attribute_Single ne sont pas utilisées et peuvent prendre n'importe quelle valeur. La transmission d'une demande Set_Attribute_Single à l'attribut Bus-off Counter contient tous les éléments permettant de réinitialiser le compteur.

#### 7.7.4.6 Allocation Information

##### 7.7.4.6.1 Présentation

L'attribut Allocation Information est destiné au jeu de connexions maître/esclave prédéfini. Son aide est requise si le jeu de connexions maître/esclave prédéfini est pris en charge. Il indique si le jeu défini dans la CEI 62026-3:2008, 5.5 a été alloué. S'il l'a été, cet attribut indique l'appareil qui a procédé à l'allocation et la/les connexion(s) actuellement allouées.

Cet attribut est modifié lorsqu'une réponse qui a abouti associée à un service Allocate_Master/Slave_Connection_Set (défini ultérieurement en 7.7.6) est générée. Cet attribut ne peut pas être modifié par le service Set_Attribute_Single. Une réponse d'erreur dont le champ de code d'erreur général est défini sur 0x0E (l'attribut ne peut pas être défini) est renvoyée si une demande Set_Attribute_Single spécifie cet attribut.

L'attribut Allocation Information est composé des attributs Allocation choice et Master's MAC ID.

##### 7.7.4.6.2 Allocation choice

L'attribut Allocation choice indique quelle connexion maître/esclave prédéfinie est activée (à l'état de configuration ou à l'état établi). Son format est spécifié dans la CEI 62026-3:2008, 5.5.

L'attribut Allocation choice est remis à 00 à la mise sous tension ou la réinitialisation.

##### 7.7.4.6.3 Master's MAC ID

L'attribut Master's MAC ID contient le MAC ID de l'appareil qui a alloué le jeu de connexions maître/esclave prédéfini par l'intermédiaire du service Allocate_Master/Slave_Connection_Set. Il contient le champ Allocator's MAC ID copié à partir de la demande Allocate_Master/Slave_Connection_Set.

La plage de valeurs est comprise entre 0 et 63 et compte 255 décimales. Une valeur comprise entre 0 et 63 indique que le jeu de connexions maître/esclave prédéfini est actuellement alloué et signale le MAC ID de l'appareil qui a exécuté l'allocation. La valeur 255 indique que le jeu de connexions maître/esclave prédéfini n'a pas été alloué.

L'attribut Master's MAC ID est initialisé à 255 (0xFF) à la mise sous tension/réinitialisation de l'appareil.

#### 7.7.4.7 Compteurs de diagnostic

Cet attribut reporte les comptes des différents types d'erreurs réseau. Le premier champ (descripteur de compteurs de diagnostic) identifie les compteurs pris en charge par l'appareil. Chaque bit indique si un compteur désigné est pris en charge. Si une valeur est attribuée, le compteur est pris en charge. Le Tableau 78 spécifie l'attribution de bit de compteur de diagnostic.

**Tableau 78 – Description de bit des compteurs de diagnostic**

| Bit | Compteur | Peut être supprimé |
|---|---|---|
| 0 | Arbitration loss | Oui |
| 1 | Overload error | Oui |
| 2 | Bit error | Oui |
| 3 | Stuff error | Oui |
| 4 | Ack error | Oui |
| 5 | Form error | Oui |
| 6 | CRC error | Oui |
| 7 | Rx message loss | Oui |
| 8 | Warning error | Oui |
| 9 | Rx error | Non |
| 10 | Tx error | Non |
| 11 – 15 | Réservé (doit être égal à 0) | N/A |

Les champs restants indiquent les valeurs de compteur pour chaque compteur pris en charge. Les compteurs sont avancés de un pour chaque occurrence de l'erreur, sauf pour les compteurs Rx error et Tx error (champs 10 et 11), qui sont incrémentés et décrémentés par le périphérique CAN conformément à la norme ISO 11898, et ne passent pas à zéro.

Tous les compteurs sont réinitialisés lorsque l'appareil entre le message de demande de vérification de l'envoi de MAC ID en double (voir le diagramme de transition d'états d'accès à la liaison de la CEI 62026-3:2008, 5.4).

#### 7.7.4.8 Active Node Table

L'attribut Active Node Table reporte l'état d'activité de chaque nœud du réseau, en fonction du nombre de nœuds. Chaque bit de la matrice représente un nœud du réseau local, la position du bit correspondant à l'adresse de nœud représenté (c'est-à-dire Bit 0 = Adresse de nœud 0, Bit 1 = Adresse de nœud 1). Si une valeur est attribuée au bit (TRUE), le nœud est actif sur la liaison. Si une valeur est supprimée (FALSE), le nœud est inactif sur la liaison.

Une valeur est attribuée au bit (le nœud est indiqué comme étant actif) lorsque l'état du nœud qu'il représente est en ligne, d'envoi du message de demande de vérification de MAC ID en double ou d'attente de message de vérification de MAC ID en double. Le bit doit être défini dans la seconde qui suit un nœud préalablement inactif transmettant sur la liaison. De même, l'appareil contenant l'attribut Active Node Table doit définir tous les bits lors du passage à l'état en ligne.

Le bit est supprimé (le nœud est indiqué comme étant inactif) lorsque le nœud qu'il représente est à l'état inexistant ou à l'état de défaut de communication. Le bit doit être supprimé dans les 20 s qui suivent le retrait du nœud préalablement actif de la liaison (ou, si le nœud est absent du réseau, dans les 20 s qui suivent le passage à l'état en ligne de l'appareil contenant les transitions Active Node Table). De même, l'appareil contenant l'attribut Active Node Table doit supprimer tous les bits lors de l'arrêt de l'état en ligne.

Un routeur ne doit pas tenter d'acheminer une demande de message déconnectée à un nœud du réseau local lorsque le bit Active Node Table est indiqué comme étant inactif (le bit est supprimé). Il doit plutôt immédiatement renvoyer un état d'erreur générale 0x01 et d'erreur étendue 0x0204.

### 7.7.5 Services communs

#### 7.7.5.1 Généralités

L'objet DeviceNet doit prendre en charge les services communs spécifiés dans le Tableau 79.

**Tableau 79 – Services communs de l'objet DeviceNet**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| **0x05** | N/A | Sous condition[a] | Reset | Utilisé pour réinitialiser cette instance de l'objet DeviceNet |
| **0x10** | N/A | Facultatif | Set_Attribute_Single | Définit les informations de configuration spécifiques au réseau ou au nœud |
| **0x0E** | Facultatif | Facultatif | Get_Attribute_Single | Obtient les informations de configuration spécifiques au réseau ou au nœud |

[a] Ce service est requis lorsque cette instance de l'objet DeviceNet peut être adressée par un mécanisme d'accès CPF 2 autre que l'interface de sous-réseau contrôlée par cette instance de l'objet DeviceNet.

#### 7.7.5.2 Reset

Lorsque l'objet DeviceNet reçoit une demande Reset, il

- détermine s'il peut fournir le type de réinitialisation demandée;
- répond à la demande;
- exécute le type de réinitialisation demandée.

Le service commun Reset contient le paramètre spécifique à l'objet spécifié dans le Tableau 80.

**Tableau 80 – Paramètre du service Reset**

| Nom | Type | Description des paramètres de demande | Sémantique des valeurs |
|---|---|---|---|
| Type | USINT | Type de réinitialisation | Voir Tableau 81 |

Le paramètre type du service commun Reset contient les spécifications d'énumération indiquées dans le Tableau 81.

**Tableau 81 – Valeurs de paramètre du service Reset**

| Valeur | Type de réinitialisation |
|---|---|
| 0 | Emuler de la meilleure façon possible la puissance de cyclage sur la liaison de réseau que l'instance de l'objet DeviceNet représente. La valeur est celle par défaut si ce paramètre est ignoré |
| 1 | Renvoie de la meilleure façon possible à la configuration prête à l'emploi, puis émule la puissance de cyclage sur le réseau aussi proche que possible |
| 2 | Renvoie de la meilleure façon possible à la configuration prête à l'emploi, puis émule la puissance de cyclage sur le réseau aussi proche que possible. Le MAC ID et la vitesse en bits ne changent pas suite à cette réinitialisation |
| 3 – 99 | Réservé |
| 100 – 199 | Comportement de réinitialisation spécifique au fournisseur |
| 200 – 255 | Réservé |

### 7.7.6 Services spécifiques à la classe

#### 7.7.6.1 Généralités

L'objet DeviceNet doit prendre en charge les services spécifiques à la classe spécifiés dans le Tableau 82.

**Tableau 82 – Services spécifiques à la classe de l'objet DeviceNet**

| Code de service | Nécessité dans la mise en œuvre | Nom du service | Description du service |
|---|---|---|---|
| **0x4B** | Facultatif | Allocate_Master/Slave_ Connection_Set | Demande l'utilisation du jeu de connexions maître/esclave prédéfini |
| **0x4C** | Sous condition[a] | Release_Master/Slave_ Connection_Set | Indique que les connexions spécifiées dans le jeu de connexions maître/esclave prédéfini ne sont plus souhaitées. Ces connexions doivent être libérées (supprimées) |
| **0x4D** | Sous condition[b] | Clear_Diagnostics | Supprime les compteurs de diagnostic renvoyés dans l'attribut 12 |

[a] Le service Release_Master/Slave_Connection_Set est requis si le service Allocate_Master/Slave_Connection_Set est mis en œuvre dans l'appareil.

[b] Ce service doit être pris en charge si les compteurs qui peuvent être supprimés sont mis en œuvre dans l'attribut Diagnostic Counters.

#### 7.7.6.2 Allocate_Master/Slave_Connection_Set, Release_Master/Slave_Connection_Set

Ces services permettent d'allouer et de désallouer le jeu de connexions maître/esclave prédéfini décrit dans la CEI 62026-3:2008, 5.5.

Un appareil qui se comporte comme le client dans le jeu de connexions maître/esclave prédéfini est appelé maître. Un appareil qui se comporte comme le serveur dans le jeu de connexions maître/esclave prédéfini est appelé esclave. Dans les limites des descriptions du service, un maître et/ou esclave se présente comme une unité fonctionnelle dans un appareil de communication.

Un appareil qui souhaite fonctionner comme le maître d'un autre appareil doit en premier lieu allouer le jeu de connexions maître/esclave prédéfini à l'intérieur de l'esclave. Le jeu de connexions maître/esclave prédéfini peut être alloué à un seul maître à un instant donné. L'ensemble du jeu de connexions est alloué, et le maître utilise un sous-ensemble sélectionné de connexions issues de l'ensemble (c'est-à-dire Bit Strobe uniquement ou Poll uniquement).

Pour « abandonner » son esclave, le maître libère toutes les connexions, l'esclave « désallouant » le jeu de connexions maître/esclave prédéfini.

#### 7.7.6.3 Clear_Diagnostics

Ce service supprime (remet à zéro) les compteurs qui peuvent l'être dans l'attribut 12 (compteurs de diagnostic). Les compteurs qui peuvent être supprimés sont indiqués dans le Tableau 78.

### 7.8 Objet de configuration de connexion (CCO)

#### 7.8.1 Présentation

Cet objet définit une interface de création, de configuration et de contrôle des connexions d'un appareil. Cette spécification ne définit ni ne contraint le fonctionnement du diagramme d'états de gestion des connexions de l'appareil.

NOTE L'objet CCO est un objet qu'il est possible d'adresser à un appareil (n'est pas adressable à un port).

#### 7.8.2 Historique de révision

L'historique de révision de l'objet de configuration de connexion est décrit dans le Tableau 83.

**Tableau 83 – Historique de révision de l'objet de configuration de connexion**

| Révision de la classe | Description des modifications |
|---|---|
| **01** | Définition initiale |
| **02** | Définition intermédiaire |
| **03** | Emission pour CEI 61158 |

#### 7.8.3 Attributs de classe

##### 7.8.3.1 Généralités

L'objet de configuration de connexion doit prendre en charge les attributs de classe spécifiés dans le Tableau 84.

**Tableau 84 – Attributs de classe de l'objet de configuration de connexion**

| ID de l'attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x01** | Obligatoire | Get | NV | Revision | UINT | Révision de cet objet | Troisième révision, valeur = 3 (voir 7.8.2) |
| **0x02** | Obligatoire | Get | NV | Max instance | UDINT | Nombre maximal d'instances | Valeur déterminée par les caractéristiques du nœud |
| **0x03** | Obligatoire | Get | NV | Num instance | UDINT | Nombre de connexions actuellement instanciées | |
| **0x04** | Cet attribut de classe est facultatif et est décrit dans la CEI 61158-5-2 | | | | | | |
| **0x05** | Sous condition[e] | Get | NV | Optional service list | STRUCT de taille variable | Liste des services facultatifs utilisés dans une mise en œuvre de classe d'objet | Liste des codes de service spécifiant les services facultatifs mis en œuvre dans l'appareil pour cette classe |

| ID de l'attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| | | | | Number services | UINT | Nombre de services dans la liste de services facultatifs | Nombre de codes de service de la liste |
| | | | | Optional services | MATRICE d'UINT | Liste des codes de service facultatifs | Codes de service facultatifs |
| **0x06 0x07** | Ces attributs de classe sont facultatifs et sont décrits dans la CEI 61158-5-2 | | | | | | |
| **0x08** | Obligatoire | Get | NV | Numéro de format | UDINT | Voir 7.8.3.2 | |
| **0x09** | Obligatoire | Get/Set | NV | Edit signature | UDINT | Voir 7.8.3.3 | |
| **0x0A – 0x20** | Réservé | | | | | | |
| **0x21** | Facultatif | Get/Set[a, b] | NV | Scanner Mode | BOOL | L'auteur des paquets cible pour les connexions générées par cet appareil doit refléter l'état de cet attribut | 0 = mode veille<br>1 = mode actif |
| **0x22** | Facultatif | Get | V | Scanner capabilities[c] | WORD | Bit 0, définition sur Scanner Mode (attribut 0x21) prise en charge [d]<br>Bit 1, définition sur Scanner Mode (attribut 0x21) actuellement prise en charge<br>Bit 2, les instances peuvent actuellement être créées en mode actif<br>Bit 3, les instances peuvent actuellement être modifiées en mode actif<br>Bit 4, les instances peuvent actuellement être supprimées en mode actif<br>Bit 5, les instances peuvent actuellement être créées en mode veille<br>Bit 6, les instances peuvent actuellement être modifiées en mode veille<br>Bit 7, les instances peuvent actuellement être supprimées en mode veille<br>Bit 8, Services Open_Connection/Close_Connection pris en charge<br>Bit 9, Service Stop_Connection pris en charge<br>Bit 10, Service Get_Member pour lire les tableaux d'image pris en charge<br>Bits 11 à 15 réservés et doivent être nuls | Bits 0 à 10<br>= 0, Non<br>= 1, Oui |

[a] Un attribut 0x21 doit renvoyer une erreur (0x10) de conflit d'état de l'appareil si la modification de Scanner Mode n'est pas admise lors de la réception de la commande de définition d'attribut.

[b] La valeur de l'attribut 0x21 peut être différente de la dernière valeur définie pour l'attribut 0x21 si un autre mécanisme (un commutateur à clé, par exemple) a modifié le mode du scanneur.

[c] Si l'appareil dans lequel cet objet est mis en œuvre intègre un mécanisme de modification des connexions, l'état de ces bits doit être modifié pour refléter l'état présent de l'appareil.

[d] Le bit 0 indique si l'attribut Scanner Mode a la possibilité d'être défini à l'aide d'un service de définition d'attribut.

[e] Requis si l'objet prend en charge des services facultatifs. Facultatif si l'objet ne prend pas en charge des services facultatifs.

#### 7.8.3.2 Numéro de format

Ce numéro détermine le format de l'attribut d'instance 0x09. Cette valeur de format doit être spécifiée dans le champ format_number de chaque attribut d'instance 0x09. La signification des valeurs de format est spécifiée dans le Tableau 85.

**Tableau 85 – Valeurs de numéro de format**

| Valeur | Signification |
|---|---|
| 0 | Tableaux O->T/T->O uniques, mots de 16 bits, décalages 0 |
| 1 | Plusieurs tableaux O->T/T->O, mots de 16 bits, décalages 0 |
| 2 – 99 | Réservé |
| 100 – 199 | Spécifique au fournisseur |
| 200 – ... | Toutes les autres valeurs sont réservées |

#### 7.8.3.3 Edit signature

Créé et utilisé par le logiciel de configuration afin de détecter la modification des valeurs d'attribut de l'instance. Pour CP 2/1, cette valeur, à l'origine égale à 0, est incrémentée à chaque modification complète. Pour tous les autres réseaux CPF 2, cette valeur, à l'origine égale à 0, est définie sur un CRC de 32 bits à chaque modification complète (le polynôme est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+x^{0}$); la valeur de marquage CRC doit être égale à 0. Le CRC est calculé sur le flux de données de définition de tous les attributs pour chaque instance de connexion (du plus bas au plus élevé) plus les attributs de classe 0x01, 0x02, 0x03 et 0x08.

### 7.8.4 Attributs d'instance

#### 7.8.4.1 Généralités

L'objet de configuration de connexion doit prendre en charge les attributs d'instance spécifiés dans le Tableau 86.

**Tableau 86 – Attributs d'instance de l'objet de configuration de connexion**

| ID attri-but | Nécessité dans mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x01** | Obligatoire | Get | V | Etat de connexion | STRUCT de 4 octets | Etat de connexion.<br>Seuls deux octets d'état étendu sont inclus pour les codes d'erreur | Voir 7.8.4.2.<br>NOTE Pour accéder à l'état de connexion complète, il est recommandé de mettre également en œuvre l'attribut 0x17 (Full Connection Status). |
| | | | | Gen_status | USINT | Etat général | La valeur d'état général retournée dans la réponse Forward_Open/ Large_Forward_Open |
| | | | | Réservé | USINT | Réservé | Doit être égal à zéro |
| | | | | Ext_status | UINT | Etat étendu | Si aucun état étendu n'est retourné, Ext_status doit être égal à zéro |
| **0x02** | Obligatoire | Get/Set | NV | Balises de connexion | WORD | Balises de connexion | Voir 7.8.4.3 |
| **0x03** | Obligatoire | Get/Set | NV | Target device ID | STRUCT de 8 octets | | Voir 7.8.4.4 |

| ID attri-but | Nécessité dans mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| | | | | Vendor_id | UINT | ID fournisseur | |
| | | | | Product_type | UINT | Type d'appareil | |
| | | | | Product_code | UINT | Code produit | |
| | | | | Major_rev | USINT | Révision majeure | |
| | | | | Minor_rev | USINT | Révision mineure | |
| **0x04** | Sous condition | Get/Set | NV | CS data index number | UDINT | | Voir 7.8.4.5 |
| **0x05** | Obligatoire | Get/Set | NV | Net connection parameters | STRUCT de 14 octets | | Voir 7.8.4.6 |
| | | | | Conn_timeout | USINT | Multiplicateur de délai de connexion | |
| | | | | Xport_class_and_trig | SWORD | Classe de transport et déclencheur | |
| | | | | Rpi_OT | UDINT | Intervalle du paquet cible demandé auteur/cible | |
| | | | | Net_OT | UINT | Paramètres de connexion de réseau auteur/cible | |
| | | | | Rpi_TO | UDINT | Intervalle de paquet demandé cible/auteur | |
| | | | | Net_TO | UINT | Paramètres de connexion de réseau cible/auteur | |
| **0x06** | Obligatoire | Get/Set | NV | Chemin de connexion | STRUCT de taille variable | | Voir 7.8.4.7 |
| | | | | Open_path_size | USINT | Taille du chemin de connexion ouvert | |
| | | | | Réservé | USINT | Réservé | |
| | | | | Open_connection_ path | EPATH rempli | Chemin de connexion | |
| **0x07** | Obligatoire | Get | NV | Données de configuration du proxy | STRUCT de taille variable | | Voir 7.8.4.8 |
| | | | | Config_data_size | UINT | Longueur de config_data en octets | |
| | | | | Config_data | MATRICE d'octets | Données de configuration du proxy | |
| **0x08** | Obligatoire | Get/Set | NV | Nom de la connexion | STRUCT de taille variable | | Voir 7.8.4.9 |
| | | | | name_size | USINT | Nombre de caractères du nom de connexion. | |
| | | | | Réservé | USINT | Réservé | Doit être égal à zéro |

| ID attri-but | Nécessité dans mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| | | | | connection_name | STRING2 | Nom de connexion attribué par l'utilisateur en UNICODE | |
| **0x09** | Obligatoire | Get/Set | NV | Mapping d'E/S | STRUCT de taille variable | | Voir 7.8.4.10 |
| | | | | format_number | UINT | Ce nombre détermine le format de cet attribut | La valeur de format doit correspondre à la valeur de l'attribut de classe 0x08 |
| | | | | mapping_data_ size | UINT | Taille, en octets, du champ mapping_data qui suit | |
| | | | | mapping_data | MATRICE d'octets | Informations de mapping d'E/S associées à cette instance | |
| **0x0A** | Obligatoire | Get/Set | NV | Target Config data | STRUCT de taille variable | | Voir 7.8.4.11 |
| | | | | Config_data_size | UINT | Longueur de config_data en octets | |
| | | | | Config_data | MATRICE d'octets | Target Config data | |
| **0x0B** | Obligatoire | Get/Set | NV | Proxy device ID | STRUCT de 8 octets | | Voir 7.8.4.12 |
| | | | | Vendor_id | UINT | ID fournisseur | |
| | | | | Product_type | UINT | Type d'appareil | |
| | | | | Product_code | UINT | Code produit | |
| | | | | Major_rev | USINT | Révision majeure | |
| | | | | Minor_rev | USINT | Révision mineure | |
| **0x0C** | Sous condition[a] | Get/Set | NV | Désactiver connexion | BOOL | Indique si cette instance de l'objet de configuration de connexion est désactivée | 0 – Cette instance de l'objet de configuration de connexion est activée<br>1 – Cette instance de l'objet de configuration de connexion est désactivée<br>Si un service Open est reçu, cette valeur doit être égale à 0. Si un service Close ou Stop est reçu, cette valeur doit être égale à 1<br>La valeur par défaut de ce paramètre est 0 |
| **0x0D** | Sous condition[b] | | | Safety Parameters | Voir la CEI 61784-3-2 | | |
| **0x0E** | Sous condition[b] | | | Paramètre de connexion de sécurité CRC | | | |
| **0x0F** | Sous condition[b] | | | Safety Configuration Instance | | | |
| **0x10** | Sous condition[b] | | | Safety ID Allocation | | | |

| ID attri-but | Nécessité dans mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| **0x11** | Sous condition[b] | | | Numéro de série de connexion cible de sécurité | | | |
| **0x12** | Facultatif | Get/Set | | Sélection d'attribut des paramètres de connexion nette | USINT | Sélectionne entre l'attribut 0x05 et l'attribut 0x13 | Voir 7.8.4.13 |
| **0x13** | Sous condition[c] | Get/Set | | Paramètres de connexion nette grands | STRUCT de 18 octets | | Voir 7.8.4.14 |
| | | | | Conn_timeout | USINT | Multiplicateur de délai de connexion | |
| | | | | Xport_class_and_trigger | SWORD | Classe de transport et déclencheur | |
| | | | | Rpi_OT | UDINT | Intervalle du paquet cible demandé auteur/cible | |
| | | | | Net_OT | UDINT | Paramètres de connexion de réseau large auteur/cible | |
| | | | | Rpi_TO | UDINT | Intervalle de paquet demandé cible/auteur | |
| | | | | Net_TO | UDINT | Paramètres de connexion de réseau large cible vers auteur | |
| **0x14** | Sous condition[b] | | | Format Type | Voir la CEI 61784-3-2 | | |
| **0x15** | Sous condition[b] | | | Format Status | | | |
| **0x16** | Sous condition[b] | | | Max Fault Number | | | |
| **0x17** | Facultatif | Get | V | Etat de connexion complète | STRUCT de taille variable | Cet attribut est l'attribut de connexion complète (traite les états étendus plus longs qu'un seul MOT) | |
| | | | | Gen_status | USINT | Etat général | Valeur d'état général retournée dans la réponse Forward_Open/ Large_Forward_Open |
| | | | | Size of Ext_status | USINT | Nombre de MOTS dans la matrice Ext_status | |
| | | | | Ext_status | MATRICE de MOTS | Etat étendu[d] | Valeur d'état vide ou étendu retournée dans la réponse Forward_Open/ Large_Forward_Open |

| ID attri-but | Nécessité dans mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| a L'attribut 0x0C doit être pris en charge si les services conditionnels Open_Connection (0x4C) et Close_Connection (0x4D) sont pris en charge. L'attribut 0x0C ne doit pas être pris en charge si les services conditionnels Open_Connection (0x4C) et Close_Connection (0x4D) ne sont pas pris en charge. | | | | | | | |
| b Ces attributs ne sont pas admis s'il ne s'agit pas d'un appareil de sécurité. S'il s'agit d'un appareil de sécurité, voir la CEI 61784-3-2. | | | | | | | |
| c L'attribut conditionnel 0x13 est obligatoire si l'attribut 0x12 est pris en charge, sinon l'attribut conditionnel 0x13 n'est pas admis. | | | | | | | |
| d Si size_of_Ext_status n'est pas nul, le premier MOT de Ext_status doit avoir la même valeur que l'élément Ext_status dans la structure de l'état de connexion de l'attribut 0x01. | | | | | | | |

### 7.8.4.2 Etat de connexion

Les valeurs de l'état de connexion de l'auteur sont définies dans le Tableau 87.

**Tableau 87 – Valeurs d'état de connexion de l'auteur**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | N/A | La connexion est ouverte |
| 0x01 | 0x0000 ou supérieur | Voir les codes d'erreur de la demande de service Connection Manager, CEI 61158-6-2 |
| 0x02 – 0x26 | 0x0000 ou supérieur | Voir les codes d'état général, CEI 61158-6-2 |
| 0xD0 | 0x0001 | La connexion est fermée ou arrêtée (par l'intermédiaire du service Close ou Stop) |
| | 0x0002 | L'ouverture de connexion est en attente |
| | 0x0003 | La fermeture de connexion est en attente |

Les valeurs de l'état de connexion cible sont définies dans le Tableau 88.

**Tableau 88 – Valeurs d'état de connexion cible**

| Etat général | Etat étendu | Description de l'erreur |
|---|---|---|
| 0x00 | 0x0000 | La connexion est ouverte |
| 0x01 | 0x0001 ou supérieur | Nombre de connexions à cette cible |
| 0xD0 | 0X0001 | La connexion est fermée ou arrêtée (par l'intermédiaire du service Close ou Stop) |

### 7.8.4.3 Balises de connexion

Les profils binaires des balises de connexion sont définis dans le Tableau 89.

**Tableau 89 – Balises de connexion**

| Bit | Signification |
|---|---|
| 0 | Connexion<br>0 = Auteur<br>1 = Cible |

| Bit | Signification |
|---|---|
| 1 – 3 | O->T Format de transfert en temps réel<br>000 – Utiliser l'en-tête Exécution/Programme à 32 bits pour indiquer le mode veille<br>001 – Utiliser le paquet de longueur nulle pour indiquer le mode veille<br>010 – Aucun. La connexion est composée de données pures et se présente sans mode<br>011 – Impulsion – pas de notification d'activité/de veille<br>100 – Réservé<br>101 – Sécurité (voir CEI 61784-3-2)<br>100 à 111 – Réservés à une utilisation ultérieure |
| 4 – 6 | T->O Format de transfert en temps réel<br>000 – Utiliser l'en-tête Exécution/Programme à 32 bits pour indiquer le mode veille<br>001 – Utiliser le paquet de longueur nulle pour indiquer le mode veille<br>010 – Aucun. La connexion est composée de données pures et se présente sans mode<br>011 – Impulsion – pas de notification d'activité/de veille<br>100 – Réservé<br>101 – Sécurité (voir CEI 61784-3-2)<br>100 à 111 – Réservés à une utilisation ultérieure |
| 7 – 15 | Réservé |

#### 7.8.4.4 Target device ID

Cet attribut représente l'identité de l'appareil cible pour cette instance de connexion. Ces informations d'identité ne sont pas utilisées pour vérifier (moduler) l'appareil cible en ligne. Elles sont utilisées par un outil de configuration afin de rechercher la feuille de données électronique correcte de la configuration de connexion décrite dans cette instance CCO. Pour les instances cible, cet attribut doit être l'identité de l'appareil contenant cet objet CCO.

#### 7.8.4.5 CS data index number

L'attribut CS Data Index Number est requis pour un appareil sur CP 2/1. Sinon, cet attribut ne doit pas être mis en œuvre.

L'attribut CS Data Index Number est une valeur définie par le logiciel de configuration. Il s'agit de la valeur connection_index renvoyée du service de lecture de l'objet de planification (voir 7.4.4 pour obtenir une définition des services de l'objet de planification). Cet attribut est ignoré pour les instances cible.

#### 7.8.4.6 Net connection parameters

##### 7.8.4.6.1 conn_timeout

conn_timeout est la valeur utilisée pour le champ de multiplicateur de délai de connexion défini dans la CEI 61158-6-2 (demande Forward_Open). conn_timeout est ignoré pour les instances cible.

##### 7.8.4.6.2 xport_class_and_trig

xport_class_and_trig est la valeur utilisée pour le champ Transport Type/Trigger de la CEI 61158-6-2 (demande Forward_Open).

Le champ xport_class_and_trig doit être ignoré pour les instances cible. L'appareil contenant cet objet CCO doit déterminer si le paramètre similaire Transport Type/Trigger de la demande

Forward_Open est pris en charge comme cela est spécifié dans la CEI 61158-6-2, 4.1.6.14.

#### 7.8.4.6.3 rpi_OT

rpi_OT est la valeur utilisée pour le champ O_to_T RPI défini dans la CEI 61158-6-2 (demande Forward_Open). rpi_OT est ignoré pour les instances cible.

#### 7.8.4.6.4 net_OT

net_OT est la valeur utilisée pour le champ des paramètres de connexion de réseau O_to_T de la CEI 61158-6-2 (demande Forward_Open).

Le champ net_OT doit être ignoré pour les instances cible, à l'exception de la taille de connexion. L'appareil contenant cet objet CCO doit déterminer si le paramètre similaire O->T Network Connection Parameters de la demande Forward_Open est pris en charge comme le spécifie la CEI 61158-6-2, 4.1.6.1.

#### 7.8.4.6.5 rpi_TO

rpi_TO est la valeur utilisée pour le champ T_to_O RPI défini dans la CEI 61158-6-2 (demande Forward_Open). rpi_TO est ignoré pour les instances cible.

#### 7.8.4.6.6 net_TO

net_TO est la valeur utilisée pour le champ des paramètres de connexion de réseau T_to_O de la CEI 61158-6-2 (demande Forward_Open).

Le champ net_TO doit être ignoré pour les instances cible, à l'exception de la taille de connexion. L'appareil contenant cet objet CCO doit déterminer si le paramètre similaire T->O Network Connection Parameters de la demande Forward_Open est pris en charge comme le spécifie la CEI 61158-6-2, 4.1.6.1.

### 7.8.4.7 Chemin de connexion

#### 7.8.4.7.1 open_path_size

open_path_size est la valeur utilisée pour le champ Connection_Path_size de la CEI 61158-6-2 (demandes Forward_Open/Large_Forward_Open). Pour les instances cible, open_path_size permet de mettre en correspondance les Connection_Path_Size reçus dans une demande Forward_Open/Large_Forward_Open.

#### 7.8.4.7.2 open connection path

open_connection_path est la valeur utilisée pour le champ Connection_Path de la CEI 61158-6-2 (demandes Forward_Open/Large_Forward_Open). Pour les instances cible, open_connection_path permet de mettre en correspondance le Connection_Path reçu dans une demande Forward_Open/Large_Forward_Open.

### 7.8.4.8 Données de configuration du proxy

Ne concerne pas les instances cible. Les données spécifiées dans les attributs 0x07 et 0x0A sont concaténées (dans cet ordre) en un seul segment de données, puis ajoutées au chemin de connexion de l'attribut 0x06 afin de former le chemin complet envoyé dans le service Forward_Open de l'objet Connection Manager.

Les données de configuration peuvent être réparties entre les attributs 0x07 et 0x0A. Par convention, si la connexion ne passe pas par un serveur proxy, toutes les données de configuration doivent être placées dans l'attribut 0x0A (Target Config). Par convention, si la connexion passe par un serveur proxy, les données de configuration destinées à l'appareil

mandataire doivent être placées dans l'attribut 0x07 (Proxy Config), celles destinées à l'appareil mandaté étant placées dans l'attribut 0x0A (Target Config). Proxy Configdata est ignoré pour les instances cible.

#### 7.8.4.9 Nom de la connexion

Ce champ Connection Name permet à un utilisateur de nommer chaque instance de connexion. Le nom de connexion n'a aucune signification pour l'objet de configuration de connexion.

#### 7.8.4.10 Mapping d'E/S

L'attribut mapping d'E/S contient des données de mapping d'E/S. Les données de mapping d'E/S indiquent les emplacements de table d'image dans lesquels les données cible/auteur sont placées et où les données auteur/cible sont obtenues.

Le Tableau 90 décrit la structure du champ mapping_data de l'attribut mapping d'E/S pour chacun des numéros de format.

**Tableau 90 – Formats de mapping d'E/S**

| Numéro de format | Mapping_data_size (en octets) | Nom | Type de données | Description |
|---|---|---|---|---|
| 0 | 4 | Tables d'E-S uniques | STRUCT de 4 octets | mots de 16 bits, décalages de mot de 16 bits en base 0 |
| | | Décalage O->T | UINT | Décalage dans la table d'image O->T |
| | | Décalage T->O | UINT | Décalage dans la table d'image T->O |
| 1 | 8 | Tables d'E-S multiples | STRUCT de 8 octets | Sélection de table, mots de 16 bits, décalages de mot de 16 bits en base 0 |
| | | Table O->T | UINT | Sélection de table d'image O->T |
| | | Décalage O->T | UINT | Décalage dans la table d'image O->T |
| | | Table T->O | UINT | Sélection de table d'image T->O |
| | | Décalage T->O | UINT | Décalage dans la table d'image T->O |

#### 7.8.4.11 Target Config data

Cette note ne concerne pas les instances cible. Les données spécifiées dans les attributs 0x07 et 0x0A sont concaténées (dans cet ordre) en un seul segment de données, puis ajoutées au chemin de connexion de l'attribut 0x06 afin de former le chemin complet envoyé dans le service Forward_Open de l'objet Connection Manager. Toutes les données de configuration peuvent être placées dans l'attribut 0x07, dans l'attribut 0x0A ou être réparties entre les attributs 0x07 et 0x0A. Par convention, si la connexion passe par un serveur proxy, les données de configuration destinées à l'appareil mandataire doivent être placées dans l'attribut 0x07, celles destinées à l'appareil mandaté étant placées dans l'attribut 0x0A. Target Config est ignoré pour les instances cible.

#### 7.8.4.12 Proxy Device ID

Cet attribut représente l'identité de l'appareil mandataire pour l'appareil cible de cette instance de connexion. Ces informations d'identité ne sont pas utilisées pour vérifier (moduler) l'appareil cible en ligne. Elles sont utilisées par un outil de configuration afin de rechercher la feuille de données électronique correcte de la configuration de connexion décrite dans cette instance CCO. Pour les instances cible et les connexions qui ne sont pas mandatées, cet attribut doit être ignoré.

#### 7.8.4.13 Sélection d'attribut des paramètres de connexion nette

L'attribut Sélection d'attribut des paramètres de connexion nette sélectionne entre l'utilisation de l'attribut d'instance 0x05 (Net Connection Parameters) et l'attribut 0x13 (Paramètres de connexion nette grands).

Les valeurs valides sont les suivantes:

0 = indique que l'attribut d'instance 5 doit être utilisé (par défaut);

1 = indique que l'attribut d'instance 19 doit être utilisé;

2-255 = réservé.

Si cet attribut facultatif n'est pas pris en charge, l'attribut 0x05 doit être utilisé.

#### 7.8.4.14 Paramètres de connexion nette grands

##### 7.8.4.14.1 conn_timeout

conn_timeout est la valeur utilisée pour le champ de multiplicateur de délai de connexion défini dans la CEI 61158-6-2 (demande Large_Forward_Open). conn_timeout est ignoré pour les instances cible.

##### 7.8.4.14.2 xport_class_and_trig

xport_class_and_trig est la valeur utilisée pour le champ Transport Type/Trigger défini dans la CEI 61158-6-2 (demande Large_Forward_Open).

Le champ xport_class_and_trig doit être ignoré pour les instances cible. L'appareil contenant cet objet CCO doit déterminer si le paramètre similaire Transport Type/Trigger de la demande Large_Forward_Open est pris en charge comme le spécifie la CEI 61158-6-2, 4.1.6.14.

##### 7.8.4.14.3 rpi_OT

rpi_OT est la valeur utilisée pour le champ O->T RPI défini dans la CEI 61158-6-2 (demande Large_Forward_Open). rpi_OT est ignoré pour les instances cible.

##### 7.8.4.14.4 net_OT

net_OT est la valeur utilisée pour le champ O->T Network Connection Parameters défini dans la CEI 61158-6-2 (demande Large_Forward_Open).

Le champ net_OT doit être ignoré pour les instances cible, à l'exception de la taille de connexion. L'appareil contenant cet objet CCO doit déterminer si le paramètre similaire O->T Network Connection Parameters de la demande Large_Forward_Open est pris en charge comme le spécifie la CEI 61158-6-2, 4.1.6.1.

##### 7.8.4.14.5 rpi_TO

rpi_TO est la valeur utilisée pour le champ T->O RPI défini dans la CEI 61158-6-2 (demande Large_Forward_Open). rpi_TO est ignoré pour les instances cible.

##### 7.8.4.14.6 net_TO

net_TO est la valeur utilisée pour le champ T->O Network Connection Parameters défini dans la CEI 61158-6-2 (demande Large_Forward_Open).

Le champ net_TO doit être ignoré pour les instances cible, à l'exception de la taille de connexion. L'appareil contenant cet objet CCO doit déterminer si le paramètre similaire T->O

Network Connection Parameters de la demande Large_Forward_Open est pris en charge comme le spécifie la CEI 61158-6-2, 4.1.6.1.

### 7.8.5 Contrôle de modification de l'objet de configuration de connexion

Les modifications des attributs de l'objet de configuration de la connexion peuvent uniquement être réalisées après qu'un service Change_Start a correctement été émis. Les modifications des attributs de l'objet de configuration de la connexion peuvent uniquement être réalisées après qu'un service Change_Complete a correctement été émis.

Les services qui sont valides pendant une opération de modification sont répertoriés dans le Tableau 91. Une opération de modification est déclenchée par l'émission d'un service Change_Start et terminée par un service Change_Complete.

**Tableau 91 – Services valides pendant une opération de modification**

| Code de service | Nom du service |
|---|---|
| 0x02 | Set_Attribute_All |
| 0x08 | Create |
| 0x09 | Delete |
| 0x10 | Set_Attribute_Single |
| 0x15 | Restore |
| 0x4B | Kick_Timer |
| 0x51 | Change_Complete |
| 0x52 | Audit_Changes |

### 7.8.6 Services communs

#### 7.8.6.1 Généralités

L'objet de configuration de la connexion doit prendre en charge les services communs spécifiés dans le Tableau 92.

**Tableau 92 – Services communs de l'objet de configuration de la connexion**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x01 | Obligatoire | Obligatoire | Get_Attribute_All | Obtient tous les attributs de l'instance spécifiée |
| 0x02 | N/A | Obligatoire | Set_Attribute_All | Définit tous les attributs de l'instance spécifiée |
| 0x08 | Obligatoire | N/A | Create | Crée une nouvelle instance de connexion |
| 0x09 | Obligatoire | Obligatoire | Delete | Supprime une instance de connexion existante |
| 0x0E | Facultatif | Obligatoire | Get_Attribute_Single | Renvoie le contenu de l'attribut spécifié |
| 0x10 | Obligatoire | Facultatif | Set_Attribute_Single | Modifie une valeur d'attribut |
| 0x15 | Obligatoire | Obligatoire | Restore | Restaure les attributs de la connexion actuelle |

#### 7.8.6.2 Get_Attribute_All

La structure de la réponse Get_Attribute_All au niveau de la classe est détaillée dans le Tableau 93.

**Tableau 93 – Réponse Get_Attribute_All – niveau de la classe**

| Nom | Type de données | Description |
|---|---|---|
| Revision | UINT | Révision de l'objet |
| Max instance | UDINT | Nombre maximal d'instances |
| Num instance | UDINT | Nombre de connexions actuellement instanciées |
| Numéro de format | UINT | La valeur de format doit être spécifiée dans le champ de format de chacune des instances de l'attribut 9 |
| Edit signature | UDINT | Valeur créée et utilisée par le logiciel de configuration afin de détecter la modification des valeurs d'attribut de l'instance. |

La structure de la réponse Get_Attribute_All au niveau de l'instance est détaillée dans le Tableau 94.

**Tableau 94 – Réponse Get_Attribute_All – niveau de l'instance**

| Nom | Type de données | Description |
|---|---|---|
| Etat de connexion | STRUCT de 4 octets | Informations de l'état de connexion. Lorsque la connexion n'est pas ouverte, cet attribut contient un code d'erreur indiquant la raison |
| | USINT | Etat général |
| | USINT | Plage |
| | UINT | Etat étendu |
| Balises de connexion | WORD | Balises de connexion |
| Target device ID | STRUCT de 8 octets | Identification de l'appareil cible |
| | UINT | ID fournisseur |
| | UINT | Type de produit |
| | UINT | Code produit |
| | USINT | Révision majeure |
| | USINT | Révision mineure |
| CS data index number | UDINT | Numéro connection_index des données de lecture de l'objet de planification. La valeur par défaut utilisée lorsque cet attribut n'est pas pris en charge doit être 0xFFFFFFFF |
| Net connection parameters | STRUCT de 14 octets | Paramètres de connexion de réseau des directions auteur/cible et cible/auteur |
| | USINT | Multiplicateur de délai de connexion |
| | SWORD | Classe de transport et déclencheur |
| | UDINT | RPI auteur/cible |
| | UINT | Paramètres de connexion de réseau auteur/cible |
| | UDINT | RPI cible/auteur |
| | UINT | Paramètres de connexion de réseau cible/auteur |
| Chemin de connexion | STRUCT de taille variable | Chemin de connexion |
| | USINT | Taille du chemin de connexion, en mots de 16 bits, utilisée dans la demande de service Forward_Open du gestionnaire de connexion |
| | USINT | Réservé. Doit être égal à zéro |

| Nom | Type de données | Description |
|---|---|---|
| | EPATH rempli | Chemin de connexion. La taille du chemin de connexion ouvert est égale à la longueur de cette matrice |
| Données de configuration du proxy | STRUCT de taille variable | Données de configuration proxy. Ces données sont envoyées aux appareils par l'intermédiaire des demandes de services Forward_Open/Large_Forward_Open du gestionnaire de connexion |
| | UINT | Longueur des données de configuration, en octets |
| | MATRICE d'USINT | Données de configuration remplies jusqu'à un nombre pair d'octets |
| Target Config data | STRUCT de taille variable | Données de configuration cible. Ces données sont envoyées aux appareils par l'intermédiaire des demandes de services Forward_Open/Large_Forward_Open du gestionnaire de connexion |
| | UINT | Longueur des données de configuration, en octets |
| | MATRICE d'USINT | Données de configuration de module remplies jusqu'à un nombre pair d'octets |
| Nom de la connexion | STRUCT de taille variable | Nom de la connexion |
| | USINT | Nombre de caractères du nom de connexion |
| | USINT | Plage |
| | STRING2 | Nom de la connexion, codé en UNICODE |
| Mapping d'E/S | STRUCT de taille variable | Mapping d'E/S |
| | UINT | Numéro de format |
| | UINT | Taille de l'attribut, en octets |
| | MATRICE d'USINT | Données d'attribut remplies jusqu'à un nombre pair d'octets |
| Proxy device ID | STRUCT de 8 octets | Identification de l'appareil proxy |
| | UINT | ID fournisseur |
| | UINT | Type de produit |
| | UINT | Code produit |
| | USINT | Révision majeure |
| | USINT | Révision mineure |
| Safety Parameters | STRUCT de 55 octets | Voir la CEI 61784-3-2.<br>Ces 55 octets sont inclus dans la réponse Get_Attribute_All uniquement si le format de transfert en temps réel O=>T ou T=>O de l'attribut Connection Flags (attribut 0x02) indique que le format est Safety |
| Désactiver connexion | BOOL | Bit 0, valeur de l'attribut de désactivation de connexion (attribut 0x0C), s'il est pris en charge. 0, si l'attribut de désactivation de connexion n'est pas pris en charge |
| Sélection d'attribut des paramètres de connexion nette | USINT | Sélection de l'attribut Net Connection Parameters ou de l'attribut Paramètres de connexion nette grands s'il est pris en charge. 0 si Sélection d'attribut des paramètres de connexion nette n'est pas pris en charge |
| Paramètres de connexion nette grands | STRUCT de 18 octets | Paramètres de connexion réseau pour O=>T et T=>O |
| | USINT | Multiplicateur de délai de connexion |
| | SWORD | Classe de transport et déclencheur |
| | UDINT | RPI O->T |
| | UDINT | Paramètres de connexion réseau O->T |

| Nom | Type de données | Description |
|---|---|---|
| | UDINT | RPI T->O |
| | UDINT | Paramètres de connexion réseau T->O |
| Additional safety parameters | STRUCT de taille variable | Voir la CEI 61784-3-2.<br>Ce groupe de paramètres peut être inclus dans la réponse Get_Attribute_All uniquement si le format de transfert en temps réel O=>T ou T=>O de l'attribut Connection Flags (attribut 0x02) indique que le format est Safety |

#### 7.8.6.3 Set_Attribute_All

Ce service doit définir tous les attributs associés à une instance existante et prendre en charge les codes d'erreur présentés dans le Tableau 95, en complément des codes d'erreur répertoriés dans la CEI 61158-6-2, 4.1.11.2.1.

**Tableau 95 – Codes d'erreur Set_Attribute_All**

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| 0x0C | Aucune | Conflit d'état d'objet | L'objet de configuration de connexion ne peut accepter le service Set_Attribute_Single lorsqu'une session d'édition n'est pas active |

La structure de la demande Set_Attribute_All est présentée dans le Tableau 96.

**Tableau 96 – Demande Set_Attribute_All**

| Nom | Type de données | Description |
|---|---|---|
| Balises de connexion | WORD | Balises de connexion |
| Target device ID | STRUCT de 8 octets | Identification de l'appareil cible |
| | UINT | ID fournisseur |
| | UINT | Type de produit |
| | UINT | Code produit |
| | USINT | Révision majeure |
| | USINT | Révision mineure |
| CS data index number | UDINT | Numéro connection_index des données de lecture de l'objet de planification. La valeur par défaut utilisée lorsque cet attribut n'est pas pris en charge doit être 0xFFFFFFFF |
| Net connection parameters | STRUCT de 14 octets | Paramètres de connexion de réseau pour les directions O⇒T et T⇒O. |
| | USINT | Multiplicateur de délai de connexion |
| | SWORD | Classe de transport et déclencheur |
| | UDINT | RPI O⇒T |
| | UINT | Paramètres de connexion réseau O⇒T |
| | UDINT | RPI T⇒O |
| | UINT | Paramètres de connexion réseau T⇒O |
| Chemin de connexion | STRUCT de taille variable | Chemin de connexion |
| | USINT | Taille du chemin de connexion, en mots de 16 bits, utilisée dans la demande de service Forward_Open du gestionnaire de connexion |

| Nom | Type de données | Description |
|---|---|---|
| | USINT | Réservé. Doit être égal à zéro |
| | EPATH rempli | Chemin de connexion. La taille du chemin de connexion ouvert est égale à la longueur de cette matrice |
| Données de configuration du proxy | STRUCT de taille variable | Données de configuration proxy. Ces données sont envoyées aux appareils par l'intermédiaire de la demande de service Forward_Open du gestionnaire de connexion |
| | UINT | Longueur des données de configuration de module, en octets |
| | MATRICE d'USINT | Données de configuration de module remplies jusqu'à un nombre pair d'octets |
| Target Config data | STRUCT de taille variable | Données de configuration cible. Ces données sont envoyées aux appareils par l'intermédiaire de la demande de service Forward_Open du gestionnaire de connexion |
| | UINT | Longueur des données de configuration de module, en octets |
| | MATRICE d'USINT | Données de configuration de module remplies jusqu'à un nombre pair d'octets |
| Nom de la connexion | STRUCT de taille variable | Nom de la connexion |
| | USINT | Nombre de caractères du nom de connexion |
| | USINT | Plage |
| | STRING2 | Nom de la connexion, codé en UNICODE |
| Mapping d'E/S | STRUCT de taille variable | Mapping d'E/S |
| | UINT | Numéro de format |
| | UINT | Taille de l'attribut, en octets |
| | MATRICE d'USINT | Données d'attribut remplies jusqu'à un nombre pair d'octets |
| Proxy device id | STRUCT de 8 octets | Identification de l'appareil proxy |
| | UINT | ID fournisseur |
| | UINT | Type de produit |
| | UINT | Code produit |
| | USINT | Révision majeure |
| | USINT | Révision mineure |
| Safety Parameters | STRUCT de 49 octets | Voir la CEI 61784-3-2.<br>Ces 49 octets sont inclus dans les données du service Set_Attribute_All uniquement si le format de transfert en temps réel O=>T ou T=>O de l'attribut Connection Flags (attribut 0x02) indique que le format est Safety |
| Désactiver connexion | BOOL | Bit 0, valeur de l'attribut de désactivation de connexion (attribut 12) |
| Sélection d'attribut des paramètres de connexion nette | USINT | Sélection de l'attribut Net Connection Parameters ou de l'attribut Paramètres de connexion nette grands s'il est pris en charge. |
| Paramètres de connexion nette grands | STRUCT de 18 octets | Paramètres de connexion réseau pour O=>T et T=>O |
| | USINT | Multiplicateur de délai de connexion |
| | SWORD | Classe de transport et déclencheur |
| | UDINT | RPI O->T |

| Nom | Type de données | Description |
|---|---|---|
| | UDINT | Paramètres de connexion réseau O->T |
| | UDINT | RPI T->O |
| | UDINT | Paramètres de connexion réseau T->O |
| Additional safety parameters | STRUCT de taille variable | Voir la CEI 61784-3-2.<br>Ce groupe de paramètres peut être inclus dans la réponse Set_Attribute_All uniquement si le format de transfert en temps réel O=>T ou T=>O de l'attribut Connection Flags (attribut 0x02) indique que le format est Safety |

Les données pour les attributs non pris en charge doivent être acceptées si les valeurs par défaut sont envoyées. Le service doit être rejeté si les valeurs qui ne sont pas par défaut sont envoyées pour les attributs non pris en charge.

#### 7.8.6.4 Create

Ce service crée une instance d'un objet de configuration de connexion. Les valeurs d'attribut initiales peuvent également être spécifiées avec ce service. Le numéro de l'instance créée est attribué par la classe et renvoyé au demandeur. Le Tableau 97 définit les paramètres de demande pour ce service.

**Tableau 97 – Paramètres de demande Create**

| Paramètre | Type de données | Description |
|---|---|---|
| Balises de connexion | WORD | Balises de connexion |
| NumConnParams | UINT | Nombre de paires attribut/valeur qui suivent |
| ConnParams | MATRICE de | Liste des paires numéro/valeur de l'attribut |
| | STRUCT de taille variable | |
| | UINT | Numéro d'attribut |
| | STRUCT spécifique à l'attribut d'objet/de classe | Valeur d'attribut |

Les données pour les attributs non pris en charge doivent être acceptées si les valeurs par défaut sont envoyées. Le service doit être rejeté si les valeurs qui ne sont pas par défaut sont envoyées pour les attributs non pris en charge.

Ce service doit lire tous les attributs associés à une instance existante et prendre en charge les codes d'erreur présentés dans le Tableau 98, en complément des codes d'erreur répertoriés dans la CEI 61158-6-2, 4.1.11.2.1.

**Tableau 98 – Codes d'erreur Create**

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| 0x02 | 0x0001 | Ressources insuffisantes | Le nombre maximal d'instances existe déjà |
| | 0x0002 | Ressources insuffisantes | Mémoire insuffisante sur l'appareil |
| 0x03 | Aucun | Erreur de paramètre non valide | Le nombre d'attributs n'est pas valide |
| 0x08 | Aucun | Service non pris en charge | Service non mis en œuvre |
| 0x0C | Aucun | Conflit d'état d'objet | L'objet de configuration de connexion ne peut exécuter le service Set_Attribute_Single lorsqu'une session d'édition n'est pas active |

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| 0x0E | Aucun | Attribut non définissable | Tentative de définition d'un attribut en lecture seule |
| 0x13 | Aucun | Données de demande insuffisantes | La demande est trop courte ou tronquée |
| 0x1C | Aucun | Pénurie de listes d'attributs | Il manque l'attribut requis |

#### 7.8.6.5 Delete

Ce service permet de supprimer des instances de connexion existantes. S'il est adressé au niveau de la classe, toutes les instances de connexion sont supprimées. S'il est adressé au niveau de l'instance, seule l'instance adressée est supprimée. Ce service doit prendre en charge les codes d'erreur présentés dans le Tableau 99, en complément des codes d'erreur répertoriés dans la CEI 61158-6-2, 4.1.11.2.1.

**Tableau 99 – Codes d'erreur Delete**

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| 0x0C | Aucun | Conflit d'état d'objet | L'objet de configuration de connexion ne peut exécuter le service Delete lorsqu'une session d'édition n'est pas active |

#### 7.8.6.6 Restore

Ce service doit écarter les modifications apportées aux instances qui n'ont pas encore été validées par le service Change_Complete. Si le service Restore est adressé à la classe (instance 0), les modifications en attente pour toutes les instances doivent être ignorées et la session d'édition doit prendre fin. S'il est adressé à une instance particulière, seules les modifications de cette instance doivent être ignorées, et la session d'édition ne doit pas prendre fin.

Ce service doit prendre en charge les codes d'erreur présentés dans le Tableau 100, en complément des codes d'erreur répertoriés dans la CEI 61158-6-2, 4.1.11.2.1.

**Tableau 100 – Codes d'erreur Restore**

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| 0x0C | Aucun | Conflit d'état d'objet | L'objet de configuration de connexion ne peut exécuter le service Restore lorsqu'une session d'édition n'est pas active |

### 7.8.7 Services spécifiques à la classe

#### 7.8.7.1 Généralités

L'objet de configuration de connexion doit prendre en charge les services spécifiques à la classe spécifiés dans le Tableau 101.

**Tableau 101 – Services spécifiques à la classe de l'objet de configuration de connexion**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x4B | Obligatoire | N/A | Kick_Timer | Temporisateur d'édition des expulsions |
| 0x4C | Sous condition[a] | Sous condition[a] | Open_Connection | Permet d'établir des connexions |
| 0x4D | Sous condition[a] | Sous condition[a] | Close_Connection | Permet d'interrompre des connexions |
| 0x4E | Sous condition[a] | Sous condition[a] | Stop_Connection | Permet d'arrêter des connexions |
| 0x4F | Obligatoire | N/A | Change_Start | Permet de gérer l'édition de session |
| 0x50 | Obligatoire | N/A | Get_Status | Etat Get pour plusieurs connexions |
| 0x51 | Obligatoire | N/A | Change_Complete | Permet de mettre un terme à l'édition de session |
| 0x52 | Obligatoire | N/A | Audit_Changes | Audit des modifications en attente |

[a] Soit les services Open_Connection et Close_Connection doivent être pris en charge, soit aucun d'eux ne doit l'être. Le service Stop_Connection peut uniquement être pris en charge si le service Open_Connection l'est également

#### 7.8.7.2 Kick_Timer

Ce service doit réinitialiser le temporisateur d'édition. Suite à l'exécution du service Change_Start, le temporisateur d'édition doit être démarré dans les 60 s. Ce temporisateur permet une reprise en cas de perte d'un client de configuration entre les opérations Change_Start et Change_Complete/Restore. La réception d'une demande de service doit réinitialiser le temporisateur d'édition. Les clients peuvent demander ce service pour réinitialiser le temporisateur sans affecter l'état de l'objet de configuration de connexion. Si le temporisateur d'édition expire, toutes les modifications en attente doivent être ignorées et la session d'édition doit prendre fin.

#### 7.8.7.3 Open_Connection

Open_Connection doit provoquer l'ouverture de la connexion associée à une instance de l'objet de configuration de connexion. Si le service Open_Connection est adressé à la classe (instance 0), toutes les instances de connexion doivent être ouvertes. S'il est adressé à une instance particulière, seule cette instance de connexion doit être ouverte.

#### 7.8.7.4 Close_Connection

Close_Connection doit provoquer la fermeture de la connexion associée à une instance de l'objet de configuration de connexion. Si le service Close_Connection est adressé à la classe (instance 0), toutes les instances de connexion doivent être fermées. S'il est adressé au niveau de l'instance, seule l'instance spécifiée doit être fermée. Le service Close_Connection doit permettre d'arrêter la connexion « en douceur ». En d'autres termes, une demande Forward_Close doit être envoyée à la cible de connexion. Une fois la connexion fermée par ce service, elle doit le rester tant qu'un service Open_Connection n'a pas été émis.

#### 7.8.7.5 Stop_Connection

Le service Stop_Connection doit provoquer l'arrêt immédiat de la production de données par la connexion associée à une instance de l'objet de configuration de connexion, sans envoyer de demande Forward_Close à la cible de connexion. Si le service Stop_Connection est

adressé à la classe (instance 0), toutes les instances de connexion doivent être arrêtées. S'il est adressé au niveau de l'instance, seule l'instance spécifiée doit être arrêtée. Une fois la connexion arrêtée, elle le reste tant qu'une autre demande Open_Connection n'est pas émise.

#### 7.8.7.6 Change_Start

Ce service doit

a) signaler le début d'une session d'édition;

b) synchroniser les attributs en cours et en attente;

c) placer toutes les connexions à l'état « modifiable »;

d) démarrer le temporisateur d'édition.

Le service Change_Start doit être demandé avant d'exécuter des services de modification des attributs d'une connexion. Ce service doit uniquement être adressé au niveau de la classe (instance 0). Si un service Change_Start est reçu alors qu'une session d'édition est active, une erreur 0x0C (Conflit d'état de l'objet) doit être renvoyée.

Ce service doit prendre en charge les codes d'erreur présentés dans le Tableau 102, en complément des codes d'erreur répertoriés dans la CEI 61158-6-2, 4.1.11.2.1.

**Tableau 102 – Codes d'erreur Change_Start**

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| 0x0C | Aucun | Conflit d'état d'objet | L'objet de configuration de connexion ne peut exécuter le service Restore lorsqu'une session d'édition n'est pas active |
| 0x10 | Aucun | Conflit d'état de l'appareil | L'appareil est dans un état qui empêche le démarrage d'une session d'édition |

#### 7.8.7.7 Get_Status

Le service Get_Status doit extraire l'attribut d'état (attribut 1) de plusieurs connexions par l'intermédiaire d'une seule transaction. Ce service doit être pris en charge au niveau de la classe (instance 0) uniquement. Selon le numéro d'instance de démarrage, le service Get_Status doit renvoyer des paires instance/état tant que la mémoire tampon de réponse n'est pas pleine ou que l'état de toutes les connexions n'a pas été renvoyé.

Les paramètres de demande sont définis dans le Tableau 103.

**Tableau 103 – Paramètre du service Get_Status**

| Paramètre | Type de données | Description |
|---|---|---|
| Starting Instance | UDINT | Numéro d'instance de démarrage |

Les paramètres de réponse sont définis pour ce service dans le Tableau 104.

**Tableau 104 – Réponse du service Get_Status**

| Paramètre | Type de données | Description |
|---|---|---|
| Done Indicator | UINT | 0 = Plus d'états à extraire<br>1 = Toutes les informations relatives à l'état de la connexion ont été extraites.<br>Toutes les autres valeurs réservées. |

| Paramètre | Type de données | Description |
|---|---|---|
| NumStatusEntries | UINT | Nombre de paires instance/état qui suivent |
| StatusEntries | MATRICE de | Liste des paires instance/état |
| | STRUCT de 8 octets | |
| | UDINT | Numéro d'instance de la configuration de connexion |
| | USINT | Etat général |
| | USINT | Réservé, doit être égal à 0 |
| | UINT | Etat étendu |

Ce service doit prendre en charge les codes d'erreur présentés dans le Tableau 105, en complément des codes d'erreur répertoriés dans la CEI 61158-6-2, 4.1.11.2.1.

**Tableau 105 – Codes d'erreur du service Get_Status**

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| 0x03 | Aucun | Erreur de paramètre non valide | Le nombre d'attributs n'est pas valide |

### 7.8.7.8 Change_Complete

Ce service doit signaler la fin d'une session d'édition. Les attributs en attente de toutes les instances de connexion modifiées doivent être appliqués. Ce service doit utiliser un paramètre indiquant le type de modification à apporter (totale ou incrémentielle). Si une édition incrémentielle est spécifiée, seules les connexions qui ont été modifiées doivent être interrompues, puis rétablies. Si une édition complète est spécifiée, toutes les connexions doivent être interrompues et toutes les ressources prises en charge libérées, puis de nouveau attribuées, avant de tenter de rétablir les connexions. Le service Change_Complete doit être pris en charge au niveau de la classe (instance 0) uniquement.

Si l'attribut d'instance facultatif 0x0C est pris en charge et si la valeur de l'attribut d'instance 0x0C est FALSE ou si l'attribut d'instance facultatif 0x0C n'est pas pris en charge, une ouverture de connexion doit être tentée. Si l'instance d'attribut facultatif 0x0C est prise en charge et si la valeur de l'attribut d'instance 0x0C est TRUE, aucune tentative d'ouverture de connexion ne doit être réalisée.

Le paramètre de demande est défini dans le Tableau 106.

**Tableau 106 – Paramètre du service Change_Complete**

| Paramètre | Type de données | Description |
|---|---|---|
| Change type | UINT | 0 = Complet<br>1 = Incrémentiel<br>Toutes les autres valeurs réservées. |

Ce service doit prendre en charge les codes d'erreur présentés dans le Tableau 107, en complément des codes d'erreur répertoriés dans la CEI 61158-6-2, 4.1.11.2.1.

**Tableau 107 – Codes d'erreur du service Change_Complete**

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| 0x02 | Aucun | Ressources insuffisantes | Indique que la mémoire n'est pas suffisante sur l'appareil hôte pour prendre en charge la configuration |

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| | | | spécifiée |
| 0x0C | Aucun | Conflit d'état d'objet | Une session d'édition n'est pas active |
| 0x10 | Aucun | Conflit d'état de l'appareil | L'appareil est dans un état qui empêche une session d'édition de se terminer |

#### 7.8.7.9 Audit_Changes

Ce service doit vérifier si la mémoire est suffisante sur l'appareil hôte pour prendre en charge une configuration proposée. A l'instar du service Change_Complete, ce service doit utiliser un paramètre indiquant le type de modification à apporter (totale ou incrémentielle). Le service Audit_Changes doit être pris en charge au niveau de la classe (instance 0) uniquement. Ce service permet à un client de configuration de déterminer si toutes les modifications en attente vont aboutir avant de réellement valider les modifications avec le service Change_Complete.

Le paramètre de demande est défini dans le Tableau 108.

**Tableau 108 – Paramètre du service Audit_Changes**

| Paramètre | Type de données | Description |
|---|---|---|
| Change type | UINT | 0 = Complet<br>1 = Incrémentiel<br>Toutes les autres valeurs réservées |

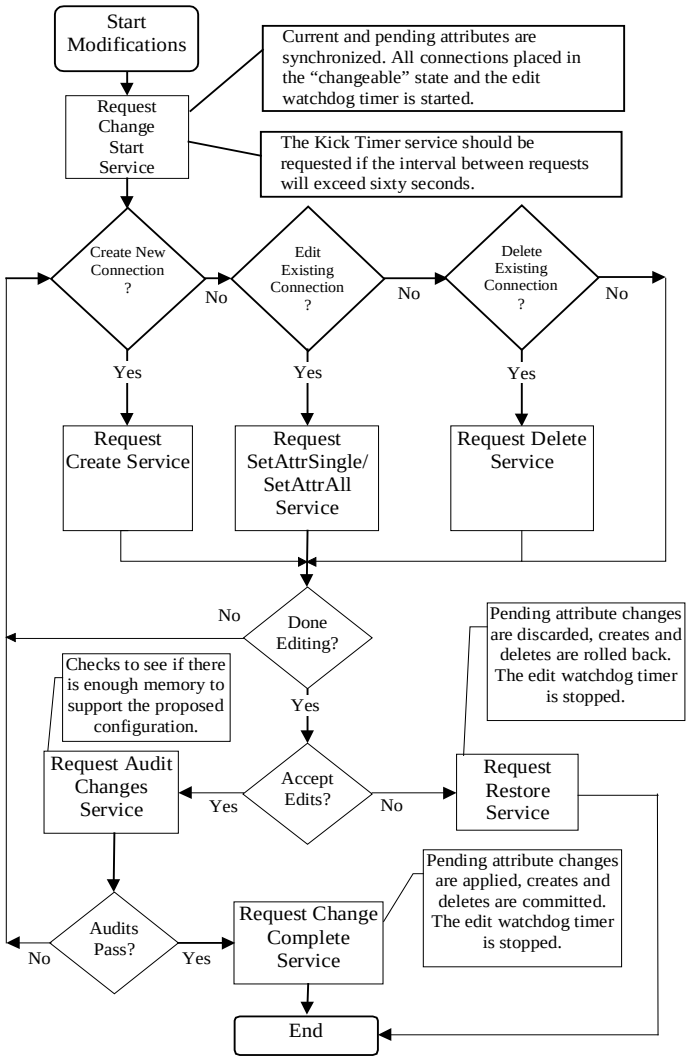
Ce service doit prendre en charge les codes d'erreur présentés dans le Tableau 109, en complément des codes d'erreur répertoriés dans la CEI 61158-6-2, 4.1.11.2.1.

**Tableau 109 – Codes d'erreur du service Audit_Changes**

| Etat général | Etat étendu | Nom de l'erreur | Description |
|---|---|---|---|
| 0x02 | Aucun | Ressources insuffisantes | Indique que la mémoire n'est pas suffisante sur l'appareil hôte pour prendre en charge la configuration spécifiée |
| 0x0C | Aucun | Conflit d'état d'objet | Une session d'édition n'est pas active |
| 0x10 | Aucun | Conflit d'état de l'appareil | L'appareil est dans un état qui empêche un audit de se terminer |

### 7.8.8 Comportement

La Figure 25 récapitule le processus de création, d'édition et de suppression des connexions.




**Légende**

| Anglais | Français |
|---|---|
| Start modifications | Début des modifications |
| Current and pending attributes are synchronized. All connections placed in the « changeable » state and the edit watchdog time ris started | Les attributs en cours et en attente sont synchronisés. Toutes les connexions sont placées à l'état "modifiable" et le temporisateur d'édition est démarré |
| Request change start service | Demande de service change_start |
| The kick timer service should be requested if the interval between requests will exceed sixty seconds | Il convient de demander le service kick_timer si l'intervalle entre les demandes dépasse soixante secondes |
| Create new connection? | Création d'une nouvelle connexion? |
| No | Non |
| Edit existing connection? | Edition de la connexion existante? |
| Delete existing connection? | Suppression de la connexion existante? |
| Yes | Oui |
| Request create service | Demande service Create |
| Request SetAtrrSingle/SetAttrAll service | Demande service SetAttrSingle/SetAttrAll |
| Request delete service | Demande service Delete |
| Done editing? | Edition réalisée? |
| Pending attribute changes are discarded, creates | Les modifications d'attribut en cours sont |

| Anglais | Français |
|---|---|
| and deletes are rolled back. The edit watchdog timer is stopped | écartées, les services Create et Delete sont repris. Le temporisateur d'édition est arrêté |
| Checks to see if there is enough memory to support the proposed configuration | Vérifications de la présence de suffisamment de mémoire pour prendre en charge la configuration proposée |
| Request audit changes service | Demande service Audit_changes |
| Accept edits? | Accepter les éditions? |
| Request restore service | Demande service Restore |
| Pending attribute changes are applied, creates and deletes are committed. The edit watchdog timer is stopped | Les modifications d'attribut en cours sont appliquées, les services Create et Delete sont engagés. Le temporisateur d'édition est arrêté |
| Request Change Complete Service | Demande service Change_Complete |
| Audits pass? | Audits réussis? |
| End | fin |

**Figure 25 – Diagramme d'édition de l'objet de configuration de connexion**

## 7.9 Objet DLR

### 7.9.1 Présentation

L'objet DLR (Device Level Ring) fournit l'interface informative de configuration et d'état de niveau d'application CPF 2 pour le protocole DLR. Le protocole DLR est intégralement spécifié à l'Article 10.

L'objet DLR doit être mis en œuvre dans les appareils CP 2/2 à plusieurs ports prenant en charge le protocole DLR.

Les appareils ne doivent pas mettre en œuvre plus d'une instance de l'objet DLR.

NOTE La prise en charge du protocole DLR sur plusieurs paires de ports fera l'objet d'une amélioration ultérieure pour le moment indéfinie.

### 7.9.2 Historique de révision

Le Tableau 110 montre l'historique de révision pour l'objet DLR.

**Tableau 110 – Historique de révision**

| Revision | Description |
|---|---|
| 1 | Révision initiale de la définition de cet objet |
| 2 | L'attribut d'instance 10 a été modifié pour requis, l'attribut d'instance 12 a été ajouté et les modifications appropriées ont été faites dans la réponse Get_Attribute_All<br>Ajout du service spécifique à l'objet Restart_Sign_On |
| 3 | Addition de l'attribut conditionnel 13 pour la configuration d'une passerelle redondante<br>Addition de l'attribut conditionnel 14 pour l'état de la passerelle redondante<br>Addition de l'attribut conditionnel 15 pour l'adresse de la passerelle active<br>Addition de l'attribut conditionnel 16 pour la priorité de la passerelle active<br>Addition du bit de capacité de passerelle redondante à l'attribut 12, balises de capacité<br>Addition du bit de capacité de prise en charge de trames Flush_Tables à l'attribut 12, balises de capacité<br>Addition de réponses Get_Attribute_All pour les appareils passerelle redondant<br>La révision 2 est obsolète |

### 7.9.3 Attributs de classe

L'objet DLR doit prendre en charge les attributs de classe spécifiés dans le Tableau 111.

**Tableau 111 – Attributs de classe de l'objet DLR**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| 0x01 | Sous condition[a] | Get | NV | Revision | UINT | Révision de cet objet | Deuxième révision, valeur = 2 |
| 0x01 à 0x07 | Ces attributs de classe sont facultatifs et sont décrits dans la CEI 61158-5-2 | | | | | | |
| [a] Obligatoire si la valeur de révision est supérieure à 1. | | | | | | | |

### 7.9.4 Attributs d'instance

#### 7.9.4.1 Généralités

L'objet DLR doit prendre en charge les attributs d'instance spécifiés dans le Tableau 112.

**Tableau 112 – Attributs d'instance de l'objet DLR**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| 0x01 | Obligatoire | Get | V | Network Topology | USINT | Mode de topologie de réseau en cours | 0 – Linéaire<br>1 – Anneau |
| 0x02 | Obligatoire | Get | V | Network Status | USINT | Etat actuel du réseau | 0 – Normal<br>1 – Défaut d'anneau<br>2 – Boucle imprévue détectée<br>3 – Défaut de réseau partiel<br>4 – Cycle rapide de défaut/restauration |
| 0x03 | Sous condition a | Get | V | Ring Supervisor Status | USINT | Balise d'état active du superviseur d'anneau | Voir 7.9.4.17 |
| 0x04 | Sous condition a | Set | NV | Ring Supervisor Config | STRUCT de | Paramètres de configuration du superviseur d'anneau | |
| | | | | Ring Supervisor Enable | BOOL | Balise d'activation du superviseur d'anneau | TRUE – L'appareil est configuré comme un superviseur d'anneau.<br>FALSE – L'appareil est configuré comme un nœud d'anneau normal<br>Valeur par défaut = FALSE |
| | | | | Ring Supervisor Precedence | USINT | Priorité d'un superviseur d'anneau dans un réseau en comportant plusieurs | Une valeur numérique supérieure indique la priorité supérieure<br>Valeur par défaut = 0 |
| | | | | Beacon Interval | UDINT | Durée de l'intervalle Beacon de l'anneau | Intervalle Beacon, en microsecondes.<br>Valeur par défaut = 400 µs |
| | | | | Beacon Timeout | UDINT | Durée du délai d'attente Beacon de l'anneau | Délai d'attente Beacon, en microsecondes.<br>Valeur par défaut = 1 960 µs |
| | | | | DLR VLAN ID | UINT | ID VLAN à utiliser dans les messages de protocole d'anneau | Plage de valeurs comprise entre 0 et 4 094<br>Valeur par défaut = 0 |
| 0x05 | Sous condition a | Set | V | Ring Faults Count | UINT | Nombre de défauts d'anneau depuis la mise sous tension | |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| 0x06 | Sous condition a | Get | V | Last Active Node on Port 1 | STRUCT de | Dernier nœud actif à la fin de la chaîne par le port 1 du superviseur d'anneau actif pendant le défaut d'anneau | |
| | | | | | UDINT | Adresse IP de l'appareil | Une valeur égale à 0 indique qu'aucune adresse IP n'a été configurée pour l'appareil. La valeur initiale doit être 0 |
| | | | | | USINT[6] | Adresse MAC de l'appareil | Adresse MAC Ethernet |
| 0x07 | Sous condition a | Get | V | Last Active Node on Port 2 | STRUCT de | Dernier nœud actif à la fin de la chaîne par le port 2 du superviseur d'anneau actif pendant le défaut d'anneau | |
| | | | | | UDINT | Adresse IP de l'appareil | Une valeur égale à 0 indique qu'aucune adresse IP n'a été configurée pour l'appareil |
| | | | | | USINT[6] | Adresse MAC de l'appareil | Adresse MAC Ethernet |
| 0x08 | Sous condition a | Get | V | Ring Protocol Participants Count | UINT | Nombre d'appareils dans la liste des participants au protocole d'anneau | |
| 0x09 | Sous condition a | Get | V | Ring Protocol Participants List | MATRICE de | Liste des appareils participant au protocole d'anneau | |
| | | | | | STRUCT de | | |
| | | | | | UDINT | Adresse IP de l'appareil | Une valeur égale à 0 indique qu'aucune adresse IP n'a été configurée pour l'appareil |
| | | | | | USINT[6] | Adresse MAC de l'appareil | Adresse MAC Ethernet |
| 0x0A | Obligatoire | Get | V | Active Supervisor Address | STRUCT de | Adresse IP et/ou MAC du superviseur d'anneau actif | |
| | | | | | UDINT | Adresse IP du superviseur | Une valeur égale à 0 indique qu'aucune adresse IP n'a été configurée pour l'appareil |
| | | | | | USINT[6] | Adresse MAC du superviseur | Adresse MAC Ethernet |
| 0x0B | Sous condition a | Get | V | Active Supervisor Precedence | USINT | Valeur de priorité du superviseur d'anneau actif | |
| 0x0C | Obligatoire | Get | NV | Capability Flags | DWORD | Décrit les capacités DLR de l'appareil | Voir 7.9.4.13 |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| 0x0D | Sous condition[b] | Set | NV | Redundant Gateway Config | STRUCT of | Paramètres de configuration de passerelle redondante | |
| | | | | Redundant Gateway Enable | BOOL | Balise d'activation de passerelle redondante | TRUE indique que l'appareil est configuré en tant que passerelle redondante.<br>FALSE indique que l'appareil n'est pas configuré en tant que passerelle redondante.<br>Valeur par défaut = FALSE |
| | | | | Gateway Precedence | USINT | Priorité d'une passerelle dans un réseau comportant plusieurs passerelles | Une valeur numérique supérieure indique la priorité supérieure.<br>Valeur par défaut = 0 |
| | | | | Advertise Interval | UDINT | Durée de l'intervalle Advertise de la passerelle active | Intervalle Advertise, en microsecondes.<br>Valeur par défaut = 2 000 µs |
| | | | | Advertise Timeout | UDINT | Durée du délai d'attente Advertise de la passerelle active | Délai d'attente Advertise, en microsecondes.<br>Valeur par défaut = 5 000 µs |
| | | | | Learning Update Enable | BOOL | Balise d'activation des mises à jour d'apprentissage | TRUE indique que tous les nœuds DLR enverront une trame Learning_Update après le changement de passerelle.<br>FALSE indique que les nœuds DLR n'enverront pas de trame Learning_Update après le changement de passerelle.<br>Valeur par défaut = TRUE |
| 0x0E | Sous condition[b] | Get | V | Redundant Gateway Status | | | Voir 7.9.4.15 |
| 0x0F | Sous condition[b] | Get | V | Active Gateway Address | STRUCT of | Adresse IP et/ou MAC de l'appareil passerelle actif | |
| | | | | | UDINT | Adresse IP de la passerelle active | Une valeur égale à 0 indique qu'aucune adresse IP n'a été configurée pour cet appareil |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | N V | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| | | | | | USINT[6] | Adresse MAC de la passerelle active | Adresse MAC Ethernet |
| 0x10 | Sous condition[b] | Get | V | Active Gateway Precedence | USINT | Valeur de priorité de la passerelle active | |

[a] Doit être mis en œuvre pour les appareils en mesure de fonctionner comme un superviseur d'anneau. Ne doit pas être mis en œuvre par les appareils non superviseurs.

[b] Doit être mis en œuvre pour les appareils en mesure de fonctionner comme une passerelle redondante. Ne doit pas être mis en œuvre par les appareils passerelle non redondants.

#### 7.9.4.2 Network Topology

L'attribut Network Topology indique le mode de topologie de réseau en cours. Une valeur égale à 0 indique une topologie « linéaire ». Une valeur 1 indique une topologie « en anneau ». La valeur de l'attribut doit correspondre à l'état DLR.

Lorsqu'un appareil superviseur est activé comme un superviseur d'anneau, l'attribut Network Topology doit toujours indiquer « Ring ». Si un appareil superviseur n'est pas activé comme un superviseur d'anneau ou s'il ne s'agit pas d'un appareil superviseur, il doit indiquer « Linear », puis basculer entre les modes « Ring » et « Linear ».

#### 7.9.4.3 Network status

L'attribut Network Status fournit l'état en cours du réseau en fonction de la vue de l'appareil du réseau (voir le comportement DLR en 10.5). Le Tableau 113 spécifie les valeurs possibles pour l'attribut Network Status.

**Tableau 113 – Valeurs de l'attribut Network Status**

| Valeur de l'attribut Network Status | Description |
|---|---|
| 0 | Fonctionnement normal dans les modes de topologie de réseau « en anneau » et « linéaire » |
| 1 | Défaut d'anneau. Un défaut d'anneau a été détecté. Valide uniquement s'il s'agit d'une topologie de réseau en anneau |
| 2 | Boucle imprévue détectée. Une boucle a été détectée dans le réseau. Valide uniquement s'il s'agit d'une topologie de réseau linéaire |
| 3 | Défaut de réseau partiel. Un défaut de réseau a été détecté dans une seule direction. Valide uniquement s'il s'agit d'une topologie de réseau en anneau et si le nœud est le superviseur d'anneau actif |
| 4 | Cycle rapide de défaut/restauration. Une série de cycles rapides de défaut/restauration d'anneau a été détectée, conformément aux critères définis en 10.5.3.5. Analogue à l'état de défaut de réseau partiel, le superviseur reste dans un état et le transfert est bloqué sur ses ports en anneau. La condition doit être clairement explicite grâce au service Clear Rapid Faults |

#### 7.9.4.4 Ring supervisor status

L'attribut Ring Supervisor Status indique l'état de l'appareil comme un superviseur d'anneau. Le Tableau 114 spécifie les valeurs possibles de l'attribut Ring Supervisor Status.

**Tableau 114 – Valeur de l'attribut Ring Supervisor Status**

| Valeur de l'attribut Ring Supervisor Status | Description |
|---|---|
| 0 | L'appareil fonctionne comme un superviseur de sauvegarde |
| 1 | L'appareil fonctionne comme le superviseur d'anneau actif |
| 2 | L'appareil fonctionne comme un nœud d'anneau normal (superviseur non activé) |
| 3 | L'appareil fonctionne dans une topologie non DLR (superviseur désactivé, aucun autre superviseur n'étant présent) |
| 4 | L'appareil ne peut pas prendre en charge les paramètres d'anneau en cours de fonctionnement (Beacon Interval et/ou Beacon Timeout) |

#### 7.9.4.5 Ring supervisor config

##### 7.9.4.5.1 Généralités

L'attribut Ring Supervisor Config contient les paramètres de configuration nécessaires au fonctionnement en anneau: Supervisor Precedence, Beacon Interval, Beacon Timeout, VLAN ID, Supervisor Enable/Disable.

Si l'appareil est le superviseur actif, les modifications apportées à l'attribut doivent être appliquées immédiatement. Si les paramètres Supervisor Precedence, Beacon Interval, Beacon Timeout ou VLAN ID ont été modifiés sur le superviseur d'anneau actif, ce dernier ne doit plus envoyer de trames Beacon pendant les deux périodes de délai Beacon, puis doit reprendre les envois à l'aide de ces nouveaux paramètres.

Les superviseurs d'anneau de sauvegarde doivent obtenir les valeurs des paramètres Beacon Interval, Beacon Timeout et VLAN ID auprès de la trame Beacon envoyée par le superviseur d'anneau actif, et les stocker dans la mémoire non volatile. Si les valeurs obtenues ne peuvent pas être prises en charge par l'appareil (si le paramètre Beacon Interval est trop petit, par exemple), l'appareil doit définir l'attribut Ring Supervisor Status comme indiqué dans la description de cet attribut, doit signaler un défaut irrécupérable mineur via l'attribut Identity object Status (5) et ne doit pas prendre la suite comme un superviseur actif après la reconfiguration en anneau.

Si l'attribut Ring Supervisor Config est modifié sur un superviseur de sauvegarde, le comportement dépend de la nouvelle valeur de priorité du superviseur de sauvegarde par rapport à celle du superviseur actif:

- la nouvelle valeur de priorité de sauvegarde est supérieure à la priorité du superviseur d'anneau actif en cours ou présente une priorité égale avec un numéro d'adresse MAC supérieur à celui du superviseur actif: la sauvegarde doit immédiatement commencer à envoyer les trames Beacon avec les nouveaux paramètres;
- la nouvelle valeur de priorité de sauvegarde est inférieure à la priorité du superviseur actif ou présente une priorité égale avec un numéro d'adresse MAC inférieur à celui du superviseur actif: la modification des paramètres Beacon Interval, Beacon Timeout et VLAN ID doit être ignorée.

Si des valeurs non valides sont affectées à l'attribut Ring Supervisor Config, le service Set doit renvoyer un code d'erreur 0x09 (valeur d'attribut non valide), que l'appareil soit un superviseur actif ou de sauvegarde.

##### 7.9.4.5.2 Ring supervisor enable

L'élément Ring Supervisor Enable permet d'activer ou de désactiver la fonction de superviseur d'anneau dans un appareil compatible. Une valeur TRUE active la fonction de superviseur. Une valeur FALSE désactive la fonction de superviseur. La valeur par défaut est FALSE.

#### 7.9.4.5.3 Ring supervisor precedence

L'élément Ring Supervisor Precedence contient la valeur de priorité attribuée par l'utilisateur donnée au superviseur d'anneau. Si plusieurs superviseurs d'anneau sont activés, la valeur de priorité permet à l'utilisateur de configurer l'ordre dans lequel les superviseurs configurés sélectionnent le superviseur actif.

La valeur de priorité du superviseur d'anneau doit être choisie dans la plage comprise entre 0 et 255, les valeurs numériques les plus élevées indiquant les priorités les plus élevées. La valeur par défaut doit être 0.

Si plusieurs superviseurs sont activés, le superviseur à la priorité la plus élevée devient le superviseur d'anneau actif, conformément à 10.5.2. Si plusieurs superviseurs ont la même priorité, le superviseur avec le numéro d'adresse MAC le plus élevé devient le superviseur actif.

#### 7.9.4.5.4 Beacon interval

L'élément Beacon Interval contient l'intervalle, en microsecondes, que le superviseur d'anneau doit utiliser pour générer des trames Beacon. Conformément à la spécification de protocole DLR de l'Article 10, la valeur par défaut doit être 400 µs. Les superviseurs doivent prendre en charge une plage comprise entre 400 µs et 100 ms. Ils peuvent prendre en charge un intervalle Beacon inférieur à 400 µs, mais ce n'est pas requis. L'intervalle Beacon minimal absolu est de 100 µs.

#### 7.9.4.5.5 Beacon timeout

L'élément Beacon Timeout contient le nombre de microsecondes pendant lesquelles le superviseur d'anneau doit attendre une trame Beacon avant de déclarer un délai d'attente Beacon.

La valeur par défaut doit être de 1 960 µs, selon une taille nominale du réseau de 50 nœuds et 100 Mbit/s et un fonctionnement en mode simultané (voir les calculs de performances en 10.11). L'utilisateur peut modifier la valeur Beacon Timeout pour d'autres circonstances exceptionnelles inhérentes au réseau (des réseaux très étendus ou de très petits réseaux à performances élevées, par exemple).

L'élément Beacon Timeout doit représenter au moins 2 fois la valeur de l'intervalle Beacon. Si l'intervalle Beacon a été modifié et que le délai d'attente Beacon est inférieur à 2 fois l'intervalle Beacon, le superviseur doit ajuster le délai d'attente Beacon à 2 fois l'intervalle Beacon.

Les superviseurs doivent prendre en charge une plage comprise entre 800 µs et 500 ms. Ils peuvent prendre en charge un délai d'attente Beacon inférieur à 800 µs, mais ce n'est pas requis. Le délai d'attente Beacon minimal absolu est de 200 µs.

#### 7.9.4.5.6 DLR VLAN ID

Le DLR VLAN ID contient le VLAN ID à utiliser dans les trames de protocole DLR. Le DLR VLAN ID doit être utilisé pour toutes les trames de protocole DLR générées par l'appareil, si ce dernier fonctionne comme le superviseur d'anneau actif. Les appareils qui ne sont pas des superviseurs d'anneau actifs doivent utiliser le VLAN ID obtenu à partir des trames du superviseur actif (voir l'Article 10 pour plus de détails).

La valeur VLAN ID doit être comprise entre 0 et 4 094. La valeur par défaut doit être 0 (n'indiquant aucun VLAN).

#### 7.9.4.6 Ring Faults Count

L'attribut Ring Faults Count contient le nombre de défauts d'anneau que l'appareil a détecté depuis le démarrage, qu'il s'agisse d'un superviseur actif ou de sauvegarde. Si la valeur FALSE est attribuée à Ring Supervisor Enable, la valeur 0 doit être affectée à l'attribut Ring Faults Count. L'attribut Ring Faults Count revient à 0 lorsqu'il atteint sa valeur maximale.

L'attribut peut également être remis à 0 par l'intermédiaire du service Set_Attribute_Single. Les valeurs différentes de 0 doivent donner une réponse d'erreur.

#### 7.9.4.7 Last active node on port 1

L'attribut Last Active Node on Port 1 contient l'adresse IP et/ou l'adresse MAC Ethernet du dernier nœud qu'il est possible d'atteindre sur le port 1 d'un superviseur d'anneau actif. La valeur de l'attribut est obtenue par l'intermédiaire des trames Link_Status/Neighbor Status (voir 10.9.6).

Lors de la transition vers FAULT_STATE, cet attribut doit rester vide tant que le superviseur ne reçoit pas d'informations d'état de la liaison/du voisin.

Lors de la transition de FAULT_STATE vers NORMAL_STATE, la valeur de l'attribut doit être conservée afin de faciliter le diagnostic du défaut d'anneau précédent.

Les valeurs initiales de l'adresse IP et de l'adresse MAC Ethernet doivent être égales à 0. Si l'appareil n'est pas activé en tant que superviseur d'anneau ou s'il fonctionne en tant que superviseur de sauvegarde, l'adresse IP et l'adresse MAC Ethernet doivent être égales à 0.

#### 7.9.4.8 Last active node on port 2

L'attribut Last Active Node on Port 2 contient l'adresse IP et/ou l'adresse MAC Ethernet du dernier nœud qu'il est possible d'atteindre sur le port 2 d'un superviseur d'anneau actif. La valeur de l'attribut est obtenue par l'intermédiaire des trames Link_Status/Neighbor Status (voir 10.9.6).

Lors de la transition vers FAULT_STATE, cet attribut doit rester vide tant que le superviseur ne reçoit pas d'informations d'état de la liaison/du voisin.

Lors de la transition de FAULT_STATE vers NORMAL_STATE, la valeur de l'attribut doit être conservée afin de faciliter le diagnostic du défaut d'anneau précédent.

Les valeurs initiales de l'adresse IP et de l'adresse MAC Ethernet doivent être égales à 0. Si l'appareil n'est pas activé en tant que superviseur d'anneau ou s'il fonctionne en tant que superviseur de sauvegarde, l'adresse IP et l'adresse MAC Ethernet doivent être égales à 0.

#### 7.9.4.9 Ring protocol participants count

Cet attribut contient le nombre de membres présents dans l'attribut Ring Protocol Participants List. Le comptage et la liste doivent être rassemblés par le superviseur d'anneau actif grâce à la trame Sign_On (voir 10.9.9).

Si l'appareil n'est pas le superviseur actif, la valeur de l'attribut doit être égale à 0.

#### 7.9.4.10 Ring protocol participants list

L'attribut Ring Protocol Participants List contient la liste des nœuds d'anneau participant au protocole d'anneau. La liste des participants est rassemblée par le superviseur d'anneau actif par l'intermédiaire de la trame Sign_On (voir 10.9.9).

Etant donné que l'attribut Ring Protocol Participants List peut être volumineux, en fonction du nombre de nœuds participant dans l'anneau, il doit être accessible avec le service Get_Member.

Les clients peuvent choisir d'utiliser le service Get_Attribute_Single pour lire l'attribut Ring Protocol Participants List. Si la liste des participants est trop volumineuse, l'objet DLR doit renvoyer un code d'erreur 0x11 (données de réponse trop volumineuses).

Si l'appareil n'est pas le superviseur actif, le code d'état 0x0C (conflit d'état d'objet) doit être retourné.

Si le nombre de participants reçu comme résultat du processus Sign_On dépasse la capacité de l'attribut Ring Protocol Participants List du superviseur, la dernière entrée dans la liste doit être toutes les valeurs (0xFFFFFFFFFFFFFFFFFFFF) de 0xFF.

#### 7.9.4.11 Active supervisor address

Cet attribut contient l'adresse IP et/ou l'adresse MAC Ethernet du superviseur d'anneau actif. Les valeurs initiales de l'adresse IP et de l'adresse MAC Ethernet doivent être égales à 0, tant que le superviseur d'anneau actif n'est pas déterminé.

#### 7.9.4.12 Active supervisor precedence

Cet attribut contient la valeur de priorité du superviseur d'anneau actif. La valeur initiale doit être égale à 0, tant que le superviseur d'anneau actif n'est pas déterminé.

#### 7.9.4.13 Capability flags

Les balises de capacité décrivent les capacités DLR de l'appareil, comme cela est spécifié le Tableau 115.

**Tableau 115 – Balises de capacité**

| Bit(s): | Nom | Définition |
|---|---|---|
| 0 | Nœud d'anneau Announce[a] | Défini si la mise en œuvre du nœud d'anneau de l'appareil est fondée sur le traitement des trames Announce (voir 10.4) |
| 1 | Nœud d'anneau Beacon[a] | Défini si la mise en œuvre du nœud d'anneau de l'appareil est fondée sur le traitement des trames Beacon (voir 10.4) |
| 2-4 | Réservé | Doit être défini à zéro. |
| 5 | Supervisor Capable | Défini si l'appareil est capable d'exécuter une fonction de superviseur. |
| 6 | Redundant Gateway Capable | Défini si l'appareil est capable d'exécuter la fonction de passerelle redondante. |
| 7 | Flush_Table frame Capable | Défini si l'appareil est capable de prendre en charge la trame Flush_Tables. |
| 8-31 | Réservé | Doit être défini à zéro. |

[a] Les bits 0 et 1 sont mutuellement exclusifs. Exactement un seul de ces bits doit être défini dans la valeur d'attribut dont fait état un appareil.

#### 7.9.4.14 Redundant Gateway Config

##### 7.9.4.14.1 Généralités

L'attribut Redundant Gateway Config contient les paramètres de configuration nécessaires au fonctionnement de la passerelle redondante: Gateway enable/disable, Gateway Precedence, Advertise Interval, Advertise Timeout et Learning Update enable/disable.

Si l'appareil est la passerelle active, les modifications apportées à l'attribut doivent être appliquées immédiatement. Lorsque les paramètres Gateway Precedence, Advertise Interval, Advertise Timeout et Learning Update enable/disable sont modifiés sur la passerelle active, la passerelle active cesse d'envoyer des trames Advertise pendant 1,5 fois la période Advertise Timeout et doit alors reprendre l'envoi de trames Advertise en utilisant les nouveaux paramètres.

Les appareils passerelle de sauvegarde doivent obtenir les valeurs des paramètres Advertise Interval, Advertise Timeout et Learning Update enable/disable auprès de la trame Advertise envoyée par la passerelle active, et les stocker dans la mémoire non volatile. Si les valeurs obtenues ne peuvent pas être prises en charge par l'appareil (si le paramètre Advertise Interval est trop petit, par exemple), l'appareil doit définir l'attribut Redundant Gateway Status comme indiqué dans la description de cet attribut, doit signaler un défaut irrécupérable mineur via l'attribut Identity object Status (5), et ne doit pas prendre la suite comme une passerelle active après la reconfiguration en passerelle.

Si l'attribut Redundant Gateway Config est modifié sur une passerelle de sauvegarde, le comportement dépend de la nouvelle valeur de priorité de la passerelle de sauvegarde par rapport à celle de la passerelle active.

- La nouvelle valeur de priorité de sauvegarde est supérieure à la priorité de la passerelle active en cours ou présente une priorité égale avec un numéro d'adresse MAC supérieur à celui de la passerelle active: la sauvegarde doit immédiatement commencer à envoyer les trames Advertise avec les nouveaux paramètres (voir 10.10.4).
- La nouvelle valeur de priorité de sauvegarde est inférieure à la priorité de la passerelle active ou présente une priorité égale avec un numéro d'adresse MAC inférieur à celui de la passerelle active: Les paramètres Advertise Interval, Advertise Timeout et Learning Update enable/disable doivent être ignorés.

Si des valeurs non valides sont affectées à l'attribut Redundant Gateway Config, le service Set doit renvoyer un code d'erreur 0x09 (valeur d'attribut non valide), que l'appareil soit une passerelle active ou de sauvegarde.

#### 7.9.4.14.2 Redundant Gateway Enable

L'élément Redundant Gateway Enable active ou désactive la fonction de passerelle dans un appareil compatible avec une passerelle redondante. Une valeur TRUE active la fonction de passerelle. Une valeur FALSE désactive la fonction de passerelle. La valeur par défaut est FALSE.

#### 7.9.4.14.3 Gateway Precedence

L'élément Gateway Precedence contient la valeur de priorité attribuée à une passerelle par l'utilisateur. Si plusieurs passerelles sont activées, la valeur de priorité permet à l'utilisateur de configurer l'ordre dans lequel les passerelles configurées sélectionnent la passerelle active.

La valeur de priorité de la passerelle doit être choisie dans la plage comprise entre 0 et 255, les valeurs numériques les plus élevées indiquant les priorités les plus élevées. La valeur par défaut doit être 0.

Si plusieurs passerelles sont activées, la passerelle à la priorité la plus élevée devient la passerelle active, conformément à l'Article 10. Si plusieurs passerelles ont la même priorité, la passerelle avec le numéro d'adresse MAC le plus élevé devient la passerelle active.

#### 7.9.4.14.4 Advertise Interval

L'élément Advertise Interval contient l'intervalle, en microsecondes, que la passerelle active doit utiliser pour générer des trames Advertise. Conformément à la spécification de protocole

DLR de l'Article 10, la valeur par défaut doit être 2 000 µs. Les passerelles doivent prendre en charge une plage comprise entre 1 000 µs et 100 ms. Les passerelles peuvent prendre en charge un intervalle Advertise inférieur à 1 000 µs, mais ce n'est pas requis. L'intervalle Advertise minimal absolu est de 200 µs.

#### 7.9.4.14.5 Advertise Timeout

L'élément Advertise Timeout contient le nombre de microsecondes pendant lesquelles une passerelle de sauvegarde doit attendre une trame Advertise avant de déclarer un délai d'attente Advertise.

La valeur par défaut doit être de 5 000 µs, selon une taille nominale du réseau de 50 nœuds et 100 Mbit/s et un fonctionnement en mode simultané (voir les Calculs de performances de l'Article 10). L'utilisateur peut modifier la valeur Advertise Timeout pour des circonstances exceptionnelles inhérentes au réseau (des réseaux très étendus ou de très petits réseaux à performances élevées, par exemple).

L'élément Advertise Timeout doit représenter au moins 2,5 fois la valeur de l'intervalle Advertise. Si l'intervalle Advertise a été modifié et que le délai d'attente Advertise est inférieur à 2,5 fois l'intervalle Advertise, la passerelle doit ajuster le délai d'attente Advertise à 2,5 fois l'intervalle Advertise.

Les passerelles doivent prendre en charge une plage comprise entre 2 500 µs et 500 ms. Les passerelles peuvent prendre en charge un délai d'attente Advertise inférieur à 2 500 µs, mais ce n'est pas requis. Le délai d'attente Advertise minimal absolu est de 500 µs. Le délai d'attente Advertise minimal absolu est de 500 µs.

#### 7.9.4.14.6 Learning Update Enable

L'élément Learning Update Enable active ou désactive la transmission de trames Learning_Update par les nœuds DLR lorsqu'ils reçoivent une trame Flush_Tables de la passerelle active. Ce paramètre est encodé par la passerelle active dans la trame Flush_Tables et envoyé aux nœuds DLR. Les trames Learning_Update provenant des appareils DLR accélèrent l'apprentissage de la nouvelle topologie de réseau par l'ensemble des commutateurs non DLR situés en dehors du réseau DLR après le basculement sur la passerelle active. Une valeur TRUE active la fonction de mise à jour de l'apprentissage. Une valeur FALSE désactive la fonction de mise à jour de l'apprentissage. La valeur par défaut est TRUE.

### 7.9.4.15 Redundant Gateway Status

L'attribut Redundant Gateway Status indique l'état de l'appareil comme une passerelle. Le Tableau 116 montre les valeurs possibles.

**Tableau 116 – Valeur de l'attribut Redundant Gateway Status**

| Valeur Redundant Gateway Status | Description |
|---|---|
| 0 | Indique que l'appareil fonctionne comme un nœud DLR non passerelle (passerelle non activée) |
| 1 | Indique que l'appareil fonctionne comme une passerelle de sauvegarde |
| 2 | Indique que l'appareil fonctionne comme la passerelle active |
| 3 | Indique un état de défaut de la passerelle causé par une perte de communication sur le port de liaison montante |
| 4 | Indique que l'appareil ne peut pas prendre en charge les paramètres de passerelle en cours de fonctionnement (Advertise Interval et/ou Advertise Timeout) |
| 5 | Indique un état de défaut de la passerelle causé par un défaut partiel du réseau (voir 10.8.4.8) |

#### 7.9.4.16 Active Gateway Address

Cet attribut contient l'adresse IP et/ou l'adresse MAC Ethernet de l'appareil passerelle actif. Les valeurs initiales de l'adresse IP et de l'adresse MAC Ethernet doivent être égales à 0, tant que la passerelle active n'est pas déterminée.

#### 7.9.4.17 Active Gateway Precedence

Cet attribut contient la valeur de priorité l'appareil passerelle actif. La valeur initiale doit être égale à 0, tant que la passerelle active n'est pas déterminée.

### 7.9.5 Services communs

#### 7.9.5.1 Généralités

Les services communs mis en œuvre par l'objet DLR sont spécifiés dans le Tableau 117.

**Tableau 117 – Services communs de l'objet DLR**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x01 | Facultatif | Obligatoire | Get_Attribute_All | Renvoie plusieurs attributs dans l'ordre numérique |
| 0x0E | Facultatif | Obligatoire | Get_Attribute_Single | Renvoie un seul attribut |
| 0x10 | n/a | Sous condition [a] | Set_Attribute_Single | Modifie un seul attribut |
| 0x18 | n/a | Sous condition [b] | Get_Member | Renvoie les membres de l'attribut |

[a] Ce service doit être mis en œuvre pour les appareils en mesure de fonctionner comme un superviseur d'anneau et/ou une passerelle redondante.

[b] Requis pour accéder à l'attribut Ring Protocol Participants List. Le protocole étendu aux services de membre doit être pris en charge pour plusieurs membres séquentiels.

#### 7.9.5.2 Réponse Get_Attribute_All

##### 7.9.5.2.1 Niveau de classe

Au niveau de classe, la réponse Get_Attributes_All doit contenir les attributs de classe dans l'ordre numérique, jusqu'au dernier attribut mis en œuvre. Tous les attributs non mis en œuvre dans la réponse doivent utiliser les valeurs d'attribut par défaut.

##### 7.9.5.2.2 Niveau d'instance

Au niveau d'instance, la réponse Get_Attributes_All varie selon la révision de l'objet et les capacités de l'appareil. Le format de réponse peut être différentié par la longueur de la réponse.

La longueur de la réponse d'un appareil non superviseur de Révision d'objet 1 s'élève à 2 octets. Le format de réponse est spécifié dans le Tableau 118.

**Tableau 118 – Réponse Get_Attribute_All – appareil non superviseur de Révision d'objet 1**

| ID attribut | Type de données | Nom |
|---|---|---|
| 0x01 | USINT | Network Topology |
| 0x02 | USINT | Network Status |

La longueur de la réponse d'un appareil superviseur de Révision d'objet 1 s'élève à 50 octets. Le format de réponse est spécifié dans le Tableau 119.

**Tableau 119 – Réponse Get_Attribute_All– appareil superviseur de Révision d'objet 1**

| ID attribut | Type de données | Nom |
|---|---|---|
| 0x01 | USINT | Network Topology |
| 0x02 | USINT | Network Status |
| 0x03 | USINT | Ring Supervisor Status |
| 0x04 | STRUCT de | Ring Supervisor Config: |
| | BOOL | Ring Supervisor Enable |
| | USINT | Ring Supervisor Precedence |
| | UDINT | Beacon Interval |
| | UDINT | Beacon Timeout |
| | UINT | DLR VLAN ID |
| 0x05 | UINT | Ring Faults Count |
| 0x06 | STRUCT de | Last Active Node on Port 1 |
| | UDINT | Device IP Address |
| | USINT[6] | Device MAC Address |
| 0x07 | STRUCT de | Last Active Node on Port 2 |
| | UDINT | Device IP Address |
| | USINT[6] | Device MAC Address |
| 0x08 | UINT | Ring Protocol Participants Count |
| 0x0A | STRUCT de | Active Supervisor Address |
| | UDINT | Device IP Address |
| | USINT[6] | Device MAC Address |
| 0x0B | USINT | Active Supervisor Precedence |

La longueur de la réponse d'un appareil non superviseur de Révision d'objet 2 s'élève à 16 octets. Le format de réponse est spécifié dans le Tableau 120.

**Tableau 120 – Réponse Get_Attribute_All – appareil non superviseur de Révision d'objet 2**

| ID attribut | Type de données | Nom |
|---|---|---|
| 0x01 | USINT | Network Topology |
| 0x02 | USINT | Network Status |
| 0x0A | STRUCT de | Active Supervisor Address |
| | UDINT | Device IP Address |
| | USINT[6] | Device MAC Address |
| 0x0C | DWORD | Capability Flags |

Dans tous les autres cas, la réponse Get_Attribute_All (voir Tableau 121) doit contenir les attributs d'instance 0x01 à 0x08 et 0x0A à 0x10, jusqu'au dernier attribut mis en œuvre. Tous les attributs non mis en œuvre dans la réponse doivent utiliser la valeur par défaut indiquée.

La longueur de la réponse d'un appareil superviseur de Révision d'objet 2 s'élève à 54 octets.

La longueur de la réponse d'un appareil non passerelle, non superviseur de Révision d'objet 3 s'élève à 54 octets.

La longueur de la réponse d'un appareil non passerelle, superviseur de Révision d'objet 3 s'élève à 54 octets.

La longueur de la réponse d'un appareil passerelle de Révision d'objet 3 s'élève à 77 octets.

**Tableau 121 – Réponse Get_Attribute_All – tous les autres cas**

| ID attribut | Type de données | Nom | Valeur par défaut (si aucune mise en œuvre) |
|---|---|---|---|
| 0x01 | USINT | Network Topology | |
| 0x02 | USINT | Network Status | |
| 0x03 | USINT | Ring Supervisor Status | 0xFF – Non applicable |
| 0x04 | STRUCT de | Ring Supervisor Config | |
| | BOOL | Ring Supervisor Enable | 0 |
| | USINT | Ring Supervisor Precedence | 0 |
| | UDINT | Beacon Interval | 0 |
| | UDINT | Beacon Timeout | 0 |
| | UINT | DLR VLAN ID | 0 |
| 0x05 | UINT | Ring Faults Count | 0 |
| 0x06 | STRUCT de | Last Active Node on Port 1 | |
| | UDINT | Device IP Address | 0 |
| | USINT[6] | Device MAC Address | Toutes les valeurs nulles |
| 0x07 | STRUCT de | Last Active Node on Port 2 | |
| | UDINT | Device IP Address | 0 |
| | USINT[6] | Device MAC Address | Toutes les valeurs nulles |
| 0x08 | UINT | Ring Protocol Participants Count | 0xFFFF – Non applicable |
| 0x0A | STRUCT de | Active Supervisor Address | |
| | UDINT | Device IP Address | |
| | USINT[6] | Device MAC Address | |
| 0x0B | USINT | Active Supervisor Precedence | 0 |
| 0x0C | DWORD | Capability Flags | |
| 0x0D | STRUCT de | Redundant Gateway Configuration | |
| | BOOL | Redundant Gateway Enable | |
| | USINT | Gateway Precedence | |
| | UDINT | Advertise Interval | |
| | UDINT | Advertise Timeout | |
| | BOOL | Learning Update Enable | |
| 0x0E | USINT | Redundant Gateway Status | |

| ID attribut | Type de données | Nom | Valeur par défaut (si aucune mise en œuvre) |
|---|---|---|---|
| 0x0F | STRUCT de | Active Gateway Address | |
| | UDINT | Device IP Address | |
| | USINT[6] | Device MAC Address | |
| 0x10 | USINT | Active Gateway Precedence | |

### 7.9.6 Services spécifiques à la classe

#### 7.9.6.1 Généralités

L'objet DLR doit prendre en charge les services spécifiques à la classe spécifiés dans le Tableau 122.

**Tableau 122 – Services spécifiques à la classe de l'objet DLR**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x4B | n/a | Sous condition a | Verify_Fault_Location | Le superviseur d'anneau vérifie l'emplacement de défaut en adressant un message de protocole d'anneau Locate_Fault aux nœuds d'anneau et en mettant à jour les attributs Last Active Node 1 et Last Active Node 2 |
| 0x4C | n/a | Sous condition a | Clear_Rapid_Faults | Annule la condition Cycle rapide de défaut/restauration dans le superviseur d'anneau, permettant au superviseur de reprendre un fonctionnement normal |
| 0x4D | n/a | Sous condition a | Restart_Sign_On | Redémarre le processus Sign_On et actualise la liste des participants DLR |
| 0x04E | n/a | Sous condition b | Clear_Gateway_Partial_Fault | Annule la condition de défaut de réseau partiel dans l'appareil passerelle, permettant à la passerelle de reprendre un fonctionnement normal |

a Ce service doit être mis en œuvre pour les appareils en mesure de fonctionner comme un superviseur d'anneau.

b Ce service doit être mis en œuvre pour les appareils en mesure de fonctionner comme une passerelle redondante.

#### 7.9.6.2 Verify_Fault_Location

Le service Verify_Fault_Location doit permettre à un superviseur d'anneau actif de vérifier un emplacement de défaut d'anneau en retransmettant la trame Locate_Fault aux nœuds d'anneau (voir 10.9.7). Les attributs Last Active Node 1 et Last Active Node 2 doivent être mis à jour en fonction de la réponse apportée à la trame Locate_Fault.

Aucun paramètre n'est défini pour la demande ou la réponse Verify_Fault_Location.

Si le service Verify _Fault_Location est reçu alors que le superviseur n'est pas activé, qu'il s'agisse du superviseur de sauvegarde ou qu'il s'agisse du superviseur actif n'étant pas à l'état d'anomalie, le code d'état 0x0C (Conflit d'état de l'objet) doit être renvoyé, et la valeur 0 doit être affectée aux attributs Last Active Node 1 et Last Active Node 2.

#### 7.9.6.3 Clear_Rapid_Faults

Le service Clear_Rapid_Faults doit annuler la condition selon laquelle le superviseur d'anneau a détecté un cycle de défaut/restauration rapide d'anneau (voir 10.5.3.5). Lors de l'annulation de la condition, le superviseur d'anneau doit reprendre un fonctionnement normal.

Si le service Clear_Rapid_Faults est reçu alors que le superviseur n'est pas activé, qu'il s'agisse d'un superviseur de sauvegarde ou qu'il s'agisse d'un superviseur actif dont la condition n'est pas celle de défaut/restauration rapide, le code d'état 0x0C (Conflit d'état de l'objet) doit être renvoyé.

#### 7.9.6.4 Restart_Sign_On

Le service Restart_Sign_On doit redémarrer le processus Sign_On (voir 10.5.2.3 Connexion pour de plus amples informations) par le superviseur d'anneau actif, s'il n'est pas déjà en cours.

Si le service Restart_Sign_On est reçu alors que le superviseur n'est pas activé, qu'il s'agisse d'un superviseur de sauvegarde ou qu'il s'agisse d'un superviseur actif dont la condition n'est pas NORMAL_STATE, le code d'état 0x0C (Conflit d'état de l'objet) doit être renvoyé. Si le service Restart_Sign_On est reçu lorsqu'un précédent processus Sign_On est en cours, le résultat doit être retourné et la demande doit être ignorée.

#### 7.9.6.5 Clear_Gateway_Partial_Fault

Le service Clear_Gateway_Partial_Fault doit annuler la condition où la passerelle a détecté un défaut de réseau partiel (voir Article 10). Une fois la condition annulée, la passerelle doit exécuter le diagramme d'états tel que décrit à l'Article 10.

Si le service Clear_Gateway_Partial_Fault est reçu alors que la passerelle n'est pas activée ou que la condition n'est pas un défaut de réseau partiel, le code d'état 0x0C (conflit d'état de l'objet) doit être renvoyé.

### 7.10 Objet QoS

#### 7.10.1 Présentation

La Qualité de service (QoS, pour Quality of Service) est un terme général relatif aux mécanismes utilisés pour traiter les flux de trafic dont les priorités relatives sont différentes ou offrant d'autres caractéristiques de remise. Les mécanismes QoS standard comprennent l'IEEE 802.1D/IEEE 802.1Q (Priorité des trames Ethernet) et les services différenciés (Differentiated Services, DiffServ) dans la suite de protocoles TCP/IP.

L'objet QoS permet de configurer certains comportements liés à QoS dans les appareils CP 2/2.

L'objet QoS est nécessaire aux appareils prenant en charge l'envoi de messages CP 2/2 contenant des points de code DiffServ différents de zéro (DSCP) ou l'envoi de messages CP 2/2 dans des trames balisées IEEE 802.1Q.

#### 7.10.2 Historique de révision

L'historique de révision de l'objet QoS est décrit dans le Tableau 123.

**Tableau 123 – Historique de révision de l'objet QoS**

| Revision | Historique |
|---|---|
| 01 | Définition initiale |

### 7.10.3 Attributs de classe

L'objet QoS doit prendre en charge les attributs de classe spécifiés dans le Tableau 124.

**Tableau 124 – Attributs de classe de l'objet QoS**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| 0x01 | Obligatoire | Get | Revision | UINT | Révision de cet objet | Valeurs en cours de cet attribut = 1 |
| 0x02 à 0x07 | Ces attributs de classe sont facultatifs et sont décrits dans la CEI 61158-5-2 | | | | | |

### 7.10.4 Attributs d'instance

#### 7.10.4.1 Généralités

L'objet QoS doit prendre en charge les attributs d'instance spécifiés dans le Tableau 125.

**Tableau 125 – Attributs d'instance de l'objet QoS**

| Attr ID | Nécessité dans mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| 0x01 | Sous condition[a] | Set | NV | 802.1Q Tag Enable | USINT | Permet d'activer ou de désactiver l'envoi de trames IEEE 802.1Q dans les messages CP 2/2 et CEI 61588 | Une valeur égale à 0 indique que les trames balisées sont désactivées. Une valeur égale à 1 indique que les trames balisées sont activées.<br>La valeur par défaut doit être 0. |
| 0x02 | Sous condition[b] | Set | NV | DSCP PTP Event | USINT | Valeur DSCP des messages d'événement PTP (CEI 61588) | Voir 7.10.4.3 |
| 0x03 | Sous condition[b] | Set | NV | DSCP PTP General | USINT | Valeur DSCP des messages généraux PTP (CEI 61588) | Voir 7.10.4.3 |
| 0x04 | Obligatoire | Set | NV | DSCP Urgent | USINT | Valeur DSCP de la classe de transport CP 2/2 0/1 Messages de priorité urgente | Voir 7.10.4.3 |
| 0x05 | Obligatoire | Set | NV | DSCP Scheduled | USINT | Valeur DSCP de la classe de transport CP 2/2 0/1 Messages de priorité planifiée | Voir 7.10.4.3 |
| 0x06 | Obligatoire | Set | NV | DSCP High | USINT | Valeur DSCP de la classe de transport CP 2/2 0/1 Messages de priorité élevée | Voir 7.10.4.3 |
| 0x07 | Obligatoire | Set | NV | DSCP Low | USINT | Valeur DSCP de la classe de transport CP 2/2 0/1 Messages de priorité basse | Voir 7.10.4.3 |

| Attr ID | Nécessité dans mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| 0x08 | Obligatoire | Set | NV | DSCP Explicit | USINT | Valeur DSCP des messages explicites CP 2/2 (classe de transport 2/3 et UCMM) ainsi que tous les autres messages d'encapsulation CP 2/2 | Voir 7.10.4.3 |
| a Requis si l'appareil prend en charge l'envoi de trames IEEE 802.1Q. | | | | | | | |
| b Requis si l'appareil prend en charge l'objet Time Sync. | | | | | | | |

### 7.10.4.2 802.1Q Tag Enable

L'attribut 802.1Q Tag Enable permet d'activer ou de désactiver l'envoi de trames IEEE 802.1Q dans les messages CP 2/2 et CEI 61588. Si l'attribut est activé, l'appareil doit envoyer des trames IEEE 802.1Q pour tous les messages CP 2/2 et CEI 61588.

Une valeur 1 indique qu'il est activé. Une valeur 0 indique qu'il est désactivé. La valeur par défaut doit être 0. Une modification de la valeur de l'attribut doit prendre effet au démarrage suivant de l'appareil.

Les appareils doivent toujours utiliser les valeurs DSCP correspondantes indiquant si les trames IEEE 802.1Q sont activées ou désactivées.

### 7.10.4.3 Attributs de valeur DSCP

Les attributs 0x02 à 0x08 contiennent les valeurs DSCP qui doivent être utilisées pour les différents types de trafic CP 2/2.

Le format de la valeur DSCP dans l'en-tête IP est spécifié dans la CEI 61158-6-2, 11.7.4. Etant donné que la taille du champ DSCP est de 6 bits, la plage de valeurs valide de ces attributs est comprise entre 0 à 63. Si la valeur DSCP est placée directement dans le champ ToS de l'en-tête IP, elle doit être décalée de 2 bits vers la gauche.

Le Tableau 126 spécifie les valeurs DSCP par défaut et les utilisations du trafic.

**Tableau 126 – Valeurs DCSP par défaut et utilisations**

| Attribut | Utilisation du type de trafic | DSCP par défaut |
|---|---|---|
| DSCP PTP Event | Messages d'événement PTP (CEI 61588) | 59 ('111011') |
| DSCP PTP General | Messages généraux PTP (CEI 61588) | 47 ('101111') |
| DSCP Urgent | Messages 0/1 de classe de transport avec priorité urgente | 55 ('110111') |
| DSCP Scheduled | Messages 0/1 de classe de transport avec priorité planifiée | 47 ('101111') |
| DSCP High | Messages 0/1 de classe de transport avec priorité élevée | 43 ('101011') |
| DSCP Low | Messages 0/1 de classe de transport avec priorité basse | 31 ('011111') |
| DSCP Explicit | Messages UCMM, messages de transport de classe 2/3 ainsi que tous les autres messages d'encapsulation CP 2/2 | 27 ('011011') |

Toute modification de la valeur des attributs ci-dessus doit prendre effet au démarrage suivant de l'appareil.

### 7.10.5 Services communs

L'objet QoS doit prendre en charge les services communs spécifiés dans le Tableau 127.

**Tableau 127 – Services communs de l'objet QoS**

| Service Code | Nécessité dans la mise en œuvre | | Service nom | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x01 | Facultatif | N/A | Get_Attribute_All | Renvoie plusieurs attributs dans l'ordre numérique |
| 0x0E | Sous condition[a] | Obligatoire | Get_Attribute_Single | Renvoie le contenu de l'attribut spécifié |
| 0x10 | N/A | Obligatoire | Set_Attribute_Single | Modifie un seul attribut |

[a] Obligatoire si des attributs de classe sont mis en œuvre.

### 7.10.6 Réponse Get_Attribute_All (niveau de la classe)

La réponse Get_Attributes_All doit contenir les attributs de classe dans l'ordre numérique, jusqu'au dernier attribut mis en œuvre. Tous les attributs non mis en œuvre dans la réponse doivent utiliser les valeurs d'attribut par défaut.

## 7.11 Objet de port

### 7.11.1 Présentation

L'objet de port décrit les ports CPF 2 présents sur l'appareil. Une instance doit exister pour chaque port routable CPF 2 (voir IEC 61158-5-2, 6.2.2.3 pour la définition d'un port routable). Les instances peuvent également exister pour les ports non routables. Les appareils à port CPF 2 unique ne sont pas obligés de prendre en charge l'objet de port.

### 7.11.2 Attributs de classe

L'objet de port doit prendre en charge les attributs de classe spécifiés dans le Tableau 128.

**Tableau 128 – Attributs de classe de l'objet de port**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| 0x01 | Cet attribut de classe est facultatif et est décrit dans la CEI 61158-5-2 | | | | | |
| 0x02 | Obligatoire | Get | Max Instance | UINT | Nombre maximal d'instances | |
| 0x03 | Obligatoire | Get | Num Instances | UINT | Nombre de ports actuellement instanciés | |
| 0x04 à 0x07 | Ces attributs de classe sont facultatifs et sont décrits dans la CEI 61158-5-2 | | | | | |
| 0x08 | Obligatoire | Get | Entry Port | UINT | Renvoie l'instance de l'objet de port qui décrit le port au niveau duquel la demande est entrée dans l'appareil | |
| 0x09 | Obligatoire | Get | Port Instance Info | MATRICE de STRUCT de | Matrice de structures contenant les attributs d'instance 1 et 2 de chaque instance | La matrice est indexée par le nombre d'instances, jusqu'au nombre maximal d'instances. |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|
| | | | | | | Les valeurs à l'indice 1 (décalage 0) et toutes les instances non instanciées doivent être nulles. |
| | | | Port Type | UINT | Indique le type de port | Voir l'attribut d'instance 1 |
| | | | Port Number | UINT | Numéro de port CPF 2 associé à ce port | Voir l'attribut d'instance 2 |

### 7.11.3 Attributs d'instance

#### 7.11.3.1 Généralités

L'objet de port doit prendre en charge les attributs d'instance spécifiés dans le Tableau 129.

**Tableau 129 – Attributs d'instance de l'objet de port**

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| 0x01 | Obligatoire | Get | NV | Port Type | UINT | Indique le type de port | Voir 7.11.3.2 |
| 0x02 | Obligatoire | Get | NV | Port Number | UINT | Numéro de port CPF 2 associé à ce port | Voir 7.11.3.3 |
| 0x03 | Obligatoire | Get | NV | Link object | STRUCT de | | |
| | | | | Path Length | UINT | Nombre de mots de 16 bits dans le chemin suivant | Plage = 2 à 6 |
| | | | | Link Path | EPATH rempli | Segments de chemin logique qui identifient l'objet pour ce port. Par exemple, il peut s'agir de l'objet d'interface TCP/IP. | Le chemin doit comporter un segment de classe logique et un segment d'instance logique.<br>La taille maximale est de 12 octets. Voir la définition des segments logiques dans la CEI 61158-6-2, 4.1.9.4. |
| 0x04 | Obligatoire | Get | NV | Port Name | SHORT_ STRING | Chaîne du nom du port réseau physique. Le nombre maximal de caractères dans la chaîne est de 64.<br>Par exemple, il peut s'agir de "port A". | Cette valeur est unique pour chaque port réseau physique. Si plusieurs ports CPF 2 utilisent le même port réseau physique, ils doivent tous avoir la même valeur Port Name.<br>Par ailleurs, un port CPF 2 ne doit pas avoir la même valeur Port Name qu'un autre port CPF 2 s'ils ne partagent pas le même port réseau physique. |

| ID attribut | Nécessité dans la mise en œuvre | Règle d'accès | NV | Nom | Type de données | Description d'attribut | Sémantique des valeurs |
|---|---|---|---|---|---|---|---|
| 0x05 | Facultatif | Get | NV | Port Type Name | SHORT_ STRING | Chaîne désignant le type de port. Le nombre maximal de caractères dans la chaîne est de 64.<br>Par exemple, il peut s'agir de "EtherNet/IP". | |
| 0x06 | Facultatif | Set | NV | Port Description | SHORT_ STRING | Chaîne décrivant le port. Le nombre maximal de caractères dans la chaîne est de 64.<br>Par exemple, il peut s'agir de "Product Line 22". | |
| 0x07 | Obligatoire | Get | NV | Node Address [a] | EPATH rempli | Numéro de nœud de cet appareil sur le port. La plage pour ce type de données est limitée à un segment de port. | Le numéro de port encodé doit correspondre à la valeur indiquée à l'attribut 2. |
| 0x08 | Sous condition | Get | NV | Port Node Range [b] | STRUCT de | | |
| | | | | Minimum Node Number | UINT | Nombre de nœuds minimal sur le port | |
| | | | | Maximum Node Number | UINT | Nombre de nœuds maximal sur le port | |
| 0x09 | Facultatif | Get | NV | Port Key | EPATH rempli | Clé électronique du réseau/châssis auquel est attaché le port.<br>Cet attribut doit être limité au format 4 du segment de clé électronique logique. | |

[a] Un appareil qui ne possède pas de numéro de nœud sur le port peut indiquer une adresse de nœud de longueur nulle dans le segment de port (0x10 0x00).

[b] Si un appareil peut faire état de ses caractéristiques de port dans la plage admise (par exemple, MAC ID de Type 2), il doit alors prendre en charge cet attribut. Sinon (par exemple adresse Internet), il ne doit pas prendre en charge cet attribut.

#### 7.11.3.2 Port Type

Toutes les valeurs Port Type, à l'exception de 0, indiquent des ports routables du type défini.

Les valeurs de Port Type sont:

0 = utilisé lorsque le port ne prend pas en charge le routage CIP. Attribut 2 (Numéro de port) est ignoré;

1 = réservé pour une compatibilité avec les protocoles existants;

2 = ControlNet;

3 = ControlNet redondant;

4 = EtherNet/IP (ce type de port était anciennement désigné par TCP/IP);

5 = DeviceNet;

6 à 99 = réservé pour une compatibilité avec les protocoles existants;

100 à 199 = spécifique au fournisseur;

200 = CompoNet;

201 = Modbus/TCP;

202 = Modbus/SL;

203 = SERCOS_III

204 à 65 534 = Réservé pour une utilisation ultérieure;

65 535 = port non configuré

#### 7.11.3.3 Port Number

Le fabricant affecte une valeur unique pour l'identification de chacun des ports de communication. La valeur 1 est définie pour une utilisation du produit interne (c'est-à-dire fond de panier). La valeur 0 est réservée et ne peut être utilisée.

La valeur Port Number est l'identificateur de port défini pour les segments de port dans la CEI 61158-6-2, 4.1.9.3 et utilisé pour le routage.

Si Port Type est égal à 0, l'attribut Port Number est ignoré.

### 7.11.4 Services communs

#### 7.11.4.1 Généralités

Les services communs mis en œuvre par l'objet de port sont spécifiés dans le Tableau 130.

**Tableau 130 – Services communs de l'objet de port**

| Code de service | Nécessité dans la mise en œuvre | | Nom du service | Description du service |
|---|---|---|---|---|
| | Classe | Instance | | |
| 0x01 | Facultatif | Facultatif | Get_Attribute_All | Retourne le contenu de tous les attributs de classe/d'instance |
| 0x05 | n/a | Facultatif | Reset | |
| 0x0E | Sous condition [a] | Sous condition [a] | Get_Attribute_Single | Utilisé pour lire l'attribut de classe/d'instance d'un objet de port |
| 0x10 | n/a | Facultatif | Set_Attribute_Single | |
| [a] Cet service doit être mis en œuvre si les attributs de classe/d'instance sont pris en charge. | | | | |

#### 7.11.4.2 Réponse Get_Attribute_All

##### 7.11.4.2.1 Niveau de classe

Au niveau de classe, la réponse Get_Attributes_All doit concaténer les attributs 1, 2, 3, 8 et 9 dans cet ordre. Si l'attribut de classe 1 (Revision) n'est pas pris en charge, alors une valeur par défaut (1) doit être retournée.

##### 7.11.4.2.2 Niveau d'instance

Au niveau d'instance, la réponse Get_Attributes_All doit concaténer les attributs 1, 2, 3, 4 et 7 dans cet ordre.

# 8 Autre éléments de procédure DLE

## 8.1 Moniteur de connexion au réseau (NAM)

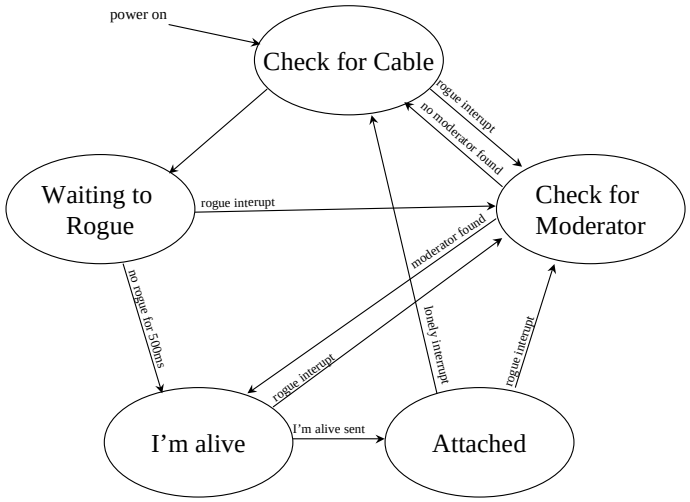
### 8.1.1 Généralités

Le moniteur de connexion au réseau (NAM) doit contrôler la DLL pour permettre aux nouveaux nœuds de rejoindre une liaison de travail sans interruption.

Le NAM contrôle l'association de la DLL à une liaison existante sans interrompre les transmissions sur cette liaison. Pour ce faire, il surveille et contrôle la DLL grâce à l'interface de gestion de station et à l'interface normale d'envoi/de réception avec la DLL (voir Tableau 131 et Figure 26).

**Tableau 131 – Etats NAM**

| Etat | Actions |
|---|---|
| **Check for Cable** | 1) les nœuds d'adresse automatique doivent trouver une adresse avant d'entrer (voir 8.1.3)<br>2) utiliser les paramètres par défaut (voir 8.1.2)<br>3) transmettre les DLPDU, mais pas les modérateurs<br>4) en cas de réception d'un modérateur suspect, aller à « Check for Moderator »<br>5) en cas de réception d'une DLPDU correcte, transmettre une DLPDU supplémentaire, puis aller à « Wait to Rogue » |
| **Wait to Rogue** | 1) utiliser les paramètres par défaut (voir 8.1.2)<br>2) ne transmettre aucune DLPDU<br>3) en cas d'écoute d'un modérateur suspect, aller à « Check for Moderator »<br>4) si aucun modérateur suspect n'est entendu pendant 400 ms à 600 ms, aller à « I'm Alive » |
| **Check for Moderator** | 1) ne transmettre aucune DLPDU<br>2) si 9 modérateurs valides identiques sont entendus dans des NUT successives, aller à « I'm Alive » en utilisant les paramètres dans ces modérateurs<br>3) si aucun modérateur n'a été entendu pendant 1,0 s à 3,0 s, aller à « Check for Cable »<br>4) en cas de réception d'un modérateur doté de paramètres de liaison non valides, rester à cet état, mais modifier le voyant d'état du réseau en configuration de liaison non valide (rouge/vert clignotant) |
| **I'm Alive** | 1) les nœuds d'adresse automatique doivent trouver une adresse avant d'entrer (voir 8.1.3)<br>2) transmettre les DLPDU, y compris les modérateurs, si ce nœud détient le MAC ID le plus bas<br>3) transmettre trois Lpackets 0x80 à balise fixe<br>4) après leur envoi, aller à « Attached »<br>5) en cas de réception d'un modérateur suspect, aller à « Check for Moderator » |
| **Attached** | 1) fonctionnement normal du réseau<br>2) transmettre les DLPDU, y compris les modérateurs, si ce nœud détient le MAC ID le plus bas<br>3) en cas de réception d'un modérateur suspect, aller à « Check for Moderator »<br>4) en l'absence d'autre nœud sur la liaison pendant 8 NUT, aller à « Check for Cable » |




**Légende**

| Anglais | Français |
|---|---|
| Power on | Sous tension |
| Rogue interrupt | Interruption rogue |
| No moderator found | Aucun modérateur trouvé |
| Moderator found | Modérateur trouvé |
| No rogue for 500ms | Pas de rogue pendant 500 ms |
| Lonely interrupt | Interruption indépendante |
| I'm alive sent | I'm alive envoyé |

**Figure 26 – Diagramme d'états du NAM**

### 8.1.2 Paramètres par défaut

Les paramètres de liaison par défaut d'un nœud doivent se présenter comme indiqué dans le Tableau 132:

**Tableau 132 – Paramètres de liaison par défaut**

| Paramètre | Type de données | Valeur |
|---|---|---|
| NUT_length | UINT | 100 ms (10 000 intervalles de 10 µs chacun) |
| smax | USINT | 0 |
| umax | USINT | 99 |
| slotTime | USINT | 254 (255 µs) |
| blanking | USINT | 6 (temps d'octet) |
| gb_start | USINT | 610 µs (61 intervalles de 10 µs chacun) |
| gb_center | USINT | 450 µs (45 intervalles de 10 µs chacun) |
| modulus | USINT | 127 |
| gb_prestart | USINT | 920 µs (92 intervalles de 10 µs chacun) |

### 8.1.3 Adressage automatique

Certains nœuds peuvent ne pas être dotés d'un MAC ID préalablement assigné. Ces nœuds, appelés nœuds d'adresse automatique, doivent rechercher un MAC ID non utilisé en

observant la liaison. Le nœud d'adresse automatique ne doit pas transmettre tant qu'un MAC ID libre n'a pas été trouvé.

NOTE En général, les nœuds d'adresse automatique sont des nœuds transitoires (des terminaux de données portatifs, par exemple).

### 8.1.4 MAC ID valides

Sur une liaison comportant des paramètres de configuration par défaut, un nœud d'adresse automatique doit rechercher dans la plage comprise entre 92 et 99. Sur une liaison comportant d'autres paramètres de configuration, un nœud d'adresse automatique doit rechercher dans la plage comprise entre SMAX+1 et UMAX. Un MAC ID libre doit être considéré comme ayant été trouvé si au moins trois opportunités de transmission non planifiées séquentielles correspondant à ce MAC ID ont eu lieu sans recevoir de DLPDU provenant de ce MAC ID.

NOTE L'objet ControlNet offre une interface permettant de déterminer le MAC ID qui a été choisi.

### 8.1.5 Description du diagramme d'états

La description ci-dessous du diagramme d'états doit définir le comportement du moniteur de connexion au réseau:

```
// Network Attachment Monitor state machine description
//
// This state machine monitors and controls the DLL to make sure that it goes on-line
// in a fashion that does not disrupt a running network.
//
//
// Lpacket constants: masks for ctl octet
//
#define FIXEDSCREEN 1
#define TAGPAD 2

#define DATAPAD 4
#define OCTETWORD 8
//
// moderator Lpacket constants
//
#define MODERATOR_SIZE 9      // moderator Lpacket size octet
#define MODERATOR_CTL 1       // moderator Lpacket control octet
#define MODERATOR_TAG 0       // moderator Lpacket tag octet

//
// Lpacket class
//
class Lpacket
   {
   public:

   USINT size;         // return the size octet
   USINT ctl;          // return the control octet

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }
   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }
   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }
   // get the next octet to be transmitted
   USINT get_next_octet(void);
```

```
   // is this Lpacket aborted?
   BOOL abort(void);
   };
class fixedLpacket: public Lpacket
   {
   public:
   USINT     tag;
   USINT     dest;
   };
class moderatorLpacket: public fixedLpacket
   {
   public:
   UINT      NUT_length;
   USINT     smax;
   USINT     umax;
   USINT     slotTime;
   USINT     blanking;
   USINT     gb_start;
   USINT     gb_center;
   USINT     usr;
   USINT     interval_count;
   USINT     modulus;
   USINT     tMinus;
   USINT     gb_prestart;
   USINT     spare;
   BOOL isValid(void);
                  // this checks for umax<myaddr, and similar parameter
                  // error that would prevent a node from successfully
                  // joining a network
   BOOL operator==(moderatorLpacket &);
                  // the equality test for moderator Lpackets excludes
                  // the interval count, and usr, but includes the
                  // tMinus counter
   };
//
// interface to station management
//
BOOL SM_powerup;          // input indication: powerup has occurred
void SM_LEDS(LED_STATE);        // set the LEDS to a specified pattern
#define LED_bad_network_parameters     // flashes the network status indicator

class DLL_config_data
   {
   public:
   BOOL   listen_only;
   USINT  myaddr;
   UINT   NUT_length;
   USINT  smax;
   USINT  umax;
   USINT  slotTime;
   USINT  blanking;
   USINT  gb_start;
   USINT  gb_center;
   USINT  interval_count;
   USINT  modulus;
   USINT  gb_prestart;
   };
//
// interface to DLL
//
void DLL_xmit_fixed_request(char *comment);
void DLL_xmit_fixed_confirm(void);
void DLL_recv_fixed_indication();
//
// These timer objects are as used internally to the ACM of the DLL.
//
// various behaviors for timers
enum {ONCE_UP, ONCE_DOWN, REPEAT_UP, REPEAT_DOWN} timer_mode;
class timer
   {
   public:
```

```
   float count;           // number of ticks left
   float preset;          // initialize ticks, or cycle length
   BOOL expire;           // latched flag when timer expires
   timer_mode mode;

   // constructor: defines mode
   timer(timer_mode m)
      {
      mode = m;
      }

   void restart(void)
      {
      if(mode == REPEAT_DOWN
       || mode == ONCE_DOWN) // if counting down,
         {
         count = preset;     // start with preset
         }
      else
         {
         count = 0;      // else start from zero
         }
      // start counting
      }

   void start(float interval)   // start counting down from interval towards zero
      {
      mode = ONCE_DOWN;
      preset = interval;     // set interval
      restart();          // start counting
      }

   void begin_counting(void)    // start counting up from zero
      {
      mode = ONCE_UP;
      restart();
      }

   bool expired(void)        // returns true if the timer has expired; clears flag
      {
      BOOL tmp;
      tmp = expire;
      expire = 0;
      return tmp;
      }
   };
//
// time conversion factors
//
#define usec
#define msec *1000
#define sec *1000000
class timer timer(ONCE_DOWN);    // general purpose timer
int counter;           // general purpose counter
moderatorLpacket new_mod;     // buffer for getting new moderator Lpackets
moderatorLpacket old_mod;     // buffer for saving moderator Lpackets
DLL_config_data default_config;  // DLL config parameters for default mode
DLL_config_data DLL_config;      // scratch DLL config buffer
// Wait for powerup.
// Bring the DLL on-line.
// It is assumed that the node's MAC ID is known before this machine is allowed to
power-up.
// If this node is auto-addressed, a free MAC ID shall be determined
// before enabling this machine.
state: powerup0

   event:    SM_powerup
   destination: checkForCable0
   action:
      DLL_set_current_request(default_config);  // set default parameters
state: checkForCable0

   event:    DLL_set_current_confirm()    // default parameters were set
   destination: checkForCable1
```

```
   action:
      DLL_enable_moderator_request(FALSE);      // disable moderators
      DLL_online_request(TRUE);       // send DLL on-line
state: checkForCable1

   event:    DLL_online_confirm()         // DLL is now on-line, but can't moderate
   destination: checkForCable2
state: checkForCable2

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   event:       DLL_recv_fixed_indication
             || DLL_recv_gen_indication   // any Lpacket received
   destination: waitForRogue
   action:
      wait until transmit once more;
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_moderator_request(FALSE);      // re-enable moderators
      timer.start(500 msec);

   event:    MAC has not transmitted any frames for 1300 to 1500 msec
   destination: checkForCable0
   action:
      DLL_set_current_request(default_config);     // set default parameters
state: waitForRogue

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   event:    timer.expired()
   destination: alive0
   action:
      DLL_listen_only_request(FALSE);     // send DLL to listen only
      DLL_xmit_fixed_request("send an I'm alive message");
//
// in the "alive" states, the DLL is on-line
// three "I'm alive" message are sent
// if lonely is detected, do nothing
//
state: alive0

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   // first message sent, send the second
   event:    DLL_xmit_fixed_confirm();
   destination: alive1
   action:
      DLL_xmit_fixed_request("send an I'm alive message");
state: alive1

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   // second message sent, send the third
   event:    DLL_xmit_fixed_confirm();
```

```
   destination: alive2
   action:
      DLL_xmit_fixed_request("send an I'm alive message");

state: alive2

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   // third message send, consider attachment to be complete
   event:    DLL_xmit_fixed_confirm();
   destination: attached

//
// the DLL is on-line and fully functional
//
state: attached

   // rogue -> checkForModerator
   event:    DLL_SM_report(rogue)
   destination: checkForModerator0
   action:
      DLL_listen_only_request(TRUE);      // send DLL to listen only
      DLL_enable_fixed_request(MODERATOR_TAG);  // enable reception of moderators

   event:    DLL_SM_report(lonely)
   destination: checkForCable
   action:
      DLL_enable_moderator_request(FALSE);      // disable moderators

//
// a rogue event was detected
// the DLL instantly disables itself (it goes to its rogue state)
//
state: checkForModerator0

   event:    DLL_enable_fixed_confirm();      // receive moderators enabled
   destination: checkForModerator1
   action:
      counter = 0;              // init moderator counter
      old_mod = NULL;
      timer.start(3 sec);

state: checkForModerator1

   // timeout
   event:    timer.expired
   destination: checkForCable0
   action:
      DLL_set_current_request(default_params);  // set default parameters


   // ignore irrelevant received DLPDUs
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is not mod
          || received Lpacket is not TUI
          || TUI.net_change == false
   destination: checkForModerator1

   // received a TUI indicating a net change in progress -- reset counter
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is TUI
          && TUI.net_change == TRUE    // the new moderator is different
   destination: checkForModerator1
   action:
      counter = 0;

   // received a moderator that would not permit this node
   // to operate normally (for example, - myaddr>umax)
   // reset counter and flash bad_net_parameter indicator
```

```
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   !isValid(new_mod)         // the new moderator is no good for us
   destination: checkForModerator1
   action:
      SM_LEDS(LED_bad_network_parameters);      // flash the indicator
      old_mod = new_mod;
      counter = 0;

   // received moderator that doesn't match the last one -- reset counter
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is mod
         && new_mod!= old_mod         // the new moderator is different
   destination: checkForModerator1
   action:
      old_mod = new_mod;
      counter = 0;

   // received moderator that matches, but not the ninth match -- just increment the
counter
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is mod
         && isValid(new_mod)
         && new_mod==old_mod
         && counter<8
   destination: checkForModerator1
   action:
      counter++;

   // have received 9 successive identical moderators
   // adopt these parameters and proceed to go back on-line
   event:    DLL_recv_fixed_indication(new_mod)  // got a moderator Lpacket
   condition:   received Lpacket is mod
         && isValid(new_mod)
         && new_mod==old_mod
         && counter==8
   destination: checkForModerator2
   action:
      DLL_online_request(FALSE);      // send DLL off-line

//
// second step to going back on-line
//
state: checkForModerator2

   event:    DLL_online_confirm()         // now off-line
   destination: check ForModerator3
   action:
      copy parameters from new_mod to DLL_config;   // get net configuration from
moderator
      DLL_set_current_request(DLL_config);      // adopt those parameters

//
// third step to going back on-line
//
state: checkForModerator3

   event:    DLL_set_current_confirm()    // have now adopted new parameters
   destination: alive0
   action:
      DLL_listen_only_request(FALSE);     // send DLL to listen only
      DLL_xmit_fixed_request("send an I'm alive message");
```

## 8.2 Calcul des paramètres de liaison

### 8.2.1 Paramètres de liaison

Le Paragraphe 8.2 décrit les exigences relatives à la sélection des valeurs des variables de configuration suivantes:

– NUT_length;

– gb_prestart;

– gb_start;
– gb_center;
– slotTime;
– blanking.

NOTE Les définitions des paramètres de liaison incluent `NUT_length`, `gb_center`, `gb_prestart`, `gb_start`, `blanking`, et `slotTime`.

### 8.2.2 Conditions affectant les paramètres de liaison

La sélection des valeurs de paramètre doit admettre les conditions les plus défavorables de ces facteurs:

– temps de propagation du signal entre les nœuds;
– exactitude de référence de temporisation aux différents nœuds;
– les modifications du modérateur donnent lieu à une modification de la distance du câble ou du temps de propagation du signal entre le modérateur et des nœuds non modérateurs.

NOTE 1 En règle générale, une modification de modérateur se déroule comme suit: à un instant donné, le modérateur précédent est déconnecté du câble, n'est plus alimenté ou arrête de transmettre des modérateurs pour certaines raisons. Deux NUT (parfois trois) s'écoulent, au cours desquelles aucune DLPDU modérateur n'est transmise. Le nouveau modérateur transmet sa première DLPDU modérateur dans la bande de garde de la troisième NUT depuis la dernière réception d'une DLPDU modérateur valide.

NOTE 2 Si le modérateur précédent a été perdu au moment de la transmission de la DLPDU modérateur, il se pourrait que le nouveau modérateur ait reçu la dernière DLPDU modérateur du précédent modérateur, alors que les autres nœuds n'auraient pas reçu la dernière DLPDU modérateur.

Par conséquent, il est nécessaire que les nœuds tolèrent un intervalle de 4 NUT entre la réception des DLPDU modérateurs valides.

### 8.2.3 Changement de modérateur

Dans les exigences qui suivent, le terme « changement de modérateur » doit être interprété par un nœud non modérateur comme étant:

– la réception d'une DLPDU modérateur valide provenant du nœud modérateur d'origine,
– un intervalle de 4 NUT pendant lequel aucune DLPDU modérateur valide n'est reçue,
– la réception d'une DLPDU modérateur valide provenant du deuxième nœud modérateur.

### 8.2.4 Temporisation NUT

#### 8.2.4.1 Tonalité

Une durée de mise à jour du réseau (NUT, pour Network Update Time) d'un nœud doit commencer en un instant appelé tonalité et doit se terminer à la tonalité suivante.

NOTE Les durées spécifiées dans le Paragraphe 8.2.4 supposent une horloge parfaitement précise. Toute imprécision admise est explicitement incluse dans les exigences. Les sources d'imprécision peuvent inclure le délai du microprogramme, le délai du matériel et l'inexactitude du cristal local.

Etant donné que les exigences de plage de temps ne spécifient pas explicitement la tolérance de précision de l'horloge, il convient que les mises en œuvre prévoient de définir les temporisateurs internes de manière à satisfaire aux exigences.

#### 8.2.4.2 Précision de l'horloge

Les spécifications de temporisation de la signalisation PhL pour prendre en charge la tonalité et la précision NUT doivent être telles que définies dans le Tableau 133.

**Tableau 133 – Caractéristiques de temporisation PhL**

| Spécification | Limites/caractéristiques | Commentaires |
|---|---|---|
| Vitesse de transmission des données | 5 Mbit/s ± CA | également appelé débit M_Symbol, *données « zéro » ou « un »* |
| Temps binaire | 200 ns ± CA | également appelé temps M_Symbol, *données « zéro » ou « un »* |
| Temps de symbole PhL | 100 ns ± CA | également appelé temps Phy_Symbol, voir les règles de codage des données |
| Précision de l'horloge (CA) | ± 150 parties/million maximum | y compris la température et la stabilité à long et court termes |

#### 8.2.4.3 Restriction sur `NUT_length`

La valeur maximale de `NUT_length` doit être 10 000 (100 ms). La valeur minimale `NUT_length` doit assurer que chaque NUT permet d'envoyer un Lpacket non planifié de longueur maximale après l'envoi des Lpackets planifiés.

NOTE Par exemple, avec les connexions non planifiées, la valeur minimale de `NUT_length` est d'environ 1 ms.

#### 8.2.4.4 Temporisation du nœud modérateur

Le nœud modérateur doit commencer une nouvelle NUT à (`NUT_length` × 10 μs) ± CA après la tonalité précédente. La réception d'une DLPDU valide ou non valide ne doit pas affecter la temporisation de la transmission de la DLPDU modérateur par le nœud modérateur.

Le nœud modérateur doit commencer à transmettre la DLPDU modérateur à (`NUT_length` – `gb_center`) × 10 μs ± CA après la tonalité.

#### 8.2.4.5 Temporisation du nœud non modérateur

Un nœud non modérateur doit commencer une nouvelle NUT à (`gb_center` × 10 – 40 ± 2) μs après le dernier M_Symbol du délimiteur de fin d'une DLPDU modérateur valide reçue. Un nœud non modérateur doit commencer une nouvelle NUT à `NUT_length` × 10) μs ± CA après la tonalité précédente si une DLPDU modérateur valide n'est pas reçue dans la NUT en cours.

Chaque nœud non modérateur doit recevoir et traiter une DLPDU modérateur valide qui commence entre (tonalité + 20) μs et (tonalité + (`NUT_length` × 10 – 30) μs ± CA).

#### 8.2.4.6 Silence post-tonalité

Aucune DLPDU ne doit être transmise entre la tonalité et 20 μs après la première tonalité.

#### 8.2.4.7 Silence pré-tonalité

Aucune DLPDU ne doit être transmise, sauf la DLPDU modérateur, si elle ne se termine pas avant ((`NUT_length` – `gb_prestart`) × 10 + 11,2) μs ± CA après la tonalité.

La valeur de `gb_prestart` doit faire en sorte que le temps de blocage après la dernière DLPDU non modérateur transmise par un nœud et reçue par un autre nœud expire avant le début de la bande de garde du nœud destinataire. Cette condition doit également être satisfaite pendant une modification de modérateur.

#### 8.2.4.8 Début de la bande de garde

La bande de garde doit commencer à (NUT_length – gb_start) × 10 μs ± CA après la tonalité.

La valeur de gb_start doit assurer qu'un nœud ne doit pas observer le début de la DLPDU modérateur avant ((NUT_length – gb_start) × 10 + 20) μs ± CA après la tonalité précédente de ce nœud. Cette condition doit être satisfaite pendant une modification de modérateur.

#### 8.2.4.9 Centre de la bande de garde

La valeur de gb_center doit assurer qu'un nœud ne doit pas observer la fin de la DLPDU modérateur après (NUT_length × 10 – 40) μs ± CA après la tonalité. Cette condition doit également être satisfaite pendant une modification de modérateur.

#### 8.2.4.10 Temps utile

Chaque nœud doit recevoir et traiter une DLPDU valide commençant entre (tonalité + 20 μs) et (tonalité + (NUT_length – gb_start) × 10 μs ± CA), sauf que la réception d'une DLPDU valide ou non valide ne doit pas affecter la temporisation de la transmission de la DLPDU modérateur par le nœud modérateur.

### 8.2.5 Intervalle

#### 8.2.5.1 Intervalle

Dans la description suivante, les emplacements doivent être numérotés de manière séquentielle. La valeur de slotTime doit assurer qu'une DLPDU transmise dans l'intervalle N est reçue dans l'intervalle N de tous les autres nœuds, en supposant les conditions les plus défavorables suivantes:

– dérive du cristal;
– délai de propagation du signal entre les nœuds;
– attribution du MAC ID;

mais à l'exclusion des événements bruyants.

#### 8.2.5.2 Premier intervalle

L'intervalle planifié 0 doit commencer à 20 μs après la tonalité.

#### 8.2.5.3 Nœud manquant

Si aucune DLPDU n'est envoyée ou reçue pendant l'intervalle N-1, l'intervalle N doit commencer à slotTime × 1 μs ± CA après le début de l'intervalle N-1.

#### 8.2.5.4 Redémarrage du temporisateur d'intervalle

Si une DLPDU est envoyée ou reçue pendant l'intervalle N-1, l'intervalle N ne doit pas commencer après 6,0 μs après le dernier M_Symbol du délimiteur de fin de la DLPDU. Si une DLPDU endommagée, avec ph_status_indication différent de Normal, est reçue pendant l'intervalle N-1, l'intervalle N ne doit pas commencer après 6,0 μs après que la valeur FALSE est attribuée à ph_lock_indication.

#### 8.2.5.5 Temps de réponse du nœud

Si les nœuds N-1 et N transmettent dans leurs intervalles de temps respectifs, le nœud N doit commencer à transmettre entre 11,2 μs et 12,8 μs après le dernier M_Symbol du délimiteur de fin de la DLPDU transmise par le nœud N-1.

#### 8.2.5.6 Début de la transmission après le nœud manqué

Si le nœud N-1 ne transmet pas dans son intervalle et que le nœud N est activé pour transmettre dans son intervalle, le nœud N ne doit pas commencer à transmettre avant le début de l'intervalle N, et pas après 6,8 μs après le début de l'intervalle N.

Le principal facteur à prendre en compte est la dérive du cristal. Dans le pire des cas, si deux nœuds sont associés au réseau à MAC ID 0 et 99, SMAX=98, UMAX=99 et USR=1, il peut exister un silence de 196 intervalles de temps entre la DLPDU planifiée transmise par le nœud 0 et la DLPDU non planifiée transmise par le nœud 99. Si les nœuds 99 et 0 sont respectivement dotés d'un cristal rapide et lent (ou inversement), il convient que la valeur de `slotTime` permette de garantir que, lorsque le nœud 99 transmet, le nœud 0 accepte l'intervalle en cours 99.

### 8.2.6 Blocage

La valeur de `blanking` doit être 6.

### 8.2.7 Exemple de mise en œuvre

Les exigences de 8.2.4, 8.2.5 et 8.2.6 sont satisfaites (de manière non optimale) dans l'exemple de programme ci-dessous qui calcule les paramètres de liaison:

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>

#define roundup(x,y) (((x) - 1L)/(y) + 1L) /* Round up integer division */

#define F_TURN      16060L    /* ns for firmware turnaround and detect      */
#define B_TURN       5720L    /* ns extra turnaround with large blanking value */
#define NULLFRAME    7L    /* octets for length of a null Lpacket transmission */
#define DELAY     4110L    /* ns per 1000 meters cable delay          */
#define RPT_DLY   815L     /* ns per repeater delay            */
#define PPMT      1500L    /* Max crystal error in tenths of PPM = +/- 150,0 ppm */

#define MODBEFORE    2     /* min moderator overhead before Lpacket (housekeeping)*/
#define MODRECOV 3     /* worst case min moderator overhead after any Lpacket */
#define MODTIME  5     /* ticks for the moderator plus error recovery if bad */
#define MINUNSCHED  850    /* minimum unscheduled time */

int main(int argc, char **argv)
   {

   /* Default values for command line input parameters. These are
    * arbitrarily chosen for our typical convenient network sizes.
    */
   int dest  = 0xff;
   int board = -1; /* If -1 use driver, else use XT version TX */
   int redund   = 3; /* Unspecified. Later code defaults to 3 */
   long NUT_time= 10000L;
   int length   = 1000;
   int smax  = 7;
   int umax  = -1;
   int repeater = 0;
   int blank = 6;
   int maxfrm   = 255;
   int intmod   = 256;
   int terse = 0;
   int listenonly  = 0;
```

```
    int noleds   = 0;

    /* Other derived variables. */

    int gap;
    long drift;
    long prop;
    long turn;
    long bslot;
    long slot;
    long picodev;
    int center;
    int start;
    int prestart;
    long unscheduled;
    long nuttenth;
    int nuthigh;
    int nutlow;
    char txhead[7];
    char *a;

    /* Parse arglist */

    argc--;
    argv++;
    if(argc==0)
       {
       argc=1;
       argv[0]="-h";
       }

    while(argc && (a=&argv[0][0],*a++=='-'))
       {
       while(*a)switch(*a++)
          {
       case 'x':
          board = atoi(a);
          if ((board < 0) || (board > 3))
             {
             printf("%%ERROR -- 0 <= board number <= 3.\n");
             return(1);
             }
          goto nextarg;

       case 'l':
          listenonly = 128;
          break;
       case 'n':
          noleds = 64;
          break;
       case 'b':
          redund = 1;
          break;
       case 'a':
          redund = 2;
          break;
       case 'r':
          redund = 3;
          break;
       case 't':
          terse = 1;
          break;

       case 'h':
       default:
          printf("Compute and print a net config Lpacket.\n\n");
          printf("NETCONF [-abdhlnrt] [-x<board>] [0x<dest>]"
             " [<NUT time> [<length> [<smax>\n");
          printf(" [<umax> [<repeaters> [<blanking> [<maxframe> "
             "[<int modulus>]]]]]]]\n");
          return(1);
          }
nextarg:  argc--;
       argv++;
```

```
    }

/* Check for network address option */
if (argc && argv[0][0] == '0' && argv[0][1] == 'x')
    {
    sscanf (argv[0],"%i",&dest);
    if ((dest < 0) || (dest > 255))
        {
        printf("%%ERROR -- 0x00 <= destination address <= 0xff.\n");
        return(1);
        }
    argc--;
    argv++;
    }


/* Parse remaining arguments.
 */
switch (argc)
    {
case 8:   intmod = atoi(argv[7]);
case 7:   maxfrm = atoi(argv[6]);
case 6:   blank = atoi(argv[5]);
case 5:   repeater = atoi(argv[4]);
case 4:   umax = atoi(argv[3]);
case 3:   smax = atoi(argv[2]);
case 2:   length = atoi(argv[1]);
case 1:   NUT_time = atol(argv[0]);
    }

/* check the validity of all the arguments */

if ((intmod < 1) || (intmod > 256))
    {
    printf("%%ERROR -- 1 <= interval count modulus <= 256.\n");
    return(1);
    }
if ((maxfrm < 0) || (maxfrm > 255))
    {
    printf("%%ERROR -- 0 <= max scheduled frame <= 255.\n");
    return(1);
    }
if ((blank < 0) || (blank > 255))
    {
    printf("%%ERROR -- 0 <= blanking <= 255.\n");
    return(1);
    }
if (repeater < 0)
    {
    printf("%%ERROR -- number repeaters >= 0.\n");
    return(1);
    }
if ((smax < 0) || (smax > 254))
    {
    printf("%%ERROR -- 0 <= max scheduled addr <= 254.\n");
    return(1);
    }
if (length < 0)
    {
    printf("%%ERROR -- cable length >= 0.\n");
    return(1);
    }
if ((NUT_time < 350L) || (NUT_time > 100000L))
    {
    printf("%%ERROR -- 350 <= nut time <= 100,000.\n");
    return(1);
    }

if (umax == -1) umax = smax;
if ((umax < smax) || (umax > 254))
    {
    printf("%%ERROR -- smax <= umax <= 254.\n");
    return(1);
```

```
   }
if (NUT_time%10!=0)
   {
   printf("%%ERROR -- NUT length is specified in 10 μs intervals.\n");
   return(1);
   }

/* Gap is the number of slot times that can pass in the normal
 protocol between transmissions.

 When smax<umax,then the maximum silence equals smax + umax - 1.
 The maximum silence occurs on the network with smax<umax when
 the first node is at 0, the only other is at umax with the USR==1.

 To illustrate a transmit sequence example when smax<umax,
 set smax=4 and umax=5, USR==1, with two nodes at 0 and 5.
 Slot times -> ssssuuuu -> (smax + umax - 1)
          01234123450

 When smax=umax, then the maximum silence equals smax + umax - 2.

 The maximum silence occurs on the network with smax = umax when
 the first node is at 0, the only other is at 1 with the USR===2.

 To illustrate a transmit sequence example when smax=umax,
 set smax=umax=5, USR==2, with two nodes at 0 and 1.
 Slot times -> ssssuuuu -> (smax + umax - 2)
          012345234501

 Units: slot times
*/

gap = (smax < umax): smax + umax - 1: smax + umax - 2;
if (gap < 0) gap = 0;

/* the drift is a correction factor needed to compensate for
 the amount of time that the fastest and slowest nodes drift
 apart during the longest silence due to crystal inaccuracy.

 Actual drift factor = (1-(2*gap+1)(ppm))/(1-ppm^2)
 however, ppm^2 is so small it may be ignored.

 Units: unitless factor scaled by micro/pico
*/

drift = roundup(10000000L - (long)(2*gap+1) * PPMT , 10L);

/* Prop is the time the signal takes to travel down the cable and
 work its way through all the repeaters and modems. It assumed
 on even a cable system that has no repeater boxes, that two
 two repeater exist to allow nodes connected via the NAP.

 Units: ps
*/
prop = (long)(length)*DELAY + (long)(repeater+2)*RPT_DLY*1000L;


/* turn is:

 - the amount of time it takes for the DLL to respond to a
 transmission, that is, - the time from the end of the end delimiter
 to the start of the preamble

   plus

 - the time is takes another DLL to detect that the first *has*
 responded, that is, - the time from the start of the preamble to the
 end of the start delimiter.

 Units: ns
*/
                /* turnaround could be */
turn = (long)(blank)*1600L + B_TURN;   /* <-- blanking constrained */
if (turn < F_TURN) turn = F_TURN;   /* <-- or firmware constrained */
```

```
/* bslot is the ideal slot time - which is usually defined as
 2 props + turnaround + detect (note that in this case, the
 detect time and the turn time are combined)
 Units: ps
*/

bslot = 2*prop + (turn*1000L);

/* slot is the ideal slot time adjusted to account for variances in
 crystal frequency.
 Units: µs
*/

slot = roundup(bslot , drift);
if ((slot < 1L) || (slot > 256L))
    {
    printf("%%ERROR -- Slot time is large, reduce network complexity. \n");
    return(1);
    }

/* Calculate guardband parameters (round up to a multiple of a 10 µs tick clock).*/

/* calc nut time - units: 10 usec ticks */
nuttenth = roundup(NUT_time,10L);

/* picodev is the amount the clocks of the slowest and fastest
 node drifts apart during four NUTs
 Units: ps
*/
picodev = nuttenth * PPMT * 8L;


/* center is the time before the next tone that the moderator
 begins transmitting the moderator DLPDU. This includes adjustments
 for clock drift during up to 4 NUTs, plus a moderator switchover
 from a near to a far node.
 Unit: 10 usec ticks
*/
center = MODTIME + MODRECOV +
 (int)roundup(picodev + 2*prop , 10000000L);

/* start is the time before the next tone that the guardband starts.
 Unit: 10 usec ticks
*/
start = center + MODBEFORE +
 (int)roundup(picodev , 10000000L);


/* prestart is the time before the next tone beyond which nodes may not
 transmit. Due to clock skew and other factors a transmission which
 is sent such that it ends just at prestart may actually be received
 by another node such that the transmission ends just as the guardband
 starts. The prestart margin is required in order to ensure that
 transmissions from slow nodes do not cross over into the guardbands
 of faster nodes.
 Unit: 10 usec ticks
*/
prestart = (int)roundup((nuttenth*PPMT*2L) + ((NULLFRAME+blank)*1600000L)
 + bslot , 10000000L) + start;

/* Determine if nut parameters is practical and print warnings.
 * Initial calculations are conservative due to roundup of several
 * parameters. Usage of fixround helps correct it.
 */

/* calc the max length of unscheduled time assuming that all
 scheduled nodes are silent.
 Units: µs
*/
```

```
   unscheduled = NUT_time - max(roundup(NULLFRAME*1600000L+prop+turn*1000L,
                   1000000000L),slot)*(long)(smax+1)
                               - (long)(prestart)*10L;
   if (unscheduled < MINUNSCHED)
      printf("%%WARNING -- Not enough scheduled time available. Increase nut
time.\n");

   /* Print statistics and the transmit command.
    */
   nutlow = nuttenth & 0xff;
   nuthigh = (nuttenth >> 8) & 0xff;
   if(!terse)
      {
      long deviation = roundup(picodev , 1000000L);

      printf("slot = %ld usec, max unscheduled time = %ld usec\n",
         slot,unscheduled);
      printf("dev = %ld usec, prestart = %d ticks, start = %d ticks, center = %d
ticks\n",
         deviation,prestart,start,center);
      }
   if (board == -1) sprintf(txhead,"tx");
   else sprintf(txhead,"txt /%d",board);
   printf(   "%s m a 17 %02x %02x %02x %02x %02x %02x %02x "
      "%02x %02x %02x %02x %02x %02x 00 00 00\n",
      txhead,dest,nutlow,nuthigh,smax,umax,(int)slot-1,blank,
      start,center,(listenonly|noleds|redund),maxfrm,intmod-1,prestart);
   return(0);
   }
```

# 9 Spécification détaillée des composants DL

## 9.1 Généralités

L'Article 9 spécifie chaque composant fonctionnel principal de l'architecture interne DL (voir Figure 2) à l'aide du langage de modélisation de 6.1.

## 9.2 Machine de contrôle d'accès (ACM)

La machine de contrôle d'accès (ACM) doit contrôler le séquencement des transmissions par ce nœud.

```
// ACM State Machine Description

////////////////////////////////////////////////////////////////////////////
// constant and type definitions

//
// generic type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;

//
// protocol constants
//
#define LOWCOUNTINIT 2 // NUTs as lowman before switchover to moderator
#define MODERATOR_LENGTH 40  // length of the moderator DLPDU in usec
#define TXDUPCHECKS 3  // moderators that have to be heard before transmit at powerup
#define TXNOMOD 5   // number of missed moderators before disabling transmit
#define DEAFNESS 5  // number of missed moderators before adjusting phase
#define DEAFADJUST 5   // 10 usec ticks to slip NUT if deafness is detected
#define LONELYINIT 8   // NUTs that are tolerated without hearing anything
             // before becoming lonely (should be longer than DEAFNESS
             // to ensure time for deaf recovery)
////////////////////////////////////////////////////////////////////////////
// Lpacket definition

//
// Lpacket constants: masks for control octet
//
```

```
#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

//
// moderator Lpacket constants
//
#define MODERATOR_SIZE 9  // moderator Lpacket size octet (in 16 bit words)
#define MODERATOR_CTL 1   // moderator Lpacket ctl octet (fixed tag)
#define MODERATOR_TAG 0   // moderator Lpacket service octet

//
// Lpacket class
//
class Lpacket
   {
   public:

   USINT size;  // the size octet
   USINT ctl;   // the ctl octet

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }

   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }

   // return the size of the Lpacket (in octets)
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }

   // get the next octet to be transmitted
   USINT get_next_octet(void);

   // is this Lpacket aborted?
   BOOL abort(void);
   };

//
// fixed tag Lpacket (two octet tag)
//
class fixedLpacket: public Lpacket
   {
   public:

   USINT  service;
   USINT  dest;
   };

//
// moderator fixed tag Lpacket
//
class moderatorLpacket: public fixedLpacket
   {
   public:

   UINT   NUT_length;
   USINT  smax;
   USINT  umax;
   USINT  slotTime;
   USINT  blanking;
   USINT  gb_start;
   USINT  gb_center;
   USINT  usr;
   USINT  interval_count;
```

```
   USINT  modulus;
   USINT  tMinus;
   USINT  gb_prestart;
   USINT  spare;
   };


////////////////////////////////////////////////////////////////////////
//
// internal VARIABLES
//
// unless otherwise specified, all variables are initialized to zero at powerup

int tmp;
USINT  interval_count;  // counts how many NUTs have passed (0 - modulus)
USINT  itr;          // implicit token register - tracks which MAC should xmit
USINT  usr;          // unscheduled start register -- first unscheduled token
USINT  tMinus;          // countdown to synchronized change
USINT  lowcount;     // how many times have I been lowman?
USINT  modcount;     // counts moderators (or missed moderators) depends on netState
USINT  modcountinit;    // how many NUTs to check for dupnode
USINT  nqlowman;     // optimistic lowman detector
USINT  qlowman;      // pessimistic lowman detector
USINT  lonely;          // how many NUTs have I been lonely
USINT  deafcount;       // number of times moderator apparently occurred during deaf
period
BOOL   in_guardband;    // TRUE during the guardband
BOOL   dupflag;      // edge detector for dupnode
BOOL   scheduled;       // scheduled/unscheduled part of the NUT
UINT   octetsleft;      // octets/octets left before end of current transmitted DLPDU
USINT  macSrce;      // source MAC ID for the current DLPDU
USINT  currentMod;      // the current moderator
moderatorLpacket moderator;  // buffer for moderator Lpackets


enum { OFFLINE,
   WATCH,
   DUPCHECK,
   NOTMOD,
   MOD,
   ROGUE,
   DUPNODE
   } netState; // connection state
////////////////////////////////////////////////////////////////////////
// timer definition
//
// various behaviors for timers
typedef enum {ONCE_UP, ONCE_DOWN, REPEAT_UP, REPEAT_DOWN} timer_mode;

class timer
   {
   public:
   float  count;    // number of ticks left
   float  preset;       // init ticks, or cycle length
   BOOL   expire;       // latched flag when timer expires
   timer_mode mode;

   // constructor: defines mode
   timer(timer_mode m)
      {
      mode = m;
      }

   void restart(void)
      {
      // if counting down, start with preset
      if(mode == REPEAT_DOWN || mode == ONCE_DOWN)
         {
         count = preset;
         }
      else
         {
         count = 0;   // else start from zero
         }
```

```
      // start counting (implementation not specified)
      }

   void start(float interval)// start counting down from interval towards zero
      {
      mode = ONCE_DOWN;
      preset = interval;  // set interval
      restart();      // start counting
      }

   void begin_counting(void) // start counting up from zero
      {
      mode = ONCE_UP;
      restart();
      }


   bool expired(void)     // returns true if the timer has expired; clears flag
      {
      BOOL tmp;

      tmp = expire;
      expire = 0;
      return tmp;
      }

   };
//
// time conversion factors
//
#define usec
#define msec *1000
#define sec *1000000

//
// define the timers used by the ACM
//
class timer gen_timer(ONCE_DOWN);   // general purpose, in several places
class timer slot_timer(REPEAT_DOWN);   // used for response time-out between messages
class timer watch_timer(ONCE_DOWN); // used for watch time state
class timer NUT_timer(REPEAT_DOWN); // this generates the NUT timing
class timer holdoff_timer(ONCE_DOWN);  // holds off transmit for a timer after
transmit or receive
////////////////////////////////////////////////////////////////////////////
//
// interface to PhL
//
BOOL ph_lock_indication;  // optimistic DLPDU detect (clock recovery is tracking)
BOOL ph_frame_indication; // pessimistic DLPDU detect (Manchester valid since the SD)


////////////////////////////////////////////////////////////////////////////
//
// interface to Deserializer
//
BOOL RX_dataReady;     // an octet of a DLPDU is available
USINT RX_rxData;    // octet most recently received from DLPDU (source MAC ID)
BOOL RX_endMAC;     // the end delimiter has been detected
BOOL RX_FCSOK;         // FCS on last DLPDU was OK
BOOL RX_abort;         // abort received on last DLPDU


////////////////////////////////////////////////////////////////////////////
//
// interface to RxM
//
BOOL RX_receivedLpacket;  // an Lpacket has been received
moderatorLpacket RX_Lpacket; // the Lpacket most recently received
////////////////////////////////////////////////////////////////////////////
//
// interface to transmitter (TxM)
//
void TX_sendHeader(USINT myaddr);   // send preamble, SD, source MAC ID
```

```
void TX_sendLpacket(Lpacket &lp);   // send an Lpacket
void TX_sendTrailer(void);    // send FCS, ED
BOOL TX_headerComplete;       // header has been sent
BOOL TX_LpacketComplete;      // Lpacket has been sent
BOOL TX_trailerComplete;      // trailer has been sent
BOOL TX_abort;            // DLPDU was aborted


////////////////////////////////////////////////////////////////////////////
//
// interface to LLC
//
Lpacket LLC_Lpacket;          // shared variable between LLC and ACM containing the
Lpacket
BOOL LLC_pickLpacket (BOOL scheduled,
       UINT octetsLeft);// tell LLC to pick an Lpacket for transmission
LLC_LpacketSent(void);        // tell LLC that the Lpacket was sent
////////////////////////////////////////////////////////////////////////////
//
// interface to station management
//

//
// various events that can be reported to SM
//
typedef enum
   {
   DLL_EV_rxGoodFrame,
   DLL_EV_txGoodFrame,
   DLL_EV_badFrame,
   DLL_EV_errA,
   DLL_EV_errB,
   DLL_EV_txAbort,
   DLL_EV_NUT_overrun,
   DLL_EV_dribble,
   DLL_EV_nonconcurrence,
   DLL_EV_rxAbort,
   DLL_EV_lonely,
   DLL_EV_dupNode,
   DLL_EV_noisePulse,
   DLL_EV_collision,
   DLL_EV_invalidModAddress,
   DLL_EV_rogue,
   DLL_EV_deafness,
   DLL_EV_supernode,
   }event;

extern void  DLL_event(event ev, int opt=0);     // report an event
extern void  DLL_currentMod_indication(USINT mac_ID);
extern void  DLL_tminus_zero_indication(void);

extern void  DLL_online_confirm(BOOL en);
extern void  DLL_listen_only_confirm(BOOL en);
extern void  DLL_tone_indication(USINT NUT_count);

class DLL_config_data
   {
   public:

   USINT  myaddr;

   UINT   NUT_length;
   USINT  smax;
   USINT  umax;
   USINT  slotTime;
   USINT  blanking;
   USINT  gb_start;
   USINT  gb_center;
   USINT  interval_count;
   USINT  modulus;
   USINT  gb_prestart;
   };
```

```
// interface to DLL management interface

extern DLL_config_data DLL_SM_pending;
extern DLL_config_data DLL_SM_current;

extern BOOL SM_powerup;

extern BOOL  SM_mod_enable;      // state of mod_enable
extern BOOL  SM_online;      // state of on-line/off-line
extern BOOL  SM_online_req;      // pending state of on-line/off-line
extern BOOL  SM_listen_only; // listen only state
extern BOOL  SM_listen_only_req; // pending listen only state
////////////////////////////////////////////////////////////////////////////
//
// subroutines
//
//
// take care of common processing for network parameter changes
//
void handle_net_change(void)
   {
   if(SM_listen_only!= SM_listen_only_req)// handle completion of listen_only change
      {
      SM_listen_only = SM_listen_only_req;
      DLL_listen_only_confirm(SM_listen_only);
      }

   if(tMinus==1)            // handle completion of synchronized change
      {
      tMinus = 0;
      DLL_tminus_zero_indication();
      }
   }
//
// maintenance work done whenever the moderator is received
//
void processModerator(moderatorLpacket &moderator)
   {
   //
   // copy data from the moderator
   // which cannot cause a rogue condition
   //
   interval_count = moderator.interval_count;
   usr = moderator.usr;
   tMinus = moderator.tMinus;

   lowcount = LOWCOUNTINIT; // re-initialize lowman counter

   // resynchronize the NUT timer to the moderator
   NUT_timer.count = DLL_SM_current.gb_center - MODERATOR_LENGTH/10;

   if(netState == MOD)             // if this node is moderator
      {                  // and a moderator was received!
      netState = NOTMOD;           // drop moderatorship
      modcount = 0;             // init count of missing moderators
      DLL_event(DLL_EV_nonconcurrence);      // indicate non-concurrence
      }
   else
      {
      if(!in_guardband)            // unexpected moderator
         {
         DLL_event(DLL_EV_nonconcurrence);   // indicate non-concurrence
         }

      if(netState == WATCH)           // got moderator, there is a network
         {
         netState = DUPCHECK;         // go check for dupnodes
         modcount = modcountinit;     // this is how many NUTs to check
         }
      else if(netState == DUPCHECK)       // if checking for duplicate MAC IDs
         {
         if(SM_listen_only == FALSE)      // unless this node is required to
            {            // remain in this state
```

```
            modcount--;         // decrement NUT counter
            if(modcount == 0)      // when done
               {
               netState = NOTMOD;  // it's OK to go transmit
               }
            }
         }
      else if(netState == ROGUE)       // a rogue that sees a good moderator
         {
         netState = DUPCHECK;          // can exit rogue state
         modcount = modcountinit;
         }
      else
         {
         modcount = 0;       // else just reset mod counter
         }
      }
   }
//
// recovery after a time-out while waiting for moderator
//
void missed_moderator(void)
   {
   if(netState == NOTMOD)        // if supposed to be on-line and hearing moderators
      {
      modcount++;            // count consecutive missing moderator
      if(modcount >= TXNOMOD)       // if exceeded limit
         {
         netState = DUPCHECK;       // change to a quietly waiting mode
         modcount = 1;       // init mods required before return on-line
         }

      if(SM_listen_only == FALSE    // if allowed to transmit
       && SM_mod_enable == TRUE     // and allowed to be moderator
       && DLL_SM_current.myaddr <= nqlowman) // and apparently the lowman
         {
         lowcount--;         // count consecutive NUTs as lowman
         if(lowcount == 0)      // if this is the second time
            {
            netState = MOD; // then activate the moderator on third NUT
            }
         }
      else
         {
         lowcount = LOWCOUNTINIT;  // otherwise, re-initialize lowman counter
         }
      }
   }
//
// housekeeping actions at the end of each NUT
//
void housekeeping(void)
   {
   if(netState == WATCH)            // in watch state
      {
      lonely = LONELYINIT;          // hold the lonely counter reset
      if(watch_timer.count == 0)    // if timed out (no moderators)
         {            // change to on-line -- try to build a link
         if (SM_listen_only == FALSE  // if allowed to transmit
          && SM_mod_enable == TRUE // and allowed to moderate
          && nqlowman >= DLL_SM_current.myaddr) // and lowman
            {
            netState = MOD; // activate moderator state
            }
         else
            {
            netState = DUPCHECK;   // else wait for another node to moderate
            modcount = modcountinit;
            }
         }

      }
   else if (qlowman == 255          // no good FCS received in the last NUT
       && DLL_SM_current.myaddr!= 0// local node is not supernode
```

```
        && netState!= WATCH)        // not already in the watch state
       {
       lonely--;
       if(lonely == 0 || netState == ROGUE || netState == DUPNODE)
          {
          DLL_event(DLL_EV_lonely);
          if(netState == ROGUE)
              {
              watch_timer.start(0,75 sec);
              }
          else
              {
              watch_timer.start(1,25 sec);
              }
          netState = WATCH;
          }
       }
    else
       {
       lonely = LONELYINIT;
       }

    if(deafcount >= DEAFNESS)
       {
       deafcount = 0;
       DLL_event(DLL_EV_deafness);
       NUT_timer.count += DEAFADJUST;
       }

    // this counter determines how many NUTS a node should
    // check the link before deciding that it is not
    // a dupnode. if 0<myaddr<=SMAX a node is guaranteed to
    // an access each NUT, so the check time is low. On the
    // other hand, if smax<myaddr<=umax, a node might have to wait
    // a long time for its access slot come up, so use a long
    // time. If a DUPCHECKing unscheduled only node ever sees its slot
    // go by, it sets modcount to a lower number immediately.

    // nodes between SMAX and UMAX aren't guaranteed to get a slot every NUT
    // if not an unscheduled only node, use a low number else use a huge number

    if (DLL_SM_current.myaddr > DLL_SM_current.smax)
       modcountinit = 255;
    else
       modcountinit =TXDUPCHECKS;

    // update the interval count once per NUT
    interval_count = (interval_count + 1) % DLL_SM_current.modulus;

    // update the usr once per NUT
    usr = (usr + 1) % (DLL_SM_current.umax + 1);
    }

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//
// The ACM State Machine
//
state: powerup

    event: SM_powerup
    destination: offline

//
// be idle until told to go on-line
//
state: offline

    // normal case
    event: SM_online == TRUE
    condition: ph_frame_indication == FALSE
    destination: rxMod0
    action:
```

```
        in_guardband = TRUE; // start up at the beginning of a guardband
        if(DLL_SM_current.myaddr == 0) // either local node is supernode
            {
            netState = MOD;
            SM_listen_only = FALSE;
            }
        else // or not a supernode
            {
            netState = WATCH;
            watch_timer.start(1,25 sec);
            }
        scheduled = FALSE;
        in_guardband = TRUE;
        DLL_online_confirm(SM_online);


    // exception: there was an Lpacket on the wire at transition to on-line
    event: SM_online == TRUE
    condition: ph_frame_indication == TRUE
    destination: badMod // treat it as an error inside the guardband
    action:
        DLL_event(DLL_EV_badFrame,macSrce);
        in_guardband = TRUE;
        if(DLL_SM_current.myaddr == 0)
            {
            netState = MOD;
            SM_listen_only = FALSE;
            }
        else
            {
            netState = WATCH;
            watch_timer.start(1,25 sec);
            }
        DLL_online_confirm(SM_online);
//
// wait for either energy on the wire or a slot time-out
//
state: waitFrame

    event:    ph_lock_indication == TRUE
    destination: frEst1
    action:      gen_timer.begin_counting();

    event:    slot_timer.expired()
    destination: gap
    action:      gen_timer.start(0,6 usec);

//
// wait for frame start, or noise
//
state: frEst1
    event:    ph_frame_indication == TRUE
    destination: rxFrame

    event:    ph_lock_indication == FALSE      // short energy burst == noise blip
    destination: waitFrame
    action:      DLL_event(DLL_EV_noisePulse);    // ignore it

    event:    gen_timer.count >= 8,0 usec
    destination: frEst2
//
// continue waiting
//
state: frEst2

    event:    ph_frame_indication == TRUE
    destination: rxFrame

    event:    ph_lock_indication == FALSE      // longer burst treat as a damaged DLPDU
    destination: endFrame
    action:
        DLL_event(DLL_EV_badFrame,macSrce);
        gen_timer.start(5,2 usec);
```

```
   event:    gen_timer.count >= 11,2 usec     // energy this long with no data start
   destination: endFrame           // treat as bad DLPDU, no start delimiter
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);

//
// wait here for there to be no detectable energy on the wire (no clock lock)
//
state: endFrame

   event:    ph_lock_indication == FALSE
   destination: gap
   action:
      slot_timer.restart();

//
// simulate first half of node turnaround time with this timing gap
//
state: gap

   event:    ph_frame_indication == TRUE  // DLPDU start overrides all
   destination: rxFrame

   event:    gen_timer.expired()    // process the rest of
   destination: nextSlot
   action:
      gen_timer.start(6.1 usec);   // start second half of turnaround

      if(scheduled)          // do implicit token rotation, scheduled
         {
         itr++;
         if(itr>DLL_SM_current.smax)  // after SMAX,
            {         // change from scheduled to unscheduled
            itr=usr;
            scheduled = FALSE;
            }
         }
      else               // implicit token rotation, unscheduled
         {
         itr = (itr + 1) % (DLL_SM_current.umax + 1);
         }

//
// decide what comes next
//
state: nextSlot

   event:    ph_frame_indication == TRUE     // detect a DLPDU start at all times
   destination: rxFrame

   // time for the guardband
   event:    gen_timer.expired()
   condition:   NUT_timer.count < DLL_SM_current.gb_prestart
   destination: guardBand
   action:

      // scheduled time not completed -- log the error and proceed
      if(scheduled == TRUE)
         {
         DLL_event(DLL_EV_NUT_overrun);
         scheduled = FALSE;
         }
      in_guardband = TRUE;

   // not this node's slot, keep counting.
   // Note that the slot timer auto repeats to maintain regular slot intervals
   event:    gen_timer.expired()
   condition:   NUT_timer.count >= DLL_SM_current.gb_prestart
         && itr!= DLL_SM_current.myaddr
   destination: waitFrame

   // this node's slot, but not allowed to talk
```

```
    event:    gen_timer.expired()
    condition:   NUT_timer.count >= DLL_SM_current.gb_prestart
          && itr == DLL_SM_current.myaddr
          &&!(netState == MOD || netState == NOTMOD)
    destination: waitFrame
    action:      modcount = min(modcount, TXDUPCHECKS); // optimize dupcheck

    // this node's slot, and allowed to talk
    event:    gen_timer.expired()
    condition:   NUT_timer.count >= DLL_SM_current.gb_prestart
          && itr == DLL_SM_current.myaddr
          && (netState==MOD || netState==NOTMOD)
    destination: waitForHold


//
// a DLPDU has started, this deals with the data
//
state: rxFrame

    // prematurely lost the DLPDU event
    event:    ph_frame_indication == FALSE
    destination: endFrame
    action:
       DLL_event(DLL_EV_badFrame,macSrce);
       gen_timer.start(5,2 usec);

       // start hold timer after line is quiet
       holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

    // duplicate node condition
    event:    RX_dataReady
    condition:   RX_rxData == DLL_SM_current.myaddr
    destination: dupNode
    action:
       macSrce = RX_rxData;
       nqlowman = min(nqlowman,macSrce);

    // record the source MAC ID
    event:    RX_dataReady
    condition:   RX_rxData!= DLL_SM_current.myaddr
    destination: nextLpacket
    action:
       macSrce = RX_rxData;
       nqlowman = min(nqlowman,macSrce);

//
// start here to parse each new Lpacket
// the RxM delivers one complete Lpacket at a time
//
state: nextLpacket

    // received an out-of-sequence moderator -- attempt resync
    event:    RX_receivedLpacket
    condition:   RX_Lpacket.service == MODERATOR_TAG
    destination: rxMod3              // unexpected moderator

    // ignore most other received Lpackets
    event:    RX_receivedLpacket
    condition:   RX_Lpacket.service!= MODERATOR_TAG
    destination: nextLpacket

    // lost the DLPDU in the middle
    event:    ph_frame_indication == FALSE
    destination: endFrame
    action:
       DLL_event(DLL_EV_badFrame,macSrce);
       gen_timer.start(5,2 usec);
       // start hold timer after line is quiet
       holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

    // the DLPDU was aborted
    event: RX_abort
    destination: endFrame
```

```
   action:
      DLL_event(DLL_EV_rxAbort);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // if bad FCS, or DLPDU extended into the guardband, treat as bad DLPDU
   event:    RX_endMAC
   condition:   NUT_timer.count <= DLL_SM_current.gb_start ||!RX_FCSOK
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // good DLPDU, and not guardband time
   // use the source ID
   event:    RX_endMAC
   condition:   NUT_timer.count > DLL_SM_current.gb_start && RX_FCSOK
   destination: endFrame
   action:
      qlowman = min(qlowman, macSrce);
      DLL_event(DLL_EV_rxGoodFrame);
      if(itr!= macSrce) DLL_event(DLL_EV_nonconcurrence);
      itr = macSrce;
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// deal with a duplicate node indication, local MAC ID already in by another node
//
state: dupNode

   // bad DLPDU so ignore indication
   event:    ph_frame_indication == FALSE
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // bad DLPDU so ignore indication
   event:    RX_endMAC
   condition:   !RX_FCSOK
   destination: endFrame
   action:
      DLL_event(DLL_EV_badFrame,macSrce);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // good DLPDU so accept the duplicate indication
   event:    RX_endMAC
   condition:   RX_FCSOK
   destination: endFrame
   action:
      netState = DUPNODE;
      if (!dupflag)
         {
         dupflag == TRUE;
         DLL_event(DLL_EV_dupNode);
         }
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// Hold off transmission until blanking + 1 octet has expired
// since the last transmit or receive. It is possible that the
// hold timer expires before the gen_timer, dealing with turnaround time.
```

```
//
state: waitForHold

   // time to transmit, but line has activity, treat it as a collision, and back off.
   event:    holdoff_timer.expired()
   condition:   ph_lock_indication == TRUE
   destination: endFrame
   action:
      DLL_event(DLL_EV_collision);
      gen_timer.start(0,6 usec);

   // no problem, start transmission
   event:    holdoff_timer.expired()
   condition:   ph_lock_indication == FALSE
   destination: sendHeader
   action:
      // calculate time remaining in this NUT
      tmp = (unsigned)(NUT_timer.count-DLL_SM_current.gb_prestart);
      octetsleft = min(255, (tmp<<1) + tmp + (tmp>>3)) * 2;
      TX_sendHeader(DLL_SM_current.myaddr);

//
// when header has been sent, start first Lpacket
//
state: sendHeader

   // TxLLC has something, send a new Lpacket
   event:    TX_headerComplete
   condition:   LLC_pickLpacket(scheduled, octetsleft) == TRUE
   destination: sendNextLpacket
   action:
      TX_sendLpacket(LLC_Lpacket);
      octetsleft = octetsleft - LLC_Lpacket.wire_size();

   // nothing to send that fits in the time left, close the DLPDU
   event:    TX_headerComplete
   condition:   LLC_pickLpacket(scheduled,octetsleft) == FALSE
   destination: sendTrailer
   action:
      TX_sendTrailer();
      if(scheduled && fifo[SCHEDULED].has_data())
         {
         DLL_event(DLL_EV_dribble);
         }

//
// send subsequent Lpackets
//
state: sendNextLpacket

   // last Lpacket was aborted, end frame
   // note that the TXM has already sent the abort sequence on the wire
   event:       TX_LpacketComplete
   condition:   Tx_abort==TRUE
   destination: endFrame
   action:
      DLL_event(DLL_EV_txAbort);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

   // TxLLC has something, send a new Lpacket
   event:    TX_LpacketComplete
   condition:   (TX_abort == FALSE) && (LLC_pickLpacket(scheduled,octetsleft) == TRUE)
   destination: sendNextLpacket
   action:
      TX_sendLpacket(LLC_Lpacket);
      octetsleft = octetsleft - LLC_Lpacket.wire_size();

   // nothing to send that fits in the time left, close the DLPDU
   event:    TX_LpacketComplete
   condition:   (TX_abort == FALSE)&&(LLC_pickLpacket(scheduled,octetsleft) == FALSE)
   destination: sendTrailer
   action:
```

```
      TX_sendTrailer();

//
// complete DLPDU termination
//
state: sendTrailer

   event:    TX_trailerComplete
   destination: endFrame
   action:
      DLL_event(DLL_EV_txGoodFrame);
      gen_timer.start(5,2 usec);
      // start hold timer after line is quiet
      holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// almost guardband start time, wait for guardband start then check lots of things
//
state: guardBand

   event:    ph_frame_indication == TRUE      // received DLPDU takes precedence
   destination: rxFrame               // deal with it

   // net change not in progress, this node is not moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start // guardband start time NOW
   condition:   tMinus == 0
         && SM_listen_only == SM_listen_only_req
         && netState!= MOD
   destination: rxMod0

   // net change not in progress, this node is moderator AND can remain moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus == 0
         && SM_listen_only == SM_listen_only_req
         && netState == MOD
         && SM_mod_enable == TRUE
         && min(DLL_SM_current.myaddr, qlowman) == DLL_SM_current.myaddr
   destination: txMod0

   // net change not in progress, this node is moderator, but cannot remain moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus == 0
         && SM_listen_only == SM_listen_only_req
         && netState == MOD
         && (SM_mod_enable == FALSE
          || min(DLL_SM_current.myaddr, qlowman)!= DLL_SM_current.myaddr)
   destination: rxMod0
   action:
      netState = NOTMOD;
      modcount = 0;
      lowcount = LOWCOUNTINIT;

   // net change in progress, this node is not moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus > 1 && SM_listen_only == SM_listen_only_req && netState!= MOD
   destination: rxMod0
   action:
      tMinus = tMinus - 1;

   // net change in progress, this node is moderator, and can remain moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus > 1
         && SM_listen_only == SM_listen_only_req
         && netState == MOD
         && SM_mod_enable == TRUE
         && min(DLL_SM_current.myaddr, qlowman) == DLL_SM_current.myaddr
   destination: txMod0
   action:
      tMinus = tMinus - 1;

   // net change in progress, this node is moderator, but cannot remain moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus > 1
```

```
          && SM_listen_only == SM_listen_only_req
          && netState == MOD
          && (SM_mod_enable == FALSE
           || min(DLL_SM_current.myaddr, qlowman)!= DLL_SM_current.myaddr)
   destination: rxMod0
   action:
      tMinus = tMinus - 1;
      netState = NOTMOD;
      modcount = 0;
      lowcount = LOWCOUNTINIT;

   // net change complete, this node becomes supernode
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   tMinus == 1 && DLL_SM_pending.myaddr == 0
   destination: txMod0
   action:
      handle_net_change();
      netState = MOD;
      SM_listen_only = FALSE;          // override listen_only

   // net change complete, but shall stop transmitting for a time
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   ( tMinus == 1
          && DLL_SM_pending.myaddr!= 0
          && DLL_SM_pending.myaddr!= DLL_SM_current.myaddr
          || SM_listen_only!= SM_listen_only_req
          && SM_listen_only_req)
   destination: rxMod0
   action:
      handle_net_change();
      netState = WATCH;          // go to WATCH state
      watch_timer.start(1,25 sec);

   // net change complete, this node is not moderator, no significant changes
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   ( tMinus == 1
          && DLL_SM_pending.myaddr!= 0
          && DLL_SM_pending.myaddr == DLL_SM_current.myaddr
          || SM_listen_only!= SM_listen_only_req
          &&!SM_listen_only_req)
         && netState!= MOD
   destination: rxMod0
   action:
      handle_net_change();

   // net change complete, this node is moderator, no major changes, and can remain
moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   ( tMinus == 1
          && DLL_SM_pending.myaddr!= 0
          && DLL_SM_pending.myaddr == DLL_SM_current.myaddr
          || SM_listen_only!= SM_listen_only_req
          &&!SM_listen_only_req)
         && netState == MOD
         && SM_mod_enable == TRUE
         && min(DLL_SM_pending.myaddr, qlowman) == DLL_SM_pending.myaddr
   destination: txMod0
   action:
      handle_net_change();

   // net change complete, this node is moderator, no major changes, but can't stay
moderator
   event:    NUT_timer.count <= DLL_SM_current.gb_start
   condition:   ( tMinus == 1
          && DLL_SM_pending.myaddr!= 0
          && DLL_SM_pending.myaddr == DLL_SM_current.myaddr
          || SM_listen_only!= SM_listen_only_req
          &&!SM_listen_only_req)
         && netState == MOD
         && (SM_mod_enable == FALSE
          || min(DLL_SM_pending.myaddr, qlowman)!= DLL_SM_pending.myaddr)
   destination: rxMod0
   action:
      handle_net_change();
```

```
        netState = NOTMOD;
        modcount = 0;
        lowcount = LOWCOUNTINIT;


//
// wait for moderator start
//
state: rxMod0

    // got it
    event:    ph_frame_indication == TRUE
    destination: rxMod1

    // timed out waiting for moderator
    event:    NUT_timer.count <= 20 usec
    destination: waitTone
    action:
        missed_moderator();
        housekeeping();

//
// receive the moderator
//
state: rxMod1

    // bad moderator
    event:    ph_frame_indication == FALSE
    destination: badMod
    action:
        DLL_event(DLL_EV_badFrame,macSrce);

    // get MAC source ID
    event:    RX_dataReady == TRUE
    destination: rxMod2
    action:
        macSrce = RX_rxData;
        nqlowman = min(nqlowman,macSrce);

//
// continue receiving moderator
//
state: rxMod2

    // bad DLPDU
    event:    ph_frame_indication == FALSE
    destination: badMod
    action:
        DLL_event(DLL_EV_badFrame,macSrce);

    // bad moderator DLPDU if it's null
    event:    RX_endMAC
    destination: badMod
    action:
        DLL_event(DLL_EV_badFrame,macSrce);

    // got an Lpacket, save the Lpacket for later, and go check the rest of the DLPDU
    event:    RX_receivedLpacket
    destination: rxMod3

//
// handle the end of a moderator DLPDU
//
state: rxMod3

    // bad DLPDU
    event:    ph_frame_indication == FALSE
    destination: badMod
    action:
        DLL_event(DLL_EV_badFrame,macSrce);

    // bad DLPDU
    event:    RX_abort
```

```
destination: badMod
action:
   DLL_event(DLL_EV_rxAbort);

// bad DLPDU if another Lpacket is in the moderator DLPDU
event:    RX_receivedLpacket
destination: badMod
action:
   DLL_event(DLL_EV_badFrame,macSrce);

// bad DLPDU
event:    RX_endMAC
condition:   !RX_FCSOK
destination: badMod
action:
   DLL_event(DLL_EV_badFrame,macSrce);

// dupnode!
event:    RX_endMAC
condition:   RX_FCSOK && macSrce == DLL_SM_current.myaddr
destination: endFrame
action:
   netState = DUPNODE;
   if(!dupflag)
      {
      dupflag = TRUE;
      DLL_event(DLL_EV_dupNode);
      }
   gen_timer.start(5,2 usec);
   // start hold timer after line is quiet
   holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// good DLPDU, but bad header
event:    RX_endMAC
condition:   RX_FCSOK
      && macSrce!= DLL_SM_current.myaddr
      && ( RX_Lpacket.size!= MODERATOR_SIZE
       || RX_Lpacket.ctl!= MODERATOR_CTL
       || RX_Lpacket.service!= MODERATOR_TAG
       || RX_Lpacket.dest!= 0xFF)
destination: badMod

// moderator, but from an incorrect address (not lowman)
event:    RX_endMAC
condition:   RX_FCSOK
      && macSrce!= DLL_SM_current.myaddr
      && RX_Lpacket.size == MODERATOR_SIZE
      && RX_Lpacket.ctl == MODERATOR_CTL
      && RX_Lpacket.service == MODERATOR_TAG
      && RX_Lpacket.dest == 0xFF
      && macSrce > qlowman
destination: endFrame
action:
   DLL_event(DLL_EV_invalidModAddress);
   gen_timer.start(5,2 usec);
   // start hold timer after line is quiet
   holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// this is a moderator from a supernode,
// or this node is at 0xFF so that all others look like a supernode
// adopt all the parameters from the moderator
event:    RX_endMAC
condition:   RX_FCSOK
      && macSrce!= DLL_SM_current.myaddr
      && RX_Lpacket.size == MODERATOR_SIZE
      && RX_Lpacket.ctl == MODERATOR_CTL
      && RX_Lpacket.service == MODERATOR_TAG
      && RX_Lpacket.dest == 0xFF
      && macSrce <= qlowman
      && (macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
destination: waitTone
action:
   qlowman = min(qlowman, macSrce);
   if (macSrce == 0 && currentMod!= 0) DLL_event(DLL_EV_supernode);
```

```
      currentMod = macSrce;
      DLL_currentMod_indication(currentMod);
      DLL_SM_current.NUT_length = moderator.NUT_length;
      DLL_SM_current.gb_prestart = moderator.gb_prestart;
      DLL_SM_current.gb_start = moderator.gb_start;
      DLL_SM_current.gb_center = moderator.gb_center;
      DLL_SM_current.modulus = moderator.modulus;
      DLL_SM_current.blanking = moderator.blanking;
      DLL_SM_current.slotTime = moderator.slotTime;
      DLL_SM_current.smax = moderator.smax;
      DLL_SM_current.umax = moderator.umax;
      processModerator(moderator);
      housekeeping();

   // good moderator received, everything matches local copy
   // resynchronize and continue
   event:    RX_endMAC
   condition:   RX_FCSOK
          && macSrce!= DLL_SM_current.myaddr
          && RX_Lpacket.size == MODERATOR_SIZE
          && RX_Lpacket.ctl == MODERATOR_CTL
          && RX_Lpacket.service == MODERATOR_TAG
          && RX_Lpacket.dest == 0xFF
          && macSrce <= qlowman
          &&!(macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
          && moderator.NUT_length == DLL_SM_current.NUT_length
          && moderator.gb_prestart == DLL_SM_current.gb_prestart
          && moderator.gb_start == DLL_SM_current.gb_start
          && moderator.gb_center == DLL_SM_current.gb_center
          && moderator.modulus == DLL_SM_current.modulus
          && moderator.blanking == DLL_SM_current.blanking
          && moderator.slotTime == DLL_SM_current.slotTime
          && moderator.smax == DLL_SM_current.smax
          && moderator.umax == DLL_SM_current.umax
   destination: waitTone
   action:
      qlowman = min(qlowman, macSrce);
      currentMod = macSrce;
      DLL_currentMod_indication(currentMod);
      processModerator(moderator);
      housekeeping();

   // The moderator doesn't match, so this node is a rogue.
   // Since in the guardband, go to rxmod for recovery
   event:    RX_endMAC
   condition:   RX_FCSOK
          && macSrce!= DLL_SM_current.myaddr
          && RX_Lpacket.size == MODERATOR_SIZE
          && RX_Lpacket.ctl == MODERATOR_CTL
          && RX_Lpacket.service == MODERATOR_TAG
          && RX_Lpacket.dest == 0xFF
          && macSrce <= qlowman
          &&!(macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
          &&!(    moderator.NUT_length == DLL_SM_current.NUT_length
             && moderator.gb_prestart == DLL_SM_current.gb_prestart
             && moderator.gb_start == DLL_SM_current.gb_start
             && moderator.gb_center == DLL_SM_current.gb_center
             && moderator.modulus == DLL_SM_current.modulus
             && moderator.blanking == DLL_SM_current.blanking
             && moderator.slotTime == DLL_SM_current.slotTime
             && moderator.smax == DLL_SM_current.smax
             && moderator.umax == DLL_SM_current.umax)
          && in_guardband == TRUE
   destination: rxMod0
   action:
      qlowman = min(qlowman, macSrce);
      currentMod = macSrce;
      DLL_currentMod_indication(currentMod);
      netState = ROGUE;
      DLL_event(DLL_EV_rogue);

   // The moderator doesn't match, so this node is a rogue.
   // Since currently not in the guardband, go to endframe for recovery.
```

```
    event:    RX_endMAC
    condition:   RX_FCSOK
           && macSrce!= DLL_SM_current.myaddr
           && RX_Lpacket.size == MODERATOR_SIZE
           && RX_Lpacket.ctl == MODERATOR_CTL
           && RX_Lpacket.service == MODERATOR_TAG
           && RX_Lpacket.dest == 0xFF
           && macSrce <= qlowman
           &&!(macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
           &&!(    moderator.NUT_length == DLL_SM_current.NUT_length
              && moderator.gb_prestart == DLL_SM_current.gb_prestart
              && moderator.gb_start == DLL_SM_current.gb_start
              && moderator.gb_center == DLL_SM_current.gb_center
              && moderator.modulus == DLL_SM_current.modulus
              && moderator.blanking == DLL_SM_current.blanking
              && moderator.slotTime == DLL_SM_current.slotTime
              && moderator.smax == DLL_SM_current.smax
              && moderator.umax == DLL_SM_current.umax)
           && in_guardband == FALSE
    destination: endFrame
    action:
       qlowman = min(qlowman, macSrce);
       currentMod = macSrce;
       DLL_currentMod_indication(currentMod);
       netState = ROGUE;
       DLL_event(DLL_EV_rogue);
       gen_timer.start(5,2 usec);
       // start hold timer after line is quiet
       holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// come here to handle a bad moderator DLPDU
// wait for link to become quiet, or until tone generation is due
//
state: badMod

    // This state is reached when this node identifies a moderator DLPDU,
    // but the DLPDU is subsequently reported bad so the state transfers
    // back to endFrame if this node is not in the guardband.
    event: TRUE
    condition:   in_guardband == FALSE
    destination: endFrame
    action:
       DLL_event(DLL_EV_badFrame,macSrce);
       gen_timer.start(5,2 usec);
       // start hold timer after line is quiet
       holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

    // the guardband should be ended NOW
    event:    NUT_timer.count <= 30 usec
    destination: waitTone
    action:
       missed_moderator();
       housekeeping();

    // else wait until the link is quiet
    event:    ph_lock_indication == FALSE
    destination: rxMod0


//
// send the moderator, wait for guardband center
//
state: txMod0

    event:    NUT_timer.count <= DLL_SM_current.gb_center
    destination: txMod1
    action:
       TX_sendHeader(DLL_SM_current.myaddr); // send header

//
// send more
//
state: txMod1
```

```
   event:    TX_headerComplete
   destination: txMod2
   action:
      moderator.size = MODERATOR_SIZE;
      moderator.ctl = MODERATOR_CTL;
      moderator.service = MODERATOR_TAG;
      moderator.dest = 0xff;
      moderator.NUT_length = DLL_SM_current.NUT_length;
      moderator.smax = DLL_SM_current.smax;
      moderator.umax = DLL_SM_current.umax;
      moderator.slotTime = DLL_SM_current.slotTime;
      moderator.blanking = DLL_SM_current.blanking;
      moderator.gb_start = DLL_SM_current.gb_start;
      moderator.gb_center = DLL_SM_current.gb_center;
      moderator.usr = usr;
      moderator.interval_count = interval_count;
      moderator.modulus = DLL_SM_current.modulus;
      moderator.tMinus = tMinus;
      moderator.gb_prestart = DLL_SM_current.gb_prestart;
      moderator.spare = 0;

      TX_sendLpacket(moderator); // send the Lpacket

//
// finish up
//
state: txMod2

   event: TX_LpacketComplete
   destination: txMod3
   action:
      TX_sendTrailer();

//
// at completion, transferring to waiting for tone
//
state: txMod3

   event:    TX_trailerComplete
   destination: waitTone
   action:
      currentMod = DLL_SM_current.myaddr;
      DLL_currentMod_indication(currentMod);
      housekeeping();


//
// wait for tone
//
state: waitTone

   // end of this NUT, and transferring to off-line
   event:    NUT_timer.count == 0
   condition:   SM_online == FALSE
   destination: offline
   action:
      DLL_tone_indication(interval_count);
      DLL_online_confirm(SM_online);


   // end of this NUT, start of another
   event:    NUT_timer.count == 0
   condition:   SM_online_req == TRUE
   destination: waitSlotZero
   action:
      DLL_tone_indication(interval_count);
      NUT_timer.restart();
      scheduled = TRUE;
      in_guardband = FALSE;

      // housekeeping
```

```
      // If this node detects line activity, but not moderators, attempt to recover.
      // Maybe problem is that the local node is performing housekeeping (and is
      // therefore deaf) during the time the moderator DLPDU is transmitted.

      if (!(netState == MOD || netState == NOTMOD)
         && qlowman!= 255
         && ph_frame_indication == TRUE)
         {
         deafcount++;
         }
      else
         {
         deafcount = 0;
         }

      gen_timer.start(20 usec); // start timer for start of scheduled time

//
// wait for start of scheduled
//
state: waitSlotZero

   // start a new scheduled token pass
   event:    gen_timer.expired()
   destination: gap
   action:
      itr = -1;
      if (DLL_SM_current.myaddr == 0)  // supernode doesn't try to detect lowman
         {
         nqlowman = 0;
         qlowman = 0;
         }
      else                // else initialize the detector
         {
         nqlowman = 255;
         qlowman = 255;
         }
      slot_timer.restart();
      gen_timer.start(0,6 usec);
```

### 9.3 TxLLC

Le TxLLC (transmission LLC) doit recevoir et mettre en mémoire tampon les Lpackets provenant des couches supérieures. Il doit choisir le Lpacket suivant à transmettre en fonction

– de l'ordre de mise en file d'attente des Lpackets;

– des attributs du Lpacket;

– des informations fournies par la machine de contrôle d'accès (ACM).

Le TxLLC doit présenter la sélection de Lpacket à l'ACM pour la transmission.

```
// DLL TX LLC State Machine Description

// This state machine accepts transmit requests from the DLS-user,
// queues and prioritizes them, and submits one Lpacket at a time
// to the ACM based on parameters received from the ACM.

////////////////////////////////////////////////////////////////////////
//
// type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;
typedef void *IDENTIFIER;

typedef enum {  M_0,
      M_1,
      M_ND_plus,
      M_ND_minus } M_SYMBOL;

// These are the three transmit priorities.
// hard assignments are so that the priority can be
```

```
// used as an index into an array of FIFOs
// note that HIGH and LOW are unscheduled

typedef enum {  SCHEDULED=0,
       HIGH=1,
       LOW=2} PRIORITY;

typedef enum {  OK,
       TXABORT,
       FLUSHED } TXSTATUS; // describes the result of a transmit request.

//
// Lpacket class
//
class Lpacket
   {
   public:

   // return the size octet of the current Lpacket
   USINT size;

   // return the ctl octet of the current Lpacket
   USINT ctl;

// Lpacket constants: masks for ctl octet

#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }

   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }

   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }

   // store next octet to the Lpacket
   void put_octet(USINT data);

   // get an octet from the Lpacket
   USINT &operator[](int index);

   Lpacket(void *p)       // constructor
      {
      }

   Lpacket(int size)         // constructor
      {
      }
   };

//
// superclass of Lpacket that defines an Lpacket to be transmitted
//
class txLpacket: public Lpacket
   {
   public:

   IDENTIFIER id;
   BOOL fixed;
```

```
   // constructors

   txLpacket(void *p): Lpacket(p)
      {
      }

   txLpacket(int size): Lpacket(size)  // size is size of PDU buffer
      {            // (Lpacket plus pads, in octets)
      }
   };


////////////////////////////////////////////////////////////////////////
//
// interface to DLS-user
//
void DLL_xmit_fixed_request(
         IDENTIFIER id,
       USINT Lpacket[],
         UINT size,
         PRIORITY priority,
         USINT service,
         USINT destID);

extern void DLL_xmit_fixed_confirm (IDENTIFIER id, TXSTATUS status);

void DLL_xmit_generic_request(
IDENTIFIER id,
         USINT Lpacket[],
         UINT size,
         PRIORITY priority,
         USINT tag[3]);

extern void DLL_xmit_generic_confirm (IDENTIFIER id, TXSTATUS status);

void DLL_flush-requests-by-QoS_request (PRIORITY priority);

extern void DLL_flush-requests-by-QoS_confirm (PRIORITY priority);

DLL_flush_single_request( PRIORITY priority, IDENTIFIER xmit_id );


////////////////////////////////////////////////////////////////////////
//
// interface to ACM
//
txLpacket *pickLpacket(BOOL scheduled, int octetsLeft);// pick an Lpacket to send next
void LpacketSent(TXSTATUS status);        // the Lpacket was sent


////////////////////////////////////////////////////////////////////////
//
// interface to station management
//
void SM_powerup(void);          // input indication: powerup has occurred


////////////////////////////////////////////////////////////////////////
//
// a class to represent message FIFOs
//
class FIFO
   {
   public:

   void put(txLpacket lp);      // add an Lpacket to the FIFO
   txLpacket &get(void);        // remove an Lpacket from the FIFO
   txLpacket &peek(void);       // look at the first Lpacket in the FIFO
   void flush(void);         // delete all Lpackets in the FIFO
   void flush1(IDENTIFIER id);     // delete one Lpacket in the FIFO, given it's ID
   BOOL has_data();       // true if the FIFO is not empty
   };

FIFO fifo[3];             // at least three priority levels shall be provided
```

```
////////////////////////////////////////////////////////////////////////////
//
// TxLLC implementation
//

//
// powerup initialization
//
void SM_powerup(void)
   {
   fifo[SCHEDULED].flush();
   fifo[HIGH].flush();
   fifo[LOW].flush();
   }

//
// flush a particular FIFO
//
void DLL_flush-requests-by-QoS_request (PRIORITY priority)
   {
   fifo[priority].flush();
   DLL_flush-requests-by-QoS_confirm(priority);
   }

//
// flush a particular Lpacket
//
void DLL_flush_single_request (PRIORITY priority, IDENTIFIER id)
   {
   fifo[priority].flush1(id);
   }

//
// requests from DLS-user to submit Lpackets for transmission
//
void DLL_xmit_fixed_request(
       IDENTIFIER id,
       USINT Lpacket[],
       UINT size,
       PRIORITY priority,
       USINT service,
       USINT destID)
   {
   int tmp;

   tmp = size + 4 + (size&1);   // size of DLSdu plus 4 octet fixed Lpacket header
                  // plus optional data pad

   txLpacket &lp(new txLpacket(tmp));  // create a new Lpacket buffer

   // build the PDU (Lpacket)
   lp[0] = tmp/2;            // size, in words
   lp[1] = 0x01 | ((size&1)<<2);   // ctl octet, plus data pad bit, if needed
   lp[2] = service;       // fixed tag service
   lp[3] = destID;        // destination MAC ID
   memcpy(&lp[4], Lpacket, size);      // copy the rest of the data
                  // there may be a pad octet sent after the DLSdu
   lp.id = id;            // store the user's id (for the confirmation)
   lp.fixed = TRUE;       // store type of Lpacket (fixed)
   fifo[priority].put(lp);       // queue the Lpacket
   }


void DLL_xmit_generic_request(
IDENTIFIER id,
         USINT Lpacket[],
         UINT size,
         PRIORITY priority,
         USINT tag[3])
   {
   int tmp;
```

```
   int pad;

   tmp = size + 6 + (size&1);    // octet size of DLSdu plus 6 octet GEN Lpacket header
                    //      plus data pad
   pad = 0;

   txLpacket &lp(new txLpacket(tmp));  // create a new Lpacket buffer

   // build the PDU (Lpacket)
   lp[0] = tmp/2;             // size, in words
   lp[1] = 0x12 | ((size&1)<<2);    // control octet,
                    // plus maybe a data pad bit,


   lp[2] = 0;              // tag pad octet (affects memory image of Lpacket)
   lp[3] = tag[0];         // generic tag
   lp[4] = tag[1];         // generic tag
   lp[5] = tag[2];         // generic tag
   memcpy(&lp[6], Lpacket, size);      // copy the rest of the data
                    // there may be pad octet after the DLSdu

   lp.id = id;             // store the user's identifier (for the confirm)
   lp.fixed = FALSE;          // store type of Lpacket (generic)

   fifo[priority].put(lp);       // queue the new Lpacket
   }


static txLpacket *picked = 0;       // remember which Lpacket was picked

//
// called by ACM to pick the next Lpacket to be sent out
//
txLpacket *pickLpacket(BOOL scheduled, int octetsLeft)
   {
   int wordsleft = (octetsLeft+1)/2;   // wordsleft is rounded up, accepting that in
some
                    // cases this results in an Lpacket not being sent

   if(scheduled)           // if scheduled, only look at scheduled FIFO
      {
      // if there is anything in the FIFO and it fits in the time left
      if(fifo[SCHEDULED].has_data()
      && fifo[SCHEDULED].peek().size <= wordsleft)
         {
         picked = &fifo[SCHEDULED].get(); // then pick it to be sent
         }
      else picked = 0;           // no Lpacket available
      }
   else                // if unscheduled -- look at all FIFOs in order
      {
      // if there is anything in the FIFO and it fits in the time left
      if(fifo[SCHEDULED].has_data()
       && fifo[SCHEDULED].peek().size <= wordsleft)
         {
         picked = &fifo[SCHEDULED].get(); // then pick it to be sent
         }
      else if(fifo[HIGH].has_data()       // ditto for HIGH
       && fifo[HIGH].peek().size <= wordsleft)
         {
         picked = &fifo[HIGH].get();
         }
      else if(fifo[LOW].has_data()        // ditto for LOW
       && fifo[LOW].peek().size <= wordsleft)
         {
         picked = &fifo[LOW].get();
         }
      else picked = 0;           // no Lpacket available
      }

   return picked;
   }

//
```

```
// confirmation from the ACM that the picked Lpacket was sent (or possibly aborted)
//
void LpacketSent(TXSTATUS status)
   {
   if(picked->fixed)               // if the Lpacket was fixed
      {
      DLL_xmit_fixed_confirm (picked->id, status);  // generate a fixed confirm
      }
   else                       // else
      {
      DLL_xmit_generic_confirm (picked->id, status);// generate a generic confirm
      }

   delete picked;                  // delete temp data
   picked = 0;
   }
```

### 9.4 RxLLC

Le RxLLC (réception LLC) doit placer dans la mémoire tampon les Lpackets provenant de la Receive Machine (RxM) tels qu'ils sont assemblés. Si la RxM indique qu'une DLPDU correcte a été conclue, tous les Lpackets placés dans la mémoire tampon doivent être présentés aux couches supérieures dans l'ordre de réception. Si la RxM indique qu'une DLPDU a été conclue, tous les Lpackets placés dans la mémoire tampon doivent être ignorés.

```
// DLL RX LLC State Machine Description

// This state machine gets Lpackets from the RxM,
// and DLPDU status from the deserializer.
// It handles quarantining, and passes quarantined
// Lpackets up to the DLS-user

////////////////////////////////////////////////////////////////////////////
//
// generic type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;

//
// Lpacket class
//
class Lpacket
   {
   public:

   // the size octet of the current Lpacket
   USINT size;

   // the ctl octet of the current Lpacket
   USINT ctl;

// Lpacket constants: masks for ctl octet

#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }

   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }

   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }


   // store next octet to the Lpacket
   void put_octet(USINT data);

   USINT &operator[](int index);

   // init for a new Lpacket
   void init(void);

   Lpacket(void *p) {} // constructor
   };
```

```
class rxLpacket: public Lpacket
   {
   public:
   USINT     source;          // the MAC ID of the node that sent this Lpacket
   int    memory_size;     // size of the Lpacket in memory
   BOOL      fixed;       // true if fixed , false if generic

   rxLpacket(void *p): Lpacket(p) {}
   };
////////////////////////////////////////////////////////////////////////////
//
// a class to represent message FIFOs
//
class FIFO
   {
   public:

   void put(rxLpacket lp);       // add an Lpacket to the fifo
   rxLpacket &get(void);         // remove an Lpacket from the fifo
   void flush(void);          // delete all Lpackets in the FIFO
   BOOL has_data();       // true if the fifo is not empty
   };
////////////////////////////////////////////////////////////////////////////
//
// a class to represent generic tags
//
class gen_tag
   {
   USINT data[3];
   public:
   // constructor
   gen_tag(USINT first, USINT second, USINT third)
      {
      data[0] = first;
      data[1] = second;
      data[2] = third;
      }

   USINT &operator[](int index)
      {
      return data[index];
      }
   };
////////////////////////////////////////////////////////////////////////////
//
// internal variables
//

FIFO fifo;


////////////////////////////////////////////////////////////////////////////
//
// interface to PhL
//
BOOL ph_lock_indication;  // optimistic DLPDU detect (clock recovery is tracking)
BOOL ph_frame_indication; // pessimistic DLPDU detect (Manchester valid since the SD)

////////////////////////////////////////////////////////////////////////////
//
// interface to Deserializer
//
extern BOOL RX_endMAC;    // the end delimiter has been detected
extern BOOL RX_FCSOK;     // FCS on last DLPDU was OK
extern BOOL RX_abort;     // abort received on last DLPDU

////////////////////////////////////////////////////////////////////////////
//
// interface to RxM
//
BOOL RX_receivedLpacket;  // indication: an Lpacket has been received
rxLpacket RX_Lpacket(0);  // data: the Lpacket most recently received
```

```
////////////////////////////////////////////////////////////////////////////
//
// Interface to DLS-user
//
DLL_recv_fixed_indication (
          USINT Lpacket[],
          UINT size,
          USINT service,
          USINT sourceID);

DLL_recv_generic_indication (
          USINT Lpacket[],
          UINT size,
          gen_tag tag);
////////////////////////////////////////////////////////////////////////////
//
// interface to station management
//
BOOL SM_powerup;    // input indication: powerup has occurred
////////////////////////////////////////////////////////////////////////////
//
// RxLLC states
//

// wait for powerup

state: powerup

   event:    SM_powerup
   destination: idle
   action:
      fifo.flush();


// wait for a DLPDU to start
state: idle

   event:    ph_frame_indication == TRUE
   destination: getLpacket


state: getLpacket

   // stuff a new Lpacket into FIFO
   event:    RX_receivedLpacket     // wait for a new Lpacket to be assembled
   destination: getLpacket
   action:
      fifo.put(RX_Lpacket);        // stuff the Lpacket into the FIFO


   // if FCS is good, give all Lpackets to next layer up
   event:    RX_endMAC
   condition:   RX_FCSOK
   destination: idle
   action:
      rxLpacket &lp(0);

      while(fifo.has_data())
         {
         lp = fifo.get();
```

```
         if(lp.fixed)
            {
            DLL_recv_fixed_indication(
               &lp[4],      // rip off 4 octets of header to leave DLSdu
               lp.memory_size-4,
               lp[2],    // send service octet
               lp.source);  // send source MAC ID
            }
         else
            {
            DLL_recv_generic_indication(
               &lp[6],      // rip off 6 octets of header to leave DLSdu
               lp.memory_size-6,
               gen_tag(lp[3],lp[4],lp[5])); // send a three octet tag
            }
         }


   // if DLPDU is damaged, flush all the Lpackets in the FIFO
   event:    RX_endMAC
   condition:   !RX_FCSOK
   destination: idle
   action:
      fifo.flush();             // discard all Lpackets
```

### 9.5 Machine de transmission (TxM)

La machine de transmission (TxM) doit recevoir des commandes et des données provenant de la machine de contrôle d'accès pour transmettre les composants de la DLPDU. Elle doit transmettre une DLPDU en transmettant des octets de données et des commandes au convertisseur parallèle-série.

```
// DLL TxM State Machine Description

// This state machine gets commands and Lpackets from the ACM to
// build and transmit a DLPDU. It uses the services of the
// Serializer to do this.

////////////////////////////////////////////////////////////////////////////
//
// generic type and constant definitions
//

typedef enum {FALSE=0, TRUE=1} BOOL;

typedef void *IDENTIFIER;

////////////////////////////////////////////////////////////////////////////
//
// Lpacket definition
//
// Lpacket constants: masks for ctl octet
//
#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

//
// Lpacket class
//
class Lpacket
   {
   public:

   USINT     size;
   USINT     ctl;

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }
```

```
   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }

   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }

   // get the next octet to be transmitted
   USINT get_next_octet(void);

   // is this Lpacket aborted?
   BOOL abort(void);
   };


////////////////////////////////////////////////////////////////////////////
//
// internal variables
//
int counter;


////////////////////////////////////////////////////////////////////////////
//
// interface from ACM and TxLLC
//
BOOL TX_sendHeader;          // command: send preamble, SD, source MAC ID
USINT myaddr;             // input: the source MAC ID
BOOL TX_headerComplete;          // reply: header has been sent

BOOL TX_sendLpacket;             // command: send an Lpacket
class Lpacket lp;             // input: the Lpacket to be sent
BOOL TX_LpacketComplete;         // reply: Lpacket has been sent

BOOL TX_sendTrailer;             // command: send FCS, ED
BOOL TX_done;             // reply: DLPDU is complete

BOOL TX_abort;               // status: DLPDU was aborted


////////////////////////////////////////////////////////////////////////////
//
// interface to Serializer
//
void SER_txRequest(BOOL state);     // command: turn transmitter on and off
void SER_sendData(USINT data);         // command: send a Manchester coded data octet
void SER_sendSD(void);           // command: send a start delimiter
void SER_sendED(void);           // command: send an end delimiter
void SER_clearFCS(void);         // command: clear the FCS accumulator
void SER_sendFCS(void);          // command: send the FCS

BOOL SER_operationComplete;         // reply: current operation is completed


////////////////////////////////////////////////////////////////////////////
//
// interface to station management
//
BOOL SM_powerup;          // event indication


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//
// TxM states

//
// wait for powerup
//
```

```
state: powerup

   event:    SM_powerup
   destination: idle

//
// wait for a DLPDU to send
//
state: idle

   // when commanded by ACM to start a DLPDU, send the first preamble
   event:    TX_sendHeader
   destination: pre0
   action:
      TX_done = FALSE;
      TX_headerComplete = FALSE;
      TX_abort = FALSE;
      SER_txRequest(TRUE);      // turn on transmitter
      SER_sendData(0xff);    // send first preamble octet

//
// send second preamble octet
//
state: pre0

   event:    SER_operationComplete
   destination: pre1
   action:
      SER_sendData(0xff);    // send second preamble octet

//
// send start delimiter
//
state: pre1

   event:       SER_operationComplete
   destination: sd
   action:
      SER_sendSD();      // send source MAC ID

//
// send MAC ID
//
state: sd

   event:    SER_operationComplete
   destination: finishHeader
   action:
      SER_clearFCS();
      SER_sendData(myaddr);

//
// wait for header complete
//
state: finishHeader

   event:    SER_operationComplete
   destination: nextLP
   action:
      TX_headerComplete = TRUE;

//
// wait for a command to either send an Lpacket or to terminate the DLPDU
//
state: nextLP

   // start a new Lpacket, grab the size octet
   event:       TX_sendLpacket
   destination: sendCtl
   action:
      counter = lp.wire_size() - 1;// note size zero is not a permitted value
      SER_sendData(lp.get_next_octet());
      TX_LpacketComplete = FALSE;
```

```
   // terminate DLPDU: send the FCS
   event:       TX_sendTrailer
   destination: sendFCS
   action:
      SER_sendFCS();

state: sendCtl

   // sent the CTL octet, ignoring tag pad if present
   event:    SER_operationComplete
   destination: sendData
   action:
      counter = counter - 1;
      SER_sendData(lp.get_next_octet());
      if(lp.tag_pad())    // if tag pad
         {
         lp.get_next_octet();   // discard next octet
         }

//
// send the rest of the data in an Lpacket
//
state: sendData

   // if aborted, send an abort sequence: SD followed immediately by ED
   event:    SER_operationComplete
   condition:   lp.abort()
   destination: abort1
   action:
      SER_sendData(0xff);    // required before the Start Delimiter to avoid
                  // violating run length requirement of phy layer

   // if more data, send the next octet
   event:    SER_operationComplete
   condition:   !lp.abort() && counter > 0
   destination: sendData
   action:
      counter = counter - 1;
      SER_sendData(lp.get_next_octet());


   // when done, wait for the next command
   event:    SER_operationComplete
   condition:   !lp.abort() && counter <= 0
   destination: nextLP
   action:
      if(lp.data_pad())      // if data pad
         {
         lp.get_next_octet();   // discard next octet
         }
      TX_LpacketComplete = TRUE;// signal that Lpacket is done

//
// send the ED
//
state: sendFCS

   event:    SER_operationComplete
   destination: sendED
   action:
      SER_sendED();

//
// wait for ED complete, then shut down and wait for next DLPDU
//
state: sendED

   event:    SER_operationComplete
   destination: idle
   action:
      SER_txRequest(FALSE);     // turn off transmitter
      TX_done = TRUE;     // signal that DLPDU is done
```

```
//
// sent an FF before the abort, when it's done send an Start Delimiter
//
state: abort1

   event:    SER_operationComplete
   destination: abort2
   action:
      SER_sendSD();

//
// send second part of the abort (End Delimiter)
//
state: abort2

   event:    SER_operationComplete
   destination: abort3
   action:
      SER_sendED();

//
// wait for abort complete, then shut down and wait for next DLPDU
//
state: abort3

   event:    SER_operationComplete
   destination: idle
   action:
      SER_txRequest(FALSE);     // turn off transmitter
      TX_abort = TRUE;    // assert error flag
      TX_LpacketComplete = TRUE;// signal that Lpacket is done
      TX_done = TRUE;     // signal that DLPDU is done
```

### 9.6 Machine destinataire (RxM)

La machine destinataire (RxM) doit recevoir les informations de trame et les octets de données provenant du convertisseur série-parallèle. Elle doit décoder les Lpackets provenant du flux d'octets reçu pour la présentation à l'ACM et au RxLLC. Elle doit accepter les Lpackets génériques et fixes dont le filtre de balise correspondant est activé, et ignorer les Lpackets qui ne sont pas activés.

```
// DLL RxM State Machine Description

// This state machine gets octet symbols from the Deserializer and uses
// them to build Lpackets, which are then sent to the RxLLC.


////////////////////////////////////////////////////////////////////////
//
// generic type and constant definitions
//

typedef enum {FALSE=0, TRUE=1} BOOL;

typedef void *IDENTIFIER;

////////////////////////////////////////////////////////////////////////
//
// Lpacket definition
//
// Lpacket constants: masks for ctl octet
//
#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

//
// Lpacket class
//
class Lpacket
   {
   public:

   // the size octet of the current Lpacket
   USINT size;

   // the ctl octet of the current Lpacket
   USINT ctl;

   // return the value of the tag pad bit
   int tag_pad(void)
      {
      return (ctl&TAGPAD) >>1;
      }

   // return the value of the data pad bit
   int data_pad(void)
      {
      return (ctl&DATAPAD) >>2;
      }

   // return the number of octets in the Lpacket
   int wire_size(void)
      {
      return size*2 - tag_pad() - data_pad();
      }

   // store next octet to the Lpacket
   void put_octet(USINT data);

   // get an octet from the Lpacket
   USINT operator[](int index);

   // init for a new Lpacket
   void init(void);
```

```
   };


class rxLpacket: public Lpacket
   {
   public:
   USINT  source;      // the MAC ID of the node that sent this Lpacket
   int memory_size; // size of the Lpacket in memory
   BOOL   fixed;    // true if fixed , false if generic
   };



////////////////////////////////////////////////////////////////////////////
//
// a class to represent generic tags
//
class gen_tag
   {
   USINT data[3];

   public:

   // constructor
   gen_tag(USINT first, USINT second, USINT third)
      {
      data[0] = first;
      data[1] = second;
      data[2] = third;
      }

   USINT operator[](int index)
      {
      return data[index];
      }
   };


////////////////////////////////////////////////////////////////////////////
//
// the interface to the tag filter lookup tables -- the implementation is unspecified
//
class gen_screener
   {
   public:
   BOOL enable(gen_tag tag); // set tag, requires tag, TRUE=success/FALSE=failure
   BOOL disable(gen_tag tag);// clear tag, requires tag, TRUE=success/FALSE=failure
   BOOL lookup(gen_tag tag); // lookup tag, requires tag, TRUE=enabled/FALSE=disabled
   void init(void);    // powerup init & clear
   };

class fixed_screener
   {
   public:
   BOOL enable(USINT service);  // set tag, requires tag, TRUE=success/FALSE=failure
   BOOL disable(USINT service); // clear tag, requires tag, TRUE=success/FALSE=failure
   BOOL lookup(USINT service);  // lookup tag, requires tag,
TRUE=enabled/FALSE=disabled
   void init(void);    // powerup init & clear
   };


////////////////////////////////////////////////////////////////////////////
//
// internal variables
//
// the tag filtering tables
class gen_screener gen;
class fixed_screener fixed;

// MAC ID of source of DLPDU
USINT source;
```

```
// an octet counter
int counter;

// temps for generic screener tag octets
USINT gen0;
USINT gen1;


////////////////////////////////////////////////////////////////////////////
//
// interface to PhL
//
extern BOOL ph_lock_indication;  // optimistic DLPDU detect (clock recovery is
tracking)
extern BOOL ph_frame_indication; // pessimistic DLPDU detect (Manchester valid since
the SD)


////////////////////////////////////////////////////////////////////////////
//
// interface to Deserializer
//
extern BOOL RX_dataReady;     // an octet of a DLPDU is available
extern USINT RX_rxData;          // octet most recently received from DLPDU (source
MAC ID)
extern BOOL RX_endMAC;        // the end delimiter has been detected
extern BOOL RX_FCSOK;         // FCS on last DLPDU was OK
extern BOOL RX_abort;         // abort received on last DLPDU


////////////////////////////////////////////////////////////////////////////
//
// interface to station management

class DLL_config_data
   {
   public:
   USINT  myaddr;
   UINT   NUT_length;
   USINT  smax;
   USINT  umax;
   USINT  slotTime;
   USINT  blanking;
   USINT  gb_start;
   USINT  gb_center;
   USINT  interval_count;
   USINT  modulus;
   USINT  gb_prestart;
   };

extern DLL_config_data SM_current;     // the DLL config variables -- need my MAC ID
extern BOOL SM_powerup;          // input indication: powerup has occurred


////////////////////////////////////////////////////////////////////////////
//
// interface to RxM
//


// received data interface

BOOL RX_receivedLpacket;  // an Lpacket has been received
rxLpacket RX_Lpacket;     // the Lpacket most recently received


// interface to manage tag filters

extern enable_generic_confirm (IDENTIFIER id, BOOL result);
extern disable_generic_confirm (IDENTIFIER id, BOOL result);
extern enable_fixed_confirm (IDENTIFIER id, BOOL result);
extern disable_fixed_confirm (IDENTIFIER id, BOOL result);
```

```
void DLL_enable_generic_request (IDENTIFIER id, gen_tag tag)
   {
   enable_generic_confirm (id, gen.enable(tag));
   }

void DLL_disable_generic_request (IDENTIFIER id, gen_tag tag)
   {
   disable_generic_confirm (id, gen.disable(tag));
   }

void DLL_enable_fixed_request (IDENTIFIER id, USINT service)
   {
   enable_fixed_confirm (id, fixed.enable(service));
   }

void DLL_disable_fixed_request (IDENTIFIER id, USINT service)
   {
   disable_fixed_confirm (id, fixed.disable(service));
   }



////////////////////////////////////////////////////////////////////////////
//
// RxM states
//

//
// wait for powerup
//
state: powerup

   event:    SM_powerup
   destination: idle
   action:
      RX_receivedLpacket = FALSE;

//
// wait for a DLPDU to start
//
state: idle

   event:    ph_frame_indication == TRUE
   destination: getID


state: getID

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // get the source MAC ID and save it for all the Lpackets in the DLPDU
   event:    RX_dataReady == TRUE
   destination: getSize
   action:
      source = RX_rxData;    // save the source MAC ID for subsequent Lpackets

state: getSize

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // get the size octet and init a new Lpacket
   event:    RX_dataReady == TRUE
   destination: getCtl
   action:
      RX_receivedLpacket = FALSE;      // re-initialize the interface
      RX_Lpacket.init();
      RX_Lpacket.source = source;      // save the source in the new Lpacket
      RX_Lpacket.put_octet(RX_rxData); // store the size octet
```

```
state: getCtl

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // get the CTL octet for a fixed Lpacket
   event:    RX_dataReady == TRUE && (RX_rxData & 0xFB) == 0x01  // fixed tag
   destination: getFixed0
   action:
      RX_Lpacket.put_octet(RX_rxData); // store the ctl octet
      counter = RX_Lpacket.wire_size() - 2;  // init the octet counter

   // get the CTL octet for a generic Lpacket
   event:    RX_dataReady == TRUE && (RX_rxData & 0xFB) == 0x12  // generic tag
   destination: getGen0
   action:
      RX_Lpacket.put_octet(RX_rxData); // store the ctl octet
      RX_Lpacket.put_octet(0);     // store the pad octet
      counter = RX_Lpacket.wire_size() - 2;  // init the octet counter


   // get the CTL octet for a generic Lpacket
   event:    RX_dataReady == TRUE
         && (RX_rxData & 0xFB)!= 0x01
         && (RX_rxData & 0xFB)!= 0x12
   destination: ignoreRest
   action:
      counter = RX_Lpacket.wire_size() - 2;  // init the octet counter


//
// handle service octet of fixed Lpacket
//
state: getFixed0

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // handle case if tag is enabled
   event:    RX_dataReady == TRUE
         && fixed.lookup(RX_rxData)
   destination: getFixed1
   action:
      RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
      counter--;             // count it

   // handle case if tag is disabled
   event:    RX_dataReady == TRUE
         &&!fixed.lookup(RX_rxData)
   destination: ignoreRest
   action:
      counter--;             // count it

//
// handle dest octet of fixed Lpacket
//
state: getFixed1

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // handle case if addressed to me or broadcast
   event:    RX_dataReady == TRUE
         && (RX_rxData == SM_current.myaddr || RX_rxData == 0xFF)
   destination: getRest
   action:
      RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
      counter--;             // count it
      RX_Lpacket.fixed = TRUE;
```

```
   // handle case if not addressed to me
   event:    RX_dataReady == TRUE
          && RX_rxData!= SM_current.myaddr
          && RX_rxData!= 0xFF
   destination: ignoreRest
   action:
      counter--;             // count it

//
// handle first tag octet of generic Lpacket
//
state: getGen0

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // save the next octet
   event:    RX_dataReady == TRUE
   destination: getGen1
   action:
      RX_Lpacket.put_octet(gen0=RX_rxData);  // save and put the next octet into
Lpacket
      counter--;             // count it

//
// handle second tag octet of generic Lpacket
//
state: getGen1

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // save the next octet
   event:    RX_dataReady == TRUE
   destination: getGen2
   action:
      RX_Lpacket.put_octet(gen1=RX_rxData);  // save and put next data octet into
Lpacket
      counter--;             // count it

//
// handle third octet of generic Lpacket
//
state: getGen2

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // handle case if local node is interested in this generic Lpacket
   event:    RX_dataReady == TRUE
          && gen.lookup(gen_tag(gen0, gen1, RX_rxData))
   destination: getRest
   action:
      RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
      counter--;             // count it
      RX_Lpacket.fixed = FALSE;


   // handle case if not interested in this generic Lpacket
   event:    RX_dataReady == TRUE
          &&!gen.lookup(gen_tag(gen0,gen1,RX_rxData))
   destination: ignoreRest
   action:
      counter--;             // count it


state: getRest

   // if DLPDU is terminated, wait for next DLPDU
```

```
   event:    ph_frame_indication == FALSE
   destination: idle

   // get data octets except the last
   event:    RX_dataReady == TRUE
   condition:   counter > 1
   destination: getRest
   action:
      RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
      counter--;             // count it

   // get the last data octet; handle data pad, indicate arrived Lpacket; wait for
next Lpacket
   event:    RX_dataReady == TRUE
   condition:   counter == 1
   destination: getSize
   action:
      RX_Lpacket.put_octet(RX_rxData); // get the last octet
      if(RX_Lpacket.data_pad())    // if a data pad is requested,
         {
         RX_Lpacket.put_octet(0);  // insert a zero
         }

      // remove pad octets from data count
      RX_Lpacket.memory_size -= RX_Lpacket.size*2;

      RX_receivedLpacket = TRUE;   // notify that that an Lpacket has arrived


state: ignoreRest

   // if DLPDU is terminated, wait for next DLPDU
   event:    ph_frame_indication == FALSE
   destination: idle

   // get data octets except the last
   event:    RX_dataReady == TRUE
   condition:   counter > 1
   destination: ignoreRest
   action:
      counter--;             // count it

   // handle the last data octet; wait for next Lpacket
   event:    RX_dataReady == TRUE
   condition:   counter == 1
   destination: getSize
```

### 9.7 Convertisseur parallèle-série

Pour chaque commande provenant de TxM, le convertisseur parallèle-série doit générer le flux M_symbol approprié, qui doit être la sortie vers le support via PhL.

```
// DLL octet Serializer

// This state machine gets commands to transmit octet symbols from the TxM.
// It decomposes the octet symbols into M_symbols, and
// uses the services of the PHY layer to send these on the medium.
// This specifies the correct order of transmission of the various octet symbols.

// The FCS algorithm is the same as that used by HDLC

////////////////////////////////////////////////////////////////////////////
//
// data types
//
typedef enum {FALSE, TRUE} BOOL;


////////////////////////////////////////////////////////////////////////////
//
// interface to TxM
//

// requests (TxM to SER)
```

```
void SER_txRequest(BOOL state);     // command: turn transmitter on and off
void SER_sendData(USINT data);         // command: send a Manchester coded data octet
void SER_sendSD(void);           // command: send a start delimiter
void SER_sendED(void);           // command: send an end delimiter
void SER_clearFCS(void);         // command: clear the FCS accumulator
void SER_sendFCS(void);          // command: send the FCS

// indications (SER to TxM)

extern void SER_operationComplete(void);  // notify the TxM that an operation is
complete


////////////////////////////////////////////////////////////////////////////
//
// interface to the PHY layer
//
typedef enum {M_0, M_1, M_ND_plus, M_ND_minus} M_SYMBOL;

BOOL ph_frame_request;        // transmit enable/disable
M_SYMBOL ph_data_request;        // data to be sent




////////////////////////////////////////////////////////////////////////////
//
// internal variables
//
static unsigned int accum;       // the FCS accumulator, 16 bits, sign is irrelevant
                    // as long as zeros shift in from the right

////////////////////////////////////////////////////////////////////////////
//
// internal subroutines
//

// add one octet to the FCS accumulator
// this is just one possible implementation of the HDLC FCS

static void FCS_octet(unsigned int d)
   {
   accum ^= d & 0xff;

   for(int i=0;i<8;i++)
      {
      accum = (accum>>1) ^ (0x8408 * (accum&1));// use FCS-CCITT polynomial
      }
   }
////////////////////////////////////////////////////////////////////////////
//
// Serializer implementation
//

// requests (TxM to SER)

// enable transmitter
void SER_txRequest(BOOL state)
   {
   ph_frame_request = state;
   }

// add a data octet to the FCS and send it
void SER_sendData(USINT data)
   {
   FCS_octet(data);

   // transmit data bits 0 through 7

ph_data_request = (data & 0x01)? M_1: M_0;
ph_data_request = (data & 0x02)? M_1: M_0;
ph_data_request = (data & 0x04)? M_1: M_0;
```

```
ph_data_request = (data & 0x08)? M_1: M_0;
ph_data_request = (data & 0x10)? M_1: M_0;
ph_data_request = (data & 0x20)? M_1: M_0;
ph_data_request = (data & 0x40)? M_1: M_0;
ph_data_request = (data & 0x80)? M_1: M_0;

   SER_operationComplete();     // tell TxM that operation is complete
   }


// send a start delimiter
void SER_sendSD(void)
   {
   ph_data_request = M_ND_plus;
   ph_data_request = M_0;
   ph_data_request = M_ND_minus;
   ph_data_request = M_ND_plus;
   ph_data_request = M_ND_minus;
   ph_data_request = M_1;
   ph_data_request = M_0;
   ph_data_request = M_1;
   SER_operationComplete();     // tell TxM that operation is complete
   }

// send an end delimiter
void SER_sendED(void)
   {
   ph_data_request = M_1;
   ph_data_request = M_0;
   ph_data_request = M_0;
   ph_data_request = M_1;
   ph_data_request = M_ND_plus;
   ph_data_request = M_ND_minus;
   ph_data_request = M_ND_plus;
   ph_data_request = M_ND_minus;
   SER_operationComplete();     // tell TxM that operation is complete
   }

// initialize the FCS accumulator
void SER_clearFCS(void)
   {
   accum = 0xffff;        // init the accumulator to all ones
   SER_operationComplete();     // tell TxM that operation is complete
   }

// send the FCS
void SER_sendFCS(void)
   {
   int tmp;

   tmp = accum ^ 0xffff;        // invert the FCS before sending;
   SER_sendData(tmp);        // do not calculate FCS while sending itself
   SER_sendData(tmp>>8);
   SER_operationComplete();     // tell TxM that operation is complete
   }
```

### 9.8 Convertisseur série-parallèle

#### 9.8.1 Construction d'octet

Le convertisseur série-parallèle doit accepter les M_symbols provenant de `ph_data_indication`. Les M_symbols placés au début d'une trame doivent être ignorés tant que la valeur de la ligne `ph_frame_indication` n'est pas passée de FALSE à TRUE. Cette transition doit commencer au premier octet de données (le MAC ID source). Le convertisseur série-parallèle doit alors regrouper les ensembles suivants de huit symboles de données MAC en octets. Etant donné que chaque octet de données suivant est prêt, le convertisseur série-parallèle doit passer la valeur de la ligne RX_dataReady de FALSE à TRUE.

#### 9.8.2 Contrôle FCS

Un contrôle de redondance cyclique CCITT modifié doit être réalisé sur les données transférées afin de vérifier la séquence de contrôle de trame (FCS).

Les concepts formels du générateur et contrôleur FCS sont présentés en 5.3.8.2.

La méthode de contrôle doit être identique à celle spécifiée pour HDLC (ISO/CEI 3309), sauf que les délimiteurs de début et de fin doivent être remplacés par des balises, et que le codage Manchester doit être remplacé par une insertion binaire. Le contrôleur FCS doit mettre en œuvre le polynôme $X^{16} + X^{12} + X^5 + 1$ et doit donner une séquence de contrôle de trame (FCS) de deux octets.

NOTE Les deux derniers octets de la DLPDU (octets FCS) sont calculés par le nœud émetteur de sorte que, en l'absence d'erreur, le reste du nœud destinataire soit toujours 0xF0B8 (voir 5.3.8.2).

Le traitement du FCS destinataire doit commencer par un réglage préalable du contrôleur FCS à 0xFFFF lorsque la valeur de la ligne `ph_frame_indication` passe de FALSE à TRUE. Tous les bits de données suivants de la ligne `ph_data_indication`, à l'exception du délimiteur de fin, doivent être appliqués au contrôleur FCS. A l'issue de la réception d'une DLPDU, le reste (FCS) doit être comparé à 0xF0B8.

#### 9.8.3 Fin du traitement DLPDU

Le traitement d'octet de données doit se poursuivre tant que la valeur de `ph_frame_indication` n'est pas passée de TRUE à FALSE. Cette transition peut se produire à la moitié de l'octet, mais aucune action ne doit être entreprise avant le début de l'octet suivant. Cette durée doit marquer la fin de la réception de cette trame.

Le `ph_status_indication` reçu après que la valeur de `ph_frame_indication` est passée de TRUE à FALSE doit déterminer laquelle des actions suivantes doit être entreprise:

– si Normal – attribuer la valeur TRUE à RX_FCSOK si (reste FCS = 0xF0B8); sinon, attribuer la valeur FALSE;

– si Abort – attribuer la valeur TRUE à Rx_abort; attribuer la valeur FALSE à Rx_FCSOK;

– si Invalid – attribuer la valeur FALSE à Rx_abort; attribuer la valeur FALSE à Rx_FCSOK.

### 9.9 Gestion de DLL

La gestion de DLL doit placer en mémoire tampon les variables de gestion de station nécessaires au fonctionnement de la DLL. Elle doit gérer les modifications synchronisées de ces variables par l'intermédiaire des interfaces vers Station Management et l'ACM.

```
// DLL Station Management Interface Description

// This component holds the station management variables that are
```

```
// subject to synchronized change.

////////////////////////////////////////////////////////////////////////////
//
// type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;

typedef void *IDENTIFIER;    // The data type of the identifier is unspecified.


// config data definition

class DLL_config_data
   {
   public:
   USINT  myaddr;
   UINT   NUT_length;
   USINT  smax;
   USINT  umax;
   USINT  slotTime;
   USINT  blanking;
   USINT  gb_start;
   USINT  gb_center;
   USINT  interval_count;
   USINT  modulus;
   USINT  gb_prestart;
   };

////////////////////////////////////////////////////////////////////////////
//
// interface to station management
//
void DLL-tMinus-start-countdown-request (IDENTIFIER id, USINT start_count);
extern void DLL-tMinus-start-countdown-confirm (IDENTIFIER id, BOOL result);

void DLL_set_pending_request (IDENTIFIER id, DLL_config_data params);
extern void DLL_set_pending_confirm (IDENTIFIER id, BOOL result);

void DLL_get_pending_request (IDENTIFIER id);
extern void DLL_get_pending_confirm (IDENTIFIER id , DLL_config_data params);

void DLL_set_current_request (IDENTIFIER id, DLL_config_data params);
extern void DLL_set_current_confirm (IDENTIFIER id, BOOL result);

void DLL_get_current_request (IDENTIFIER id);
extern void DLL_get_current_confirm (IDENTIFIER id , DLL_config_data params);

////////////////////////////////////////////////////////////////////////////
//
// interface to ACM
//
extern void DLL_tminus_zero_indication ();// ACM calls this when tMinus==0 occurs
DLL_config_data DLL_SM_pending;     // the ACM needs to see both current and pending
DLL_config_data DLL_SM_current;
extern ACM_tMinus;            // poke the ACM's tMinus counter to get
                    // a synchronized change started. The ACM
                    // continually writes over this every time the
                    // moderator is received, unless it is the moderator

void SM_supernode(void);         // if this node saw a moderator sent by a supernode,
                    // any pending SM update shall be cancelled


////////////////////////////////////////////////////////////////////////////
//
// internal variables
//
BOOL pending_changed = FALSE;


////////////////////////////////////////////////////////////////////////////
//
// write the pending copy of DLL_config
```

```
void DLL_set_pending_request (IDENTIFIER id, DLL_config_data params)
   {
   DLL_SM_pending = params;     // save the params in the pending buffer
   pending_changed = TRUE;
   DLL_set_pending_confirm (id, TRUE);
   }


// write the current copy of DLL_config
// ONLY USED WHEN OFFLINE
void DLL_set_current_request (IDENTIFIER id, DLL_config_data params)
   {
   DLL_SM_current = params;     // save the params in the pending buffer
   DLL_set_current_confirm (id, TRUE);
   }

// read the pending copy of DLL_config
void DLL_get_pending_request (IDENTIFIER id)
   {
   if(pending_changed == TRUE)     // if pending has been updated
      {
      // return the contents of the pending buffer
      DLL_get_pending_confirm (id , DLL_SM_pending);
      }
   else               // if pending buffer is invalid
      {
      // return the contents of the current buffer
      DLL_get_current_confirm (id , DLL_SM_current);
      }
   }

//
// read the current copy of DLL_config
//
void DLL_get_current_request (IDENTIFIER id)
   {
// return the contents of the current buffer
   DLL_get_current_confirm (id , DLL_SM_current);
   }

//
// called by the ACM when a tMinus countdown has completed
//
void DLL_tminus_zero_indication(void)
   {
   if(pending_changed)
      {
      DLL_SM_current = DLL_SM_pending;
      pending_changed = FALSE;
      }
   }

//
// start a synchronized change sequence
//
void DLL-tMinus-start-countdown-request (IDENTIFIER id, USINT start_count)
   {
   ACM_tMinus = start_count;    // the ACM polls this variable
   DLL-tMinus-start-countdown-confirm (id, TRUE);
   }

//
// cancel pending change if a supernode has been seen
//
void SM_supernode(void)
   {
   pending_changed = FALSE;
   }
```

# 10 Protocole DLR (Device Level Ring)

## 10.1 Généralités

L'Article 10 définit le protocole DLR (Device Level Ring), protocole de couche 2 offrant une redondance de support dans une topologie en anneau. Le protocole DLR est principalement destiné à être mis en œuvre dans les stations d'extrémité CP 2/2 dotées de plusieurs ports Ethernet et d'une technologie de commutateur intégré. Le protocole DLR assure la détection rapide des défauts du réseau et la reconfiguration afin de prendre en charge les applications de commande les plus exigeantes.

Etant donné que le protocole DLR fonctionne au niveau de la couche 2 (dans le modèle de réseau OSI), la présence de la topologie en anneau et le fonctionnement du protocole DLR sont transparents pour les protocoles de couche supérieure (TCP/IP et CP 2/2, par exemple), à l'exception d'un objet DLR offrant une interface de configuration et de diagnostic à CP 2/2.

Un réseau DLR contient au moins un nœud configuré en tant que superviseur d'anneau et un certain nombre de nœuds d'anneau normaux. Tous les nœuds d'anneau sont supposés comporter au moins deux ports Ethernet, ainsi qu'une technologie de commutateur intégré.

Les appareils à plusieurs ports non DLR (commutateurs ou appareils d'extrémité) peuvent être placés dans l'anneau et faire l'objet de certaines contraintes de mise en œuvre (pas de filtrage de table MAC). Les appareils non DLR auront également un impact sur le temps de remise de l'anneau dans le cas le moins favorable.

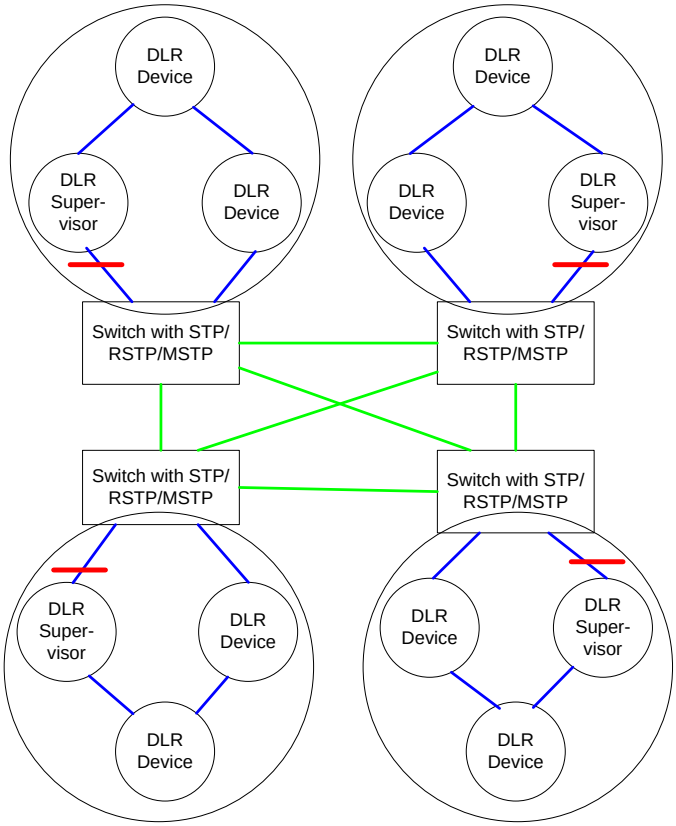
Un certain nombre d'aspects caractérise la définition du protocole DLR:

- un ensemble de comportement de nœud d'extrémité, pour les superviseurs d'anneau et les nœuds d'anneau normaux;
- les messages de protocoles et diagrammes d'état connexes;
- les exigences de mise en œuvre des appareils.

## 10.2 Topologies prises en charge

Le protocole DLR prend en charge une topologie simple à un seul anneau. Il n'existe aucun concept d'anneaux multiples ou se chevauchant. Toutefois, une installation de réseau utilise plusieurs anneaux DLR, du moment que chacun des anneaux est isolé de sorte que les messages de protocole DLR provenant d'un anneau soient absents d'un autre anneau.

Le protocole DLR peut cohabiter avec des protocoles de réseau standard (les protocoles IEEE Spanning Tree Protocols (STP, RSTP, MSTP), par exemple) et avec des protocoles de redondance spécifiques au fournisseur. En d'autres termes, les utilisateurs peuvent concevoir des topologies de réseau avec les anneaux de protocole DLR connectés à des commutateurs exécutant Spanning Tree ou d'autres protocoles d'anneau (voir Figure 27).




**Légende**

| Anglais | Français |
|---|---|
| DLR device | Appareil DLR |
| DLR supervisor | Superviseur DLR |
| Switch with ... | Commutateur avec ... |

**Figure 27 – Anneaux DLR connectés aux commutateurs**

Dans la Figure 27, chaque anneau DLR est un réseau DLR distinct, contenant chacun un superviseur d'anneau. Les superviseurs sont présentés avec un port en mode bloqué, ce qui est le cas en l'absence d'anomalie dans l'anneau.

Les commutateurs auxquels sont connectés les anneaux DLR peuvent exécuter STP/RSTP/MSTP pour éviter un fonctionnement sans boucle en présence de chemins redondants (indiquées par les lignes vertes dans la Figure 27). Les messages STP (BPDU) envoyés par le commutateur sur les ports d'anneau DLR seront bloqués par le superviseur d'anneau DLR, de sorte que les commutateurs ne bloquent pas les ports DLR (voir les considérations de l'IEEE 802.1D/IEEE 802.1Q STP/RSTP/MSTP en 10.6.4).

Les ports des commutateurs auxquels les appareils DLR sont connectés doivent être correctement configurés afin d'assurer le bon fonctionnement du réseau (voir 7.9).

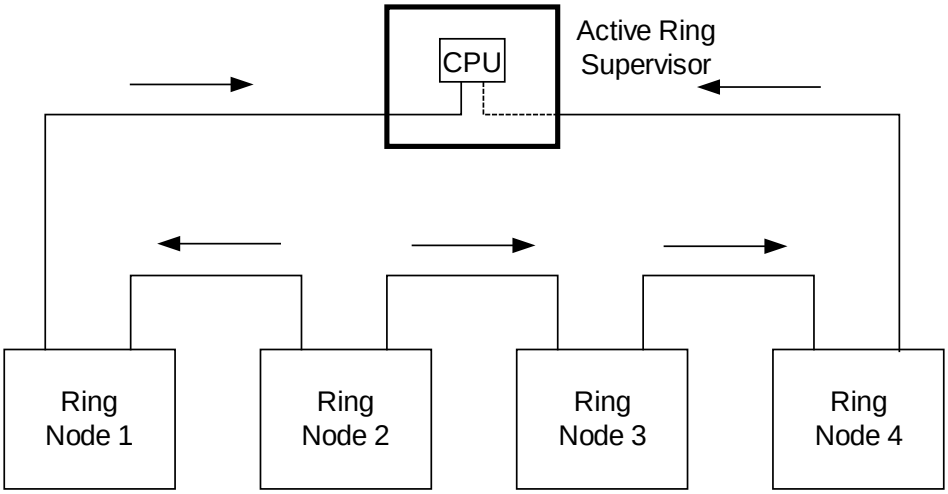
Des topologies plus complexes combinant des anneaux DLR et des commutateurs non DLR exécutant STP/RSTP/MSTP risquent de bloquer les ports DLR de manière intempestive. Voir 7.9 pour plus d'informations.

L'appareil DLR prend en charge les passerelles redondantes pour se connecter avec l'infrastructure réseau résidant hors du réseau DLR au réseau DLR lui-même. Voir 10.8 pour plus d'informations.

### 10.3 Présentation de l'opération DLR

#### 10.3.1 Fonctionnement normal

La Figure 28 illustre le fonctionnement normal d'un réseau DLR.




**Légende**

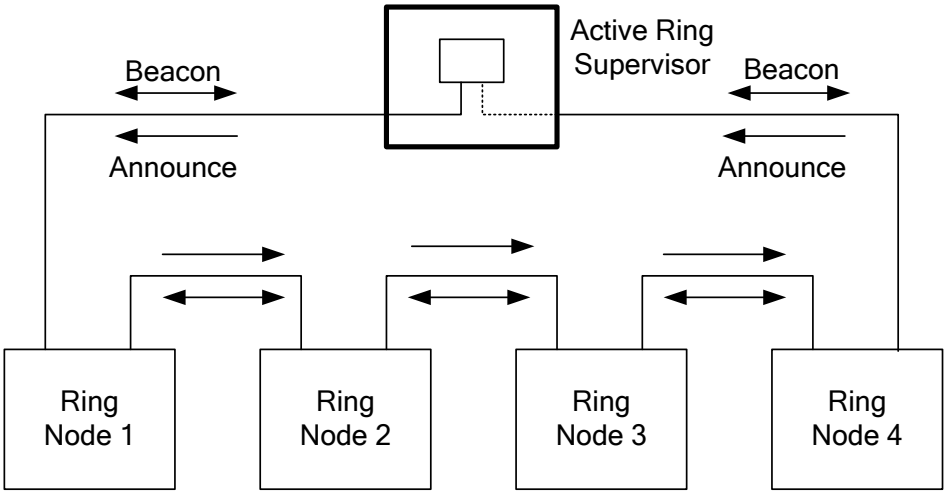
| Anglais | Français |
|---|---|
| Active ring supervisor | Superviseur d'anneau actif |
| Ring node | Nœud d'anneau |

**Figure 28 – Fonctionnement normal d'un réseau DLR**

Chaque nœud de la Figure 28 comporte deux ports Ethernet et est supposé avoir mis en œuvre un commutateur intégré. Lorsqu'un nœud d'anneau reçoit un paquet sur l'un de ses ports Ethernet, il détermine si le paquet doit être reçu par le nœud d'anneau lui-même (le paquet contient l'adresse MAC du nœud, par exemple) ou si le paquet doit être envoyé au port Ethernet de l'autre nœud.

Le superviseur d'anneau actif bloque le trafic sur l'un de ses ports (à l'exception de quelques trames particulières) et ne transfère pas le trafic d'un port à l'autre. Cette configuration permet d'éviter les boucles de réseau, et il n'existe qu'un seul chemin entre deux nœuds d'anneau pendant le fonctionnement normal.

La Figure 29 illustre l'utilisation des trames Beacon et Announce envoyées par le superviseur d'anneau actif.




**Légende**

| Anglais | Français |
|---|---|
| Active ring supervisor | Superviseur d'anneau actif |
| Ring node | Nœud d'anneau |

**Figure 29 – Trames Beacon et Announce**

Le superviseur d'anneau actif transmet une trame Beacon par l'intermédiaire de deux de ses ports Ethernet, une fois par intervalle Beacon (400 µs par défaut). Le superviseur envoie également des trames Announce, une fois par seconde. Les trames Beacon et Announce présentent plusieurs avantages:

a) leur présence informe les nœuds d'anneau de passer d'une topologie linéaire à une topologie en anneau;

b) la perte de trames Beacon au niveau du superviseur permet de détecter certains types de défauts d'anneau (les nœuds d'anneau normaux sont également en mesure de détecter et de signaler les défauts d'anneau);

c) les trames Beacon contiennent une valeur de priorité, permettant de sélectionner un superviseur actif lorsque plusieurs superviseurs d'anneau sont configurés.

### 10.3.2 Interruptions de liaison

#### 10.3.2.1 Interruptions habituelles

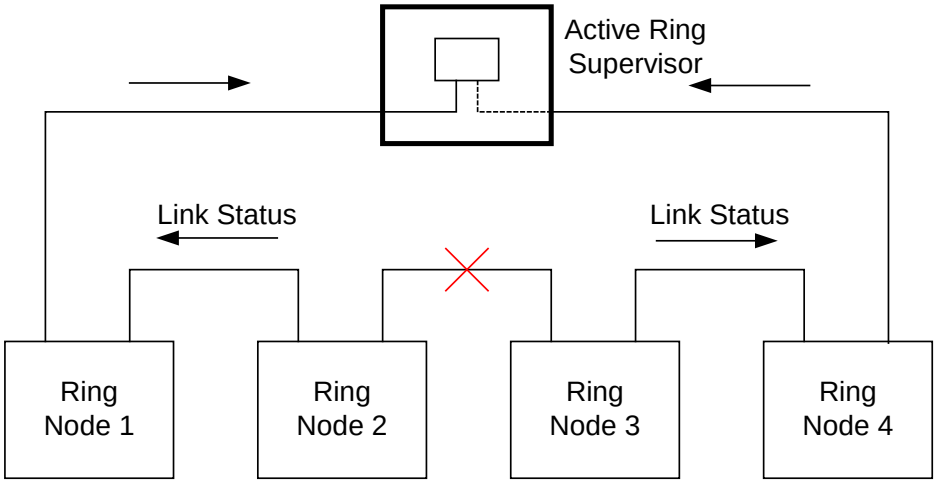
La forme la plus commune d'interruption de liaison inclut les cas suivants:

a) interruption d'une liaison ou d'une autre couche physique reconnue par un nœud adjacent à l'interruption;

b) interruption d'alimentation ou arrêt suivi d'une remise sous tension d'un nœud d'anneau, reconnu par le nœud adjacent comme étant une interruption de liaison;

c) déconnexion volontaire du support par l'utilisateur pour mettre de nouveau en ligne ou retirer des nœuds existants.

Dans les cas ci-dessus, les nœuds adjacents à la liaison interrompue envoient un message Link_Status au superviseur d'anneau. La Figure 30 illustre les nœuds d'anneau adjacents à une liaison interrompue envoyant un message Link_Status au superviseur d'anneau




**Légende**

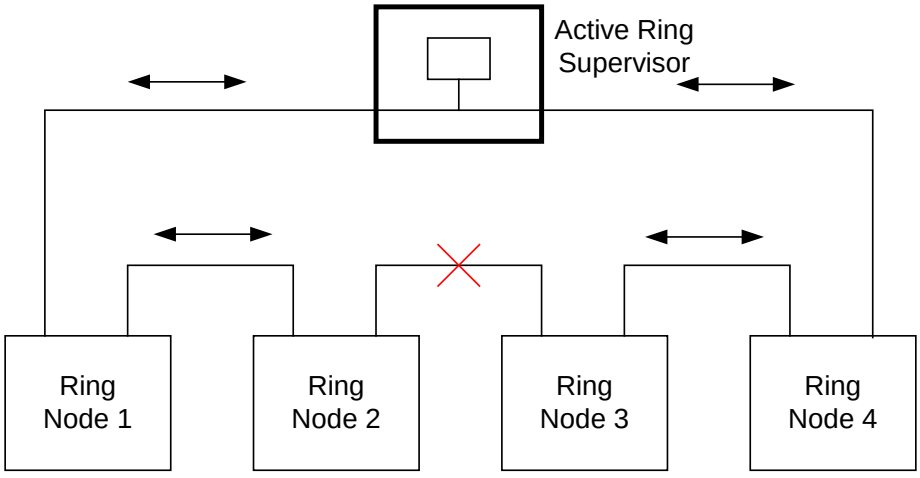
| Anglais | Français |
|---|---|
| Active ring supervisor | Superviseur d'anneau actif |
| Ring node | Nœud d'anneau |

**Figure 30 – Interruption de liaison**

Après la réception du message Link_Status, le superviseur d'anneau reconfigure le réseau en débloquant le trafic sur le port préalablement bloqué et en purgeant sa table MAC à diffusion individuelle. Le superviseur envoie immédiatement les trames Beacon et Announce avec la valeur d'état de l'anneau indiquant que l'anneau est à présent défaillant.

Les nœuds d'anneau purgent également leurs tables MAC à diffusion individuelle lors de la détection de la perte de Beacon dans une direction ou dès la réception de trames Beacon ou Announce dont la valeur d'état de l'anneau indique l'état d'anomalie de l'anneau. La purge des tables MAC à diffusion individuelle au niveau du superviseur et des nœuds d'anneau est nécessaire pour permettre au trafic du réseau d'atteindre sa destination prévue après la reconfiguration du réseau.

La Figure 31 illustre la configuration du réseau après une interruption de liaison, le superviseur d'anneau faisant passer le trafic par deux de ses ports.




**Légende**

| Anglais | Français |
|---|---|
| Active ring supervisor | Superviseur d'anneau actif |

| Anglais | Français |
|---|---|
| Ring node | Nœud d'anneau |

**Figure 31 – Reconfiguration du réseau après une interruption de liaison**

#### 10.3.2.2 Interruptions inhabituelles

Outre les interruptions de liaison les plus communes, il existe un certain nombre d'interruptions inhabituelles:

a) le(s) composant(s) matériel(s)/micrologiciel(s) de niveau supérieur d'un nœud d'anneau est/sont tombé(s) en panne, donnant lieu à une perte de trafic, mais la couche physique fonctionne normalement avec une alimentation électrique intacte;

b) des nœuds incompatibles avec le protocole d'anneau sont connectés à des nœuds compatibles avec le protocole, et l'interruption s'est produite au milieu de cette chaîne.

Dans ce cas, le superviseur d'anneau détectera la perte des trames Beacon d'abord sur un port, puis éventuellement sur deux de ses ports. Le superviseur reconfigurera le réseau (voir 10.3.2.1). De plus, le superviseur d'anneau envoie une trame Locate_Fault pour diagnostiquer l'emplacement du défaut (voir 10.5.3.7 du processus Neighbor Check).

#### 10.3.2.3 Défaut de réseau partiel

Un défaut de réseau partiel peut se produire, générant une perte de trafic dans une seule direction. Le superviseur d'anneau actif détecte un défaut partiel en surveillant la perte de trames Beacon sur un port. En cas de détection d'un défaut partiel, le superviseur actif bloque le trafic sur un port et définit une valeur d'état dans l'objet DLR. A ce stade, l'anneau sera segmenté en raison du défaut partiel, impliquant l'intervention de l'utilisateur.

#### 10.3.2.4 Cycles rapides de défaut/restauration

Certaines conditions (un connecteur réseau défaillant, par exemple) peuvent amener le superviseur d'anneau à détecter une série de cycles rapides de défaut/restauration. Si cela perdure, ce type de condition peut engendrer une instabilité du réseau difficile à diagnostiquer. Lorsqu'un superviseur actif détecte la condition de défaut/restauration rapide (5 défauts toutes les 30 s), il définit une valeur d'état dans l'objet DLR et bloque le trafic sur un port. L'utilisateur doit explicitement supprimer la condition par l'intermédiaire de l'objet DLR.

### 10.4 Classes de mise en œuvre DLR

Il existe plusieurs classes de mise en œuvre DLR, présentées ci-dessous. Les exigences détaillées pour chaque classe de mise en œuvre sont approfondies en 10.5.

- Superviseur d'anneau
  Cette classe d'appareil peut être un superviseur d'anneau. Ces appareils doivent mettre en œuvre les comportements requis du superviseur d'anneau, notamment la possibilité d'envoyer et de traiter des trames Beacon à intervalle Beacon par défaut de 400 µs. Des intervalles Beacon plus petits (100 µs, par exemple) peuvent être pris en charge, mais ne sont pas obligatoires.
- Nœud d'anneau, Beacon
  Cette classe d'appareil met en œuvre le protocole DLR, mais sans les fonctions de superviseur d'anneau. L'appareil doit être en mesure de traiter et de manipuler les trames Beacon envoyées par le superviseur d'anneau. Les nœuds d'anneau Beacon doivent prendre en charge des vitesses Beacon comprises entre 100 µs et 100 ms afin d'assurer toutes les mises en œuvre du superviseur d'anneau.
- Nœud d'anneau, Announce
  Cette classe d'appareil met en œuvre le protocole DLR, mais sans les fonctions de superviseur d'anneau. Afin d'adapter les nœuds n'ayant pas la capacité de traiter les

trames Beacon, les nœuds d'anneau peuvent simplement transférer les trames Beacon, mais peuvent ne pas les traiter explicitement. Ces nœuds doivent traiter les trames Announce.

## 10.5 Comportement DLR

### 10.5.1 Variables DLR

Le Tableau 134 récapitule les variables utilisées dans le comportement et les messages de protocole DLR. Voir 10.5.3 et 10.5.2 sur le comportement du nœud d'anneau et du superviseur d'anneau et sur les messages DLR pour plus de détails. L'objet DLR (voir 7.9) expose ces variables (à l'exception de l'état du nœud) par l'intermédiaire des attributs d'objet.

**Tableau 134 – Variables DLR**

| Variable DLR | Description |
|---|---|
| Node State | Etat interne du diagramme d'états DLR d'un nœud:<br>IDLE_STATE – état initial des non superviseurs, indiquant un mode de topologie linéaire<br>FAULT_STATE – état initial du superviseur d'anneau activé ou si un défaut d'anneau a été détecté (le superviseur et les nœuds d'anneau)<br>NORMAL_STATE – fonction normale en mode de topologie en anneau |
| Ring State | Etat du réseau en anneau, transmis par les superviseurs d'anneau dans les trames Beacon et Announce:<br>RING_NORMAL_STATE – L'anneau fonctionne, le superviseur bloquant le trafic sur un port<br>RING_FAULT_STATE – Défaut détecté, le superviseur d'anneau ne bloque pas le trafic (il s'agit également de l'état initial transmis dans les trames Beacon et Announce) |
| Beacon Interval | Intervalle auquel le superviseur d'anneau envoie des trames Beacon. Les superviseurs doivent prendre en charge une plage comprise entre 400 µs et 100 ms. La valeur par défaut doit être 400 µs. Les superviseurs peuvent prendre en charge un intervalle Beacon inférieur à 400 µs, mais ce n'est pas requis. L'intervalle Beacon minimal absolu est de 100 µs. Les nœuds d'anneau Beacon doivent prendre en charge des vitesses Beacon comprises entre 100 µs et 100 ms afin d'assurer toutes les mises en œuvre du superviseur d'anneau |
| Beacon Timeout | Durée pendant laquelle les nœuds doivent attendre avant de temporiser la réception des trames Beacon et d'entamer l'action appropriée (selon qu'il s'agit d'un superviseur ou d'un nœud d'anneau normal). Les superviseurs doivent prendre en charge une plage comprise entre 800 µs et 500 ms. La valeur par défaut doit être 1 960 µs. Les superviseurs peuvent prendre en charge un délai d'attente Beacon inférieur à 800 µs, mais ce n'est pas requis. Le délai d'attente Beacon minimal absolu est de 200 µs |
| Supervisor Precedence | Valeur de priorité attribuée à un superviseur d'anneau et transmise dans les trames Beacon. Utilisée pour sélectionner le superviseur d'anneau actif lorsque plusieurs superviseurs sont configurés.<br>La valeur par défaut est 0. Peut être modifiée par l'intermédiaire de l'objet DLR |
| DLR VLAN ID | VLAN ID utilisé lors de l'envoi des trames de protocole DLR. Le VLAN ID est configuré au niveau du superviseur d'anneau (par l'intermédiaire de l'objet DLR), puis est détecté par les nœuds d'anneau lorsqu'ils sont reçus et traités par les trames Beacon ou Announce provenant du superviseur.<br>La valeur par défaut est 0. En règle générale, il n'est pas utile de modifier le VLAN ID, à moins qu'un commutateur commercial ne soit utilisé dans l'anneau |

### 10.5.2 Superviseur d'anneau

#### 10.5.2.1 Démarrage

Un superviseur d'anneau activé doit démarrer à l'état FAULT_STATE et doit configurer les deux ports pour transférer des trames. Le superviseur doit envoyer des trames Beacon à partir de ses ports, la valeur RING_FAULT_STATE étant attribuée à l'état de l'anneau. Il doit également envoyer des trames Announce à partir de ses ports, la valeur RING_FAULT_STATE étant attribuée à l'état de l'anneau.

Une fois les trames Beacon reçues par les deux ports, le superviseur doit passer à l'état NORMAL_STATE, purger sa table d'adresses MAC à diffusion individuelle, puis reconfigurer l'un de ses ports pour ne pas transférer les paquets, sauf pour les éléments ci-dessous, qui doivent être transférés vers l'hôte pour traitement:

- Les trames Beacon avec l'adresse MAC propre du superviseur (en général nécessaire uniquement pour les mises en œuvre logicielles);
- Les trames Beacon provenant d'autres superviseurs d'anneau;
- Les trames Link_Status/Neighbor_Status;
- Demande ou réponse Neighbor_Check, et Sign_On: transfèrent toujours les trames reçues. Pour les trames diffusées par le superviseur, transfère uniquement celles dont le port source correspond au port bloqué.

Lors de la transition vers NORMAL_STATE, l'état de l'anneau des trames Beacon doit être défini sur RING_NORMAL_STATE. Le superviseur d'anneau doit également envoyer une trame Announce à partir d'un port, l'état de l'anneau étant défini sur RING_NORMAL_STATE.

#### 10.5.2.2 Superviseurs d'anneau multiples

Si plusieurs superviseurs d'anneau sont configurés, chacun d'eux envoie des trames Beacon lorsqu'il passe en ligne. Les trames Beacon contiennent une valeur de priorité de superviseur. Lorsqu'un superviseur reçoit une trame Beacon, il contrôle la valeur de priorité. Si la priorité de la trame Beacon est supérieure à celle du nœud destinataire, ce dernier passe à l'état FAULT_STATE et devient un superviseur de sauvegarde. Si les valeurs de priorité sont identiques, le nœud avec le numéro d'adresse MAC le plus élevé devient le superviseur actif.

Les superviseurs de sauvegarde configurent leurs paramètres DLR avec les valeurs obtenues auprès des trames Beacon du superviseur actif: Beacon Interval, Beacon Timeout, VLAN ID.

Les superviseurs de sauvegarde continuent à surveiller les deux ports pour connaître le délai d'attente des trames Beacon (aucun Beacon reçu dans la période de délai d'attente Beacon). Si le délai d'attente Beacon a expiré sur les deux ports, le superviseur de sauvegarde attend une période de délai d'attente Beacon supplémentaire (au cours de laquelle les autres nœuds passent en mode linéaire), puis commence à envoyer ses propres Beacon de manière à pouvoir sélectionner un nouveau superviseur.

#### 10.5.2.3 Connexion

Afin d'identifier les participants du protocole d'anneau, le superviseur d'anneau actif doit envoyer une trame Sign_On lorsqu'il passe à l'état NORMAL_STATE. Voir les informations Sign_On dans les messages DLR (voir 10.9.9).

#### 10.5.2.4 Fonctionnement normal de l'anneau

Si le superviseur d'anneau actif est à l'état NORMAL_STATE, il doit envoyer des trames Beacon à partir de ses deux ports. Il doit également envoyer une trame Announce par seconde à partir d'un port.

L'un des ports du superviseur actif doit être configuré pour ne pas transférer les trames, avec les exceptions indiquées en 10.3.2.1.

#### 10.5.2.5 Détection de défaut d'anneau

L'un des événements possibles doit amener le superviseur d'anneau actif à passer à l'état FAULT_STATE:

a) Trame Beacon reçue de l'autre superviseur avec valeur de priorité plus élevée;

b) Perte de trames Beacon sur l'un des ports pendant la période spécifiée par la variable Beacon Timeout, indiquant une interruption quelque part dans l'anneau;

c) Détection d'une perte de liaison avec le nœud avoisinant sur l'un des ports;

d) Trame Link_Status reçue à partir d'un nœud d'anneau, indiquant qu'un nœud d'anneau a détecté un défaut.

Dans tous les cas ci-dessus, le superviseur d'anneau actif doit:

- passer à l'état FAULT_STATE;
- purger sa table d'adresses MAC à diffusion individuelle;
- débloquer le port bloqué;
- envoyer la trame Beacon à partir de ses ports, la valeur RING_FAULT_STATE étant attribuée à la variable Ring State;
- envoyer la trame Announce à partir de ses ports, la valeur RING_FAULT_STATE étant attribuée à la variable Ring State.

De plus, dans le cas 2 ci-dessus, le superviseur d'anneau actif doit initier un processus Neighbor Check en émettant une trame Locate Fault. Il doit également émettre sa propre trame Neighbor Check à travers le/les port(s) sur le(s)quel(s) le délai d'attente Beacon a expiré.

A l'état FAULT_STATE, le superviseur d'anneau doit continuer à envoyer des trames Beacon afin de détecter la restauration d'anneau.

#### 10.5.2.6 Restauration d'anneau

Si le superviseur d'anneau actif est à l'état FAULT_STATE, la réception des trames Beacon sur les deux ports doit amener à passer à l'état NORMAL_STATE. Le superviseur d'anneau actif doit:

- purger la table d'adresses MAC à diffusion individuelle;
- reconfigurer ses ports de manière à ne pas transférer le trafic sur un port (avec les exceptions indiquées précédemment);
- envoyer les trames Beacon en attribuant la valeur RING_NORMAL_STATE à la variable Ring State;
- envoyer les trames Announce à partir d'un port, la valeur RING_NORMAL_STATE étant attribuée à la variable Ring State.

#### 10.5.2.7 Modification des paramètres d'anneau

Les paramètres de superviseur d'anneau suivants peuvent être modifiés par l'intermédiaire de l'objet DLR (voir 7.9):

- Valeur de priorité du superviseur;
- Beacon interval;
- Beacon timeout;
- VLAN ID;
- Superviseur activé/désactivé.

Si l'un des paramètres ci-dessus est modifié sur le superviseur d'anneau actif, ce dernier doit cesser d'envoyer des trames Beacon pendant deux périodes Beacon Timeout, puis de nouveau les envoyer en utilisant les nouveaux paramètres. L'arrêt de l'envoi des trames Beacon et Announce permet aux nœuds d'anneau de détecter les nouveaux paramètres lors de la restauration de Beacon. La production des trames Announce est davantage supprimée pour un minimum de 2 nouvelles périodes de beacon timeout pour s'assurer que le

superviseur actif est déterminé avant l'envoi de toute nouvelle trame Announce avec les nouveaux paramètres.

Si les paramètres sont modifiés sur un superviseur d'anneau de sauvegarde, le comportement dépend de la nouvelle valeur de priorité du superviseur de sauvegarde par rapport à celle du superviseur actif:

- la nouvelle valeur de priorité de sauvegarde est supérieure à la priorité du superviseur d'anneau actif en cours ou présente une priorité égale avec un numéro d'adresse MAC supérieur à celui du superviseur actif: la sauvegarde doit immédiatement commencer à envoyer les trames Beacon avec le nouveau paramètre;
- la nouvelle valeur de priorité de sauvegarde est inférieure à la priorité du superviseur d'anneau actif ou présente une priorité égale avec un numéro d'adresse MAC inférieur à celui du superviseur actif: la modification des paramètres Beacon Interval, Beacon Timeout et VLAN ID doit être ignorée.

### 10.5.3 Nœud d'anneau

#### 10.5.3.1 Mises en œuvre Beacon/Announce

Les nœuds d'anneau (à savoir les nœuds non superviseurs) peuvent faire l'objet de différentes mises en œuvre, selon qu'ils sont en mesure ou pas de traiter les trames Beacon qui, par défaut, sont envoyées toutes les 400 µs (mais qui peuvent l'être tous les 100 µs). Les nœuds capables de traiter les trames Beacon bénéficient en général d'une assistance matérielle lors de la mise en œuvre du protocole DLR, de manière à ne pas charger le CPU de l'appareil avec le traitement des trames Beacon.

Les appareils ayant besoin de traiter les trames Beacon dans le CPU peuvent configurer leur commutateur intégré pour simplement transmettre les trames Beacon sur le réseau sans interprétation ni traitement approfondi. Toutefois, ces appareils doivent traiter les trames Announce, qui indiquent également l'état de l'anneau, mais sont envoyées à une vitesse plus réduite.

NOTE Il est possible de mettre en œuvre un nœud Beacon sans assistance matérielle, pourvu que la capacité du CPU de l'appareil soit suffisante pour traiter les trames Beacon en plus de ses autres fonctions requises.

Il est préférable que les mises en œuvre d'appareils reposent sur Beacon plutôt que sur Announce, étant donné que les nœuds d'anneau capables de traiter des trames Beacon assurent de meilleures performances de remise de l'anneau. Voir les analyses de performance en 10.10.4.

#### 10.5.3.2 Démarrage – Beacon

Un nœud d'anneau Beacon doit démarrer à l'état IDLE_STATE, ce qui implique la présence d'un réseau en mode de topologie linéaire.

Lors de la réception d'une trame Beacon par un port, le nœud doit passer à l'état FAULT_STATE, ce qui implique un mode de topologie en anneau. Le nœud d'anneau doit purger sa table d'adresses MAC à diffusion individuelle et sauvegarder les paramètres de superviseur d'anneau à partir de la trame Beacon:

- Adresse MAC du superviseur;
- Valeur de priorité du superviseur;
- Beacon timeout;
- VLAN ID.

Lors de la réception de trames Beacon par les deux ports et après réception d'une trame Beacon d'un superviseur d'anneau actif avec un champ d'état d'anneau défini sur

RING_STATE_NORMAL sur l'un de ses ports, le nœud doit effectuer une transition vers NORMAL_STATE et purger son tableau d'adresse MAC à diffusion individuelle.

#### 10.5.3.3 Démarrage – Announce

Un nœud d'anneau Announce doit démarrer à l'état IDLE_STATE, ce qui implique la présence d'un réseau en mode de topologie linéaire.

Lors de la réception d'une trame Announce par un port, le nœud doit passer à l'état de l'anneau indiqué dans la trame Announce. Le nœud d'anneau doit purger sa table d'adresses MAC à diffusion individuelle et sauvegarder les paramètres de superviseur d'anneau à partir de la trame Announce:

- Adresse MAC du superviseur;
- Etat de l'anneau;
- VLAN ID.

#### 10.5.3.4 Détection de défaut

L'un des événements possibles doit amener un nœud d'anneau à quitter l'état NORMAL_STATE:

Pour les nœuds Beacon:

a) la réception d'une trame Beacon dont la valeur RING_FAULT_STATE est attribuée à la variable Ring State;

b) la réception d'une trame Beacon avec une adresse MAC différente et une priorité plus élevée que le superviseur d'anneau en cours;

c) la perte de trames Beacon sur les deux ports pendant la période spécifiée par la variable Beacon Timeout, qui amène le nœud à passer à l'état IDLE_STATE (c'est-à-dire que la topologie est désormais linéaire);

d) la perte de trame Beacon sur un seul port pendant la période spécifiée par la variable Beacon Timeout.

Pour les nœuds Announce:

a) la réception d'une trame Announce dont la valeur RING_FAULT_STATE est attribuée à la variable Ring State;

b) la perte d'une trame Announce pendant la durée de délai d'attente Announce, qui amène le nœud à passer à l'état IDLE_STATE (c'est-à-dire que la topologie est désormais linéaire).

Dans tous les cas ci-dessus, le nœud d'anneau doit:

- purger sa table d'adresses MAC à diffusion individuelle;
- passer à l'état FAULT_STATE (sauf: la perte de trame Beacon sur les deux ports ou perte de trame Announce amène à passer à l'état IDLE_STATE).

#### 10.5.3.5 Restauration d'anneau

Pour les nœuds d'anneau, le processus de restauration est identique au cas Démarrage (voir 10.5.3.2 et 10.5.3.3).

#### 10.5.3.6 Processus Sign_On

La trame Sign_On permet d'identifier tous les participants de l'anneau. Le superviseur d'anneau actif doit envoyer une trame Sign_On lors du passage à l'état NORMAL_STATE. Le superviseur actif transmet une trame Sign_On par minute à l'état NORMAL_STATE, tant qu'il ne reçoit pas une trame Sign_On qu'il a déjà envoyé. Lors de la réception d'une trame de ce

type, le superviseur actif cesse d'envoyer d'autres trames Sign_On jusqu'au passage suivant à l'état NORMAL_STATE. La liste des participants peut être accédée par l'intermédiaire de l'objet DLR.

La trame Sign_On est un message multidiffusion à partir d'un port du superviseur d'anneau actif. Le nœud participant de l'anneau retient la trame Sign_On et la transfère uniquement vers le CPU hôte. Le CPU hôte incrémente le nombre de nœuds de la liste, ajoute ses propres adresses à la liste et transmet la trame Sign_On uniquement par l'autre port que le port destinataire.

La trame Sign_On est transmise d'un nœud participant de l'anneau vers le suivant de manière analogue et atteint éventuellement le superviseur actif. Le superviseur actif peut identifier la trame Sign_On qu'il a envoyé en confirmant que la première entrée lui appartient.

Il est possible que l'anneau contienne suffisamment de nœuds et que les adresses de tous les nœuds n'entrent pas dans la trame Sign_On. Lorsqu'un nœud reçoit la trame Sign_On, si l'ajout de l'adresse du nœud dépasse la taille de trame maximale, le nœud n'ajoute pas son adresse, mais sauvegarde le port par lequel la trame a été reçue, puis envoie la trame Sign_On directement au superviseur d'anneau actif.

Lorsque le superviseur d'anneau reçoit la trame Sign_On envoyée à son adresse MAC à diffusion individuelle, il suppose que cela est dû au fait que la trame Sign_On a atteint sa taille maximale. Le superviseur redémarre le processus de connexion en envoyant une nouvelle trame Sign_On directement (diffusion individuelle) au nœud duquel il a reçu la trame Sign_On à diffusion individuelle.

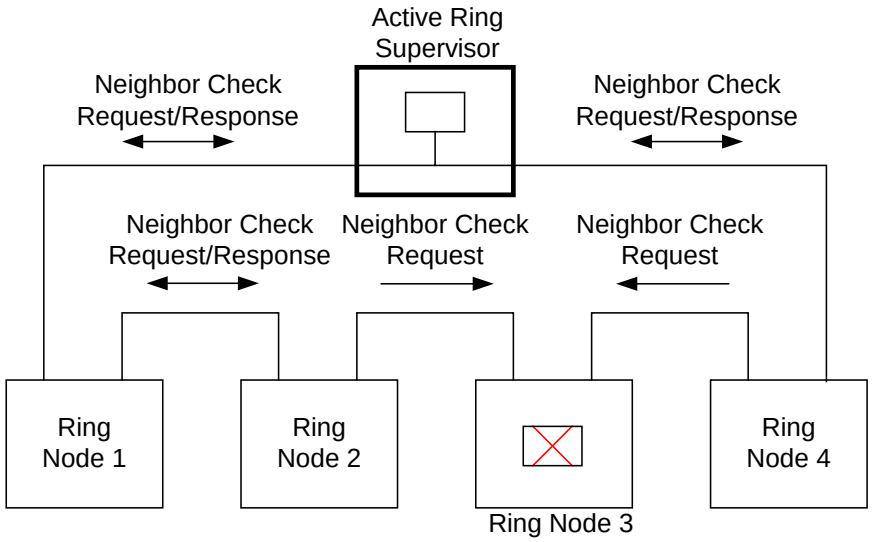
Lors de la réception de la nouvelle trame Sign_On provenant du superviseur d'anneau, le nœud d'anneau ajoute son adresse à la trame Sign_On. Ensuite, le nœud envoie la trame Sign_On (multidiffusion) par un autre port que le port sauvegardé.

#### 10.5.3.7 Processus Neighbor_Check

Lorsque le superviseur d'anneau actif détecte la perte de Beacon, il envoie une trame Locate_Fault par les deux ports.

Lors de la réception de la trame Locate_Fault, chaque nœud d'anneau émet une demande Neighbor_Check par le port sur lequel la perte de trame Beacon a été détectée (nœuds Announce envoyés par les deux ports). Le superviseur émet également sa propre demande Neighbor_Check.

Lorsqu'un nœud reçoit une trame de demande Neighbor_Check, il répond par une trame de réponse Neighbor_Check par le port sur lequel la demande d'origine a été reçue. Si le nœud qui envoie la demande Neighbor_Check ne reçoit pas de réponse en 100 ms, il doit de nouveau tenter la demande. Si aucune réponse n'est reçue après 3 tentatives, à l'issue du délai d'attente global de 300 ms, le nœud envoie une trame Neighbor_Status au superviseur d'anneau.




**Légende**

| Anglais | Français |
|---|---|
| Active ring supervisor | Superviseur d'anneau actif |
| Neighbor Check request/response | Demande/réponse Neighbor_Check |
| Ring node | Nœud d'anneau |

**Figure 32 – Processus Neighbor Check**

La Figure 32 illustre le processus Neighbor_Check. Dans cet exemple, les nœuds d'anneau sains répondent, ce qui n'est pas le cas du nœud d'anneau défaillant 3. Les nœuds d'anneau 2 et 4 enverront chacun une trame Neighbor_Status au superviseur d'anneau.

## 10.6 Exigences de mise en œuvre

### 10.6.1 Exigences et recommandations relatives au commutateur intégré

Les exigences et recommandations générales ci-dessous concernent tous les appareils mettant en œuvre la technologie de commutateur intégré (si elle est mise en œuvre par l'intermédiaire des puces disponibles dans le commerce FPGA, ASIC, par exemple).

– Opération ISO/CEI 8802-3:
  - auto-négociation, 10/100 Mbit/s, mode duplex non simultané/simultané (obligatoire);
  - paramétrage forcé de la vitesse/duplex (obligatoire);
  - ISO/CEI 8802-3 contrôle de flux en mode duplex simultané (recommandé).

– Auto MDIX (Medium Dependent Interface Crossover), en modes d'auto-négociation et vitesse/duplex forcé (obligatoire):

NOTE Il s'agit d'une question de PHY et de transformateur, pas de commutateur intégré.

- QoS:
  a) 2 files d'attente (obligatoire); 4 (recommandé);
  b) file d'attente de priorité élevée pour les trames DLR, avec planification à priorité stricte (obligatoire);
  c) priorisation via IEEE 802.1D/IEEE 802.1Q (obligatoire) et DSCP (vivement recommandé). L'utilisation doit être cohérente avec le schéma CP 2/2 QoS spécifié en 7.9.6.4. Pour les trames IP, le commutateur intégré doit utiliser la valeur DSCP. Pour les trames non IP, la priorité de l'en-tête IEEE 802.1Q doit être utilisée.

– Limite de vitesse de diffusion pour le CPU hôte (recommandé). Le seuil de diffusion toléré par un appareil dépend du CPU hôte. En général, il convient de déclencher la limite de vitesse de diffusion lorsque le trafic de diffusion dépasse 1 % de la durée de transmission de liaison;

– Filtrage des données entrantes à diffusion individuelle et multiple vers le CPU hôte (recommandé mais, dans la pratique, la plupart des appareils l'exigeront).

### 10.6.2 Exigences de mise en œuvre DLR

Les exigences de mise en œuvre suivantes s'appliquent aux nœuds DLR, qu'il s'agisse de superviseurs d'anneau ou de nœuds d'anneau:

- conserver le VLAN ID IEEE 802.1Q et la priorité de balise des trames de protocole d'anneau;
- désactiver le filtrage multidiffusion IP sur les ports d'anneau ou purger la table de filtrage multidiffusion des ports d'anneau lors des transitions d'état de l'anneau;
- configurer l'adresse multidiffusion des trames Beacon à transférer sur les ports d'anneau et vers le CPU hôte des mises en œuvre Beacon;
- configurer l'adresse multidiffusion des trames Announce et Locate_Fault à transférer vers le CPU hôte et sur les ports d'anneau;
- configurer l'adresse multidiffusion de demande/réponse Neighbor_Check et Sign_On à transférer uniquement vers le CPU hôte;
  - mécanisme de balisage du port par lequel une trame de ce type a été reçue de l'anneau;
  - mécanisme de transfert des trames du CPU hôte de l'anneau uniquement par le port duquel elles étaient censées sortir;
- configurer l'adresse MAC à diffusion individuelle du superviseur d'anneau actif de manière à transférer les trames du superviseur sur les deux ports;
- purger les tables d'adresses MAC à diffusion individuelle sur les transitions d'état de l'anneau (ou désactiver l'apprentissage);
- configurer l'adresse MAC à diffusion individuelle de manière à ne pas la purger lorsque la table d'adresses MAC est purgée;
- mettre en œuvre les attributs Interface Counters et Media Counters de l'objet de liaison Ethernet (voir 7.6) pour faciliter la surveillance du réseau;
- mettre en œuvre l'objet QoS avec au moins un marquage DSCP du trafic CP 2/2 généré par l'appareil;
- recommandé: configurer la liste de contrôle d'accès ou un autre mécanisme adapté pour retirer les trames de l'appareil du réseau lors de leur réception (lors de la restauration/le démarrage de l'anneau, par exemple).
- désactiver ISO/CEI 8802-3 et d'autres mécanismes de contrôle de flux matériel sur les ports d'anneau
- les méthodes 802.3 permettant de déterminer la présence d'une liaison lorsqu'un port est configuré pour fonctionner en mode vitesse/duplex forcé produisent fréquemment des transitions d'état de liaison transitoire lorsqu'un câble est inséré. Pour éviter les transitions d'état de l'anneau lors de l'insertion d'un câble, les appareils doivent s'assurer que la liaison est stable avant d'activer le transfert des trames depuis/vers le port associé ou de générer la trame Link_Status. Habituellement, cette opération peut être réalisée en ajoutant un antirebond (par exemple, un retard compris entre 10 ms et 100 ms selon la mise en œuvre matérielle) pour obtenir des transitions de liaison stables.

### 10.6.3 Considérations relatives à la CEI 61588 et CP 2/2.1

Pour prendre en charge les applications nécessitant une synchronisation temporelle, il est recommandé d'utiliser des appareils à ports multiples pour prendre en charge les capacités suivantes conformément à la CEI 61588 et au CP 2/2.1:

- Mise en œuvre de l'horloge transparente de bout en bout CEI 61588;
- Les appareils qui mettent également en œuvre la fonctionnalité d'horloge « ordinary/boundary clock » de la CEI 61588 doivent mesurer le retard de chemin à l'aide des trames Delay_Req/Delay_Resp à chaque modification de l'état de l'anneau ou du mode de topologie du réseau;
- Les appareils qui mettent en œuvre la fonctionnalité d'horloge « ordinary/boundary clock » de la CEI 61588 et qui sont connectés indirectement au réseau en anneau doivent mesurer le retard de chemin à l'aide des trames Delay_Req/Delay_Resp conformément au message de signalisation du profil CEI 61588 spécifique à CPF 2 (voir l'objet Time Sync dans la CEI 61158-5-2 et la CEI 61158-6-2).

Les appareils qui ne mettent pas en œuvre ces fonctions souffriront d'une faible précision de synchronisation pendant de courtes périodes après la reconfiguration d'un réseau.

#### 10.6.4 Considérations relatives au STP/RSTP/MSTP de l'IEEE 802.1D/IEEE 802.1Q

Pour que DLR cohabite avec les protocoles IEEE STP, RSTP et MSTP, le superviseur d'anneau actif ne doit pas transférer les trames multidiffusion avec l'adresse de destination 01:80:C2:00:00:00 (trames BPDU) d'un port d'anneau à l'autre, quel que soit l'état de l'anneau. Toutefois, si le superviseur d'anneau actif contient des ports qui ne sont pas des ports d'anneau (dans le cas d'un commutateur, par exemple), il doit transférer lesdites trames multidiffusion entre ces ports et uniquement entre un port d'anneau et les ports de ce type. Ce comportement doit être mis en œuvre afin de s'assurer que le protocole STP/RSTP/MSTP détecte correctement toute boucle passant par un anneau autre que celui du superviseur d'anneau actif.

### 10.7 Utilisation des nœuds non DLR dans le réseau en anneau

#### 10.7.1 Considérations générales

De par sa conception, le protocole DLR n'oblige pas tous les autres nœuds de l'anneau à le mettre en œuvre. Les nœuds non DLR peuvent être placés dans l'anneau, à condition qu'ils prennent en charge certaines capacités requises dans les paragraphes 10.7.2 à 10.7.5.

Il convient d'informer l'utilisateur final que l'utilisation de nœuds non DLR dans l'anneau peut avoir un impact sur les performances, rallongeant les temps de remise de l'anneau (voir les analyses de performance en 10.11). Pour de meilleures performances, il est vivement recommandé que tous les nœuds de l'anneau mettent en œuvre le protocole DLR.

Si l'utilisation d'appareils non DLR s'avère inévitable, il est vivement recommandé de les connecter au réseau par l'intermédiaire d'un appareil compatible DLR, par exemple un commutateur à 3 ports (2 ports connectés au réseau, 1 port connecté à l'appareil non DLR).

En cas d'utilisation de nœuds non DLR directement dans l'anneau, certaines capacités et/ou étapes de configuration de ces nœuds sont nécessaires au bon fonctionnement du protocole DLR. Ces capacités et étapes de configuration sont décrites plus en détails dans les paragraphes 10.7.2 à 10.7.5.

#### 10.7.2 Stations d'extrémité non DLR

Des exemples d'appareils non DLR peuvent inclure un module ou une unité d'E-S à 2 ports existant qui prend en charge la technologie de commutateur intégré, mais qui n'a pas mis en œuvre le protocole DLR. Ces appareils peuvent être insérés directement dans le réseau en anneau. Toutefois, afin d'assurer le bon fonctionnement de l'anneau, la station d'extrémité DLR doit comporter les capacités ci-dessous.

- Désactiver l'apprentissage d'adresse MAC à diffusion individuelle (ou ne pas utiliser du tout l'apprentissage d'adresse MAC). Etant donné que les trames Beacon arriveront sur les deux ports avec l'adresse MAC du superviseur, l'apprentissage d'adresse amènera le MAC du superviseur à passer d'un port au suivant. Si le superviseur est également un appareil d'E-S, les communications E-S peuvent être interrompues.

- Désactiver le filtrage multidiffusion sur les ports d'anneau DLR. S'il n'est pas désactivé, les messages multidiffusion risquent de ne pas être transférés vers d'autres nœuds d'anneau.
- Prendre en charge les trames IEEE 802.1Q et conserver le VLAN ID et la priorité de balise. Les messages DLR peuvent être supprimés si les trames ne sont pas prises en charge ou peuvent ne pas être correctement placés dans la file d'attente si la priorité de balise n'est pas conservée.
- Mettre en œuvre les files d'attente de priorité avec la file d'attente à priorité la plus élevée utilisée pour les messages DLR, avec planification de priorité stricte. Si elles ne sont pas prises en charge, les performances de remise de l'anneau peuvent être affectées.

Il revient à l'utilisateur final de s'assurer que l'appareil non DLR prend en charge les capacités ci-dessus.

### 10.7.3 Commutateurs non DLR

Il est possible de placer directement dans l'anneau les commutateurs Ethernet non DLR disponibles dans le commerce. Par exemple, un utilisateur peut connecter un anneau de stations d'extrémité dans un commutateur afin de connecter les stations d'extrémité dans le réseau élargi de l'utilisateur.

Pour simplifier la configuration du commutateur, il est recommandé de connecter le commutateur non DLR à l'anneau par l'intermédiaire d'un appareil compatible DLR (un simple commutateur à 3 ports, par exemple). Si le commutateur non DLR est directement connecté à l'anneau, de nombreuses étapes de configuration sont requises. Les particularités des étapes de configuration peuvent varier en fonction du fournisseur. En général, les étapes de configuration requises sont présentées en 10.7.4 et 10.7.5.

L'utilisation de commutateurs non DLR peut donner lieu à la perte temporaire de trames à diffusion individuelle suite au défaut ou à la restauration de l'anneau. Après le défaut/la restauration d'un anneau, les tables d'apprentissage MAC du commutateur peuvent être non valides, étant donné qu'il est possible d'atteindre les appareils par différents ports. Tant que les tables d'apprentissage MAC ne sont pas mises à jour après l'envoi des trames par les appareils, les trames à diffusion individuelle risquent de ne pas atteindre les appareils cible.

Dans le cas d'une connexion d'E-S CP 2/2, un cycle de production sera en général perdu. Dans certains cas, plusieurs productions peuvent être perdues. Par exemple, si T-O RPI est plus lent que O-T RPI, les paquets O-T ne seront pas remis tant que l'appareil cible n'envoie pas un paquet pour mettre à jour la table MAC du commutateur. S'il s'agit d'un appareil interrogé (un serveur de messages explicite, par exemple) ou doté uniquement de connexions unidirectionnelles, l'appareil peut ne pas générer de paquets pour mettre à jour la table d'apprentissage MAC du commutateur.

En cas de perte intempestive de paquets à diffusion individuelle, plusieurs options s'offrent à l'utilisateur:

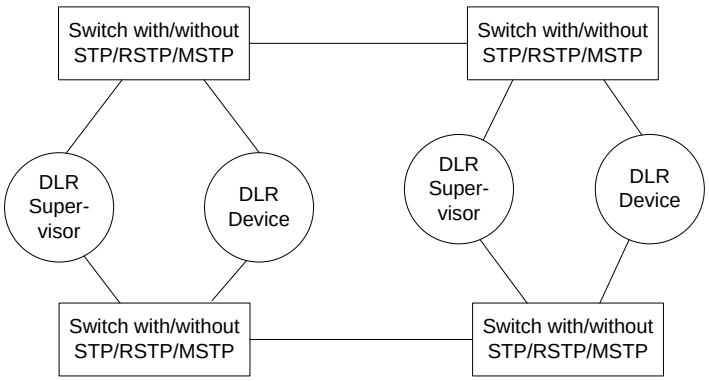
a) configurer les adresses MAC statiques à diffusion individuelle pour les ports de commutateur connectés à l'anneau;

b) désactiver l'apprentissage MAC à diffusion individuelle dans le commutateur;

c) utiliser un appareil DLR (un commutateur à 3 ports, par exemple) pour connecter le commutateur le plus important à l'anneau.

Comme indiqué en 10.2, les anneaux de protocole DLR peuvent être utilisés dans les topologies dotées de commutateurs exécutant les protocoles IEEE Spanning Tree Protocols (STP, RSTP, MSTP). Dans la plupart des configurations habituelles (voir 10.2), les anneaux DLR peuvent sans problème être connectés à ces commutateurs gérés.

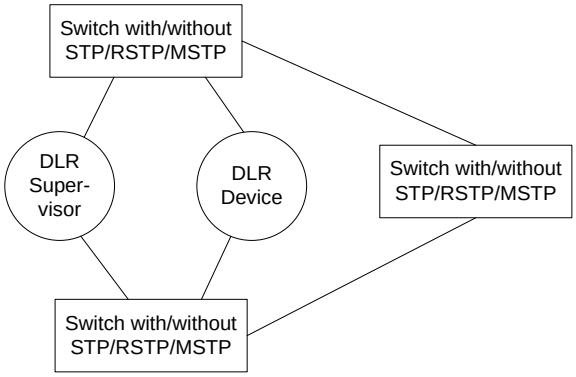
Le protocole DLR ne prend pas en charge les boucles non DLR traversant des parties d'un anneau DLR, qu'un commutateur prenne en charge ou pas DLR et les protocoles STP. La Figure 33 et la Figure 34 présentent des exemples de topologies DLR non prises en charge, avec des boucles non DLR traversant l'anneau DLR. Ces topologies peuvent générer un comportement imprévisible du réseau, qui peut être résolu en supprimant une ou plusieurs liaisons redondantes non DLR.




**Légende**

| Anglais | Français |
|---|---|
| Switch with/without | Commutateur avec/sans |
| DLR supervisor | Superviseur DLR |
| DLR device | Appareil DLR |

**Figure 33 – Topologie non prise en charge – Exemple 1**




**Légende**

| Anglais | Français |
|---|---|
| Switch with/without | Commutateur avec/sans |
| DLR supervisor | Superviseur DLR |
| DLR device | Appareil DLR |

**Figure 34 – Topologie non prise en charge – Exemple 2**

### 10.7.4 Configuration du commutateur pour les anneaux non VLAN

Le Paragraphe 10.7.4 présente la procédure générale de configuration d'un commutateur géré pour un réseau en anneau non VLAN. Les commandes réelles de configuration d'un commutateur particulier dépendent du commutateur, mais peuvent être déduites de la procédure ci-dessous.

a) Configurer la qualité de service (QoS) pour assurer le comportement déterministe et les performances élevées pour le fonctionnement de l'anneau. Le commutateur doit prendre en charge au moins deux files d'attente (quatre de préférence), ainsi que la planification à priorité stricte pour la file d'attente à priorité la plus élevée. La file d'attente à priorité la plus élevée sur les ports connectés au réseau en anneau doit être configurée pour les trames de protocole d'anneau (IEEE 802.1Q priorité 7). Alors qu'il est acceptable de partager la file d'attente à priorité la plus élevée avec les messages d'événement PTP de la CEI 61588, il est vivement recommandé de ne partager aucun autre trafic sur cette file d'attente. Pour les trames IP, le commutateur doit utiliser la valeur DSCP. Pour les trames non IP, la priorité de l'en-tête IEEE 802.1Q doit être utilisée.

b) Le cas échéant, configurer les deux ports du commutateur connecté au réseau en anneau pour conserver la priorité de balise IEEE 802.1Q des trames de protocole d'anneau lorsqu'elles passent par les ports. La plupart des commutateurs ne feront l'objet d'aucune configuration particulière pour cette étape.

c) Désactiver le filtrage multidiffusion IP sur les deux ports du commutateur connecté à l'anneau. Cette étape doit permettre de faciliter la remise ininterrompue des données de connexion multidiffusion CP 2/2 après la reconfiguration d'un anneau. Certains commutateurs offrent un moyen direct de configuration du transfert du trafic multidiffusion IP par port. D'autres commutateurs offrent un moyen de configurer la désignation de ports individuels en tant que ports de routeur multidiffusion. Ces deux méthodes permettent de désactiver le filtrage multidiffusion sur des ports spécifiques. Pour plus de détails, voir l'IETF RFC 4541.

d) Configurer statiquement les trois adresses multidiffusion utilisées par le protocole d'anneau à transférer uniquement sur deux ports du commutateur connecté à l'anneau. Cette étape doit permettre d'éviter de transférer les trames de protocole d'anneau multidiffusion sur d'autres ports du commutateur.

e) Configurer les adresses MAC multidiffusion de tous les superviseurs d'anneau configurés statiquement dans la table MAC du commutateur, de manière à transférer le trafic multidiffusion destiné aux superviseurs d'anneau par les deux ports du commutateur connecté à l'anneau. Cette étape doit permettre d'éviter que le commutateur ne confonde les trames Beacon de l'anneau bidirectionnel et le superviseur d'anneau actif.

### 10.7.5 Configuration du commutateur pour l'anneau VLAN

Le Paragraphe 10.7.5 présente la procédure générale de configuration d'un commutateur géré pour un réseau en anneau VLAN. Les commandes de configuration réelles dépendent du commutateur, mais peuvent être déduites de la procédure ci-dessous. Cette procédure suppose que tous les superviseurs d'anneau ont été configurés pour utiliser le VLAN ID *ring_vlan_id* pour les trames de protocole d'anneau. L'ID *ring_vlan_id* sera uniquement utilisé pour les trames de protocole d'anneau. Le commutateur sera censé utiliser le VLAN ID *default_vlan_id* pour tous les trafics de trame non balisés, y compris les trames CP 2/2.

a) Configurer la qualité de service (QoS) pour assurer le comportement déterministe et les performances élevées pour le fonctionnement de l'anneau. Le commutateur doit prendre en charge au moins deux files d'attente (quatre de préférence), ainsi que la planification à priorité stricte pour la file d'attente à priorité la plus élevée. La file d'attente à priorité la plus élevée sur les ports connectés au réseau en anneau doit être configurée pour les trames de protocole d'anneau (IEEE 802.1Q priorité 7). Alors qu'il est acceptable de partager la file d'attente à priorité la plus élevée avec les messages d'événement PTP de la CEI 61588, il est vivement recommandé de ne partager aucun autre trafic sur cette file d'attente. Pour les trames IP, le commutateur doit utiliser la valeur DSCP. Pour les trames non IP, la priorité de l'en-tête IEEE 802.1Q doit être utilisée.

b) Créer des VLAN pour *ring_vlan_id* et *default_vlan_id* sur le commutateur.

c) Configurer les deux ports de l'anneau connecté au commutateur pour participer à *ring_vlan_id* et *default_vlan_id* du VLAN. Les deux ports doivent être configurés de manière à ce que le trafic sur *default_vlan_id* ne soit pas balisé à la sortie et que *default_vlan_id* soit attribué au trafic entrant non balisé. Les deux ports doivent être configurés pour préserver la balise VLAN sur *ring_vlan_id* lorsque les trames de protocole d'anneau traversent les ports. Il est nécessaire de vérifier qu'aucun autre port du commutateur ne participe à VLAN *ring_vlan_id*.

d) Désactiver le filtrage multidiffusion IP sur les deux ports du commutateur connecté à l'anneau, mais uniquement pour le VLAN *default_vlan_id*. Cette étape doit permettre de faciliter la remise sans mémoire annexe des données de connexion multidiffusion CP 2/2 après la reconfiguration d'un anneau. Certains commutateurs offrent un moyen direct de configuration du transfert du trafic multidiffusion IP par port. D'autres commutateurs offrent un moyen de configurer la désignation de ports individuels en tant que ports de routeur multidiffusion. Ces deux méthodes permettent de désactiver le filtrage multidiffusion sur des ports spécifiques. Pour plus de détails, voir l'IETF RFC 4541.

e) Configurer les adresses MAC multidiffusion de tous les superviseurs d'anneau configurés statiquement dans la table MAC du commutateur, de manière à transférer le trafic multidiffusion destiné aux superviseurs d'anneau par les deux ports du commutateur connecté à l'anneau. Cette configuration doit être réalisée pour les VLAN *ring_vlan_id* et *default_vlan_id*. Cette étape doit permettre d'éviter que le commutateur ne confonde les trames Beacon de l'anneau bidirectionnel et le superviseur d'anneau actif. Cette étape peut être ignorée, mais donnera lieu à une légère perte de performance.

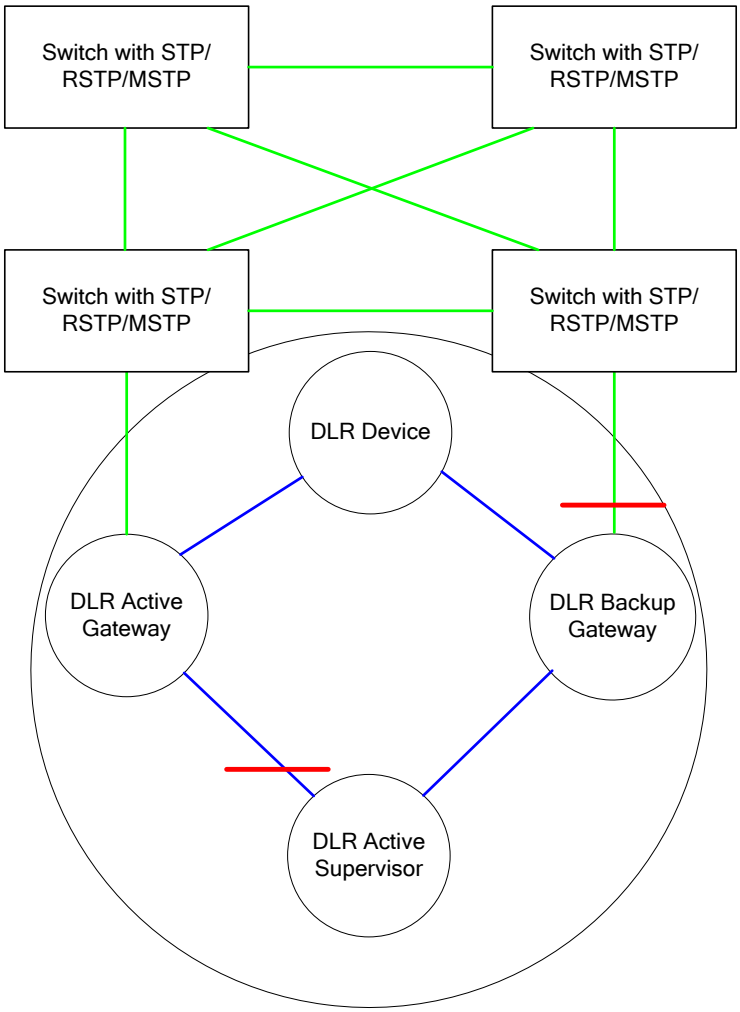
## 10.8 Appareils passerelle redondants sur un réseau DLR

### 10.8.1 Généralités

Les appareils passerelle redondants sur un réseau DLR permettent d'établir plusieurs connexions à l'infrastructure réseau résidant hors du réseau DLR. Si l'un des appareils passerelle échoue ou que la connexion d'un appareil passerelle à l'infrastructure réseau extérieure échoue, une liaison secondaire sera établie pour les communications. Le Paragraphe 10.8 spécifie le comportement des appareils passerelle redondants sur un réseau DLR.

### 10.8.2 Topologies prises en charge

La Figure 35 illustre un réseau DLR connecté à une infrastructure réseau hors du réseau DLR par l'intermédiaire d'appareils passerelle redondants doubles. En général, deux appareils passerelle redondants ou plus sur un réseau DLR sont pris en charge. L'un des appareils passerelle jouera le rôle d'appareil passerelle actif tandis que les autres joueront le rôle d'appareils passerelle de sauvegarde. A tout moment, seul l'appareil passerelle actif transmettra le trafic entre le réseau DLR et l'infrastructure réseau hors du réseau DLR. Les appareils passerelle de sauvegarde bloqueront tout le trafic entre le réseau DLR et l'infrastructure réseau hors du réseau DLR, ce qui a pour effet d'isoler les deux réseaux.




**Légende**

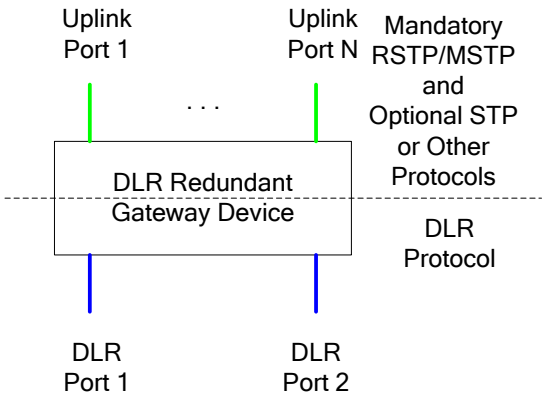
| Anglais | Français |
|---|---|
| Switch with STP/RSTP/MSTP | Commutateur exécutant STP/RSTP/MSTP |
| DLR Device | Appareil DLR |
| DLR Active Gateway | Passerelle active DLR |
| DLR Backup Gateway | Passerelle de sauvegarde DLR |
| DLR Active Supervisor | Superviseur actif DLR |

**Figure 35 – Anneau DLR connecté aux commutateurs via des passerelles redondantes**

Lorsque des appareils passerelle redondants sont utilisés sur un réseau DLR pour fournir plusieurs connexions à l'infrastructure réseau hors du réseau DLR, les appareils passerelle non redondants DLR et les commutateurs non DLR ne doivent pas être utilisés sur le réseau DLR pour se connecter à l'infrastructure réseau hors du réseau DLR.

### 10.8.3 Appareil passerelle redondant

La Figure 36 illustre les éléments conceptuels d'un appareil passerelle redondant DLR.




**Légende**

| Anglais | Français |
|---|---|
| DLR Redundant Gateway Device | Appareil passerelle redondant DLR |
| Uplink Port 1 | Port de liaison montante 1 |
| Uplink Port N | Port de liaison montante N |
| Mandatory RSTP/MSTP and Optional STP or Other Protocols | RSTP/MSTP obligatoire et STP facultatif ou autres protocoles |
| DLR Protocol | Protocole DLR |
| DLR Port 1 | Port DLR 1 |
| DLR Port 2 | Port DLR 2 |

**Figure 36 – Appareil passerelle redondant DLR**

Deux ports DLR se connectent au réseau DLR, et un ou plusieurs ports de liaison montante se connectent à l'infrastructure réseau hors du réseau DLR. Tous les ports non DLR sur un appareil passerelle peuvent ne pas être des ports de liaison montante. Le fait qu'un port non DLR puisse se comporter comme un port de liaison montante dépend à la fois de la mise en œuvre et de la configuration de l'utilisateur final.

L'appareil passerelle doit mettre en œuvre le protocole DLR sur ses deux ports DLR. Il doit mettre en œuvre soit IEEE 802.1D RSTP soit IEEE 802.1Q MSTP sur ses ports de liaison montante. De manière facultative, il peut aussi mettre en œuvre STP ou d'autres protocoles sur ses ports de liaison montante. Il doit être capable de bloquer le trafic pour ne pas qu'il soit transmis entre les ports DLR et les ports de liaison montante lorsqu'il joue le rôle d'appareil passerelle de sauvegarde. En cas de blocage du trafic en mode sauvegarde, il doit transmettre le trafic DLR seulement entre ses deux ports DLR et doit transmettre le trafic des ports de liaison montante seulement entre ses ports de liaison montante. Il existe plusieurs moyens de mettre en œuvre un tel comportement de blocage. Par exemple, un moyen consiste à recourir à un balisage VLAN additionnel à l'intérieur de l'appareil passerelle.

Qu'une passerelle bloque ou transmette le trafic entre les ports DLR et les ports de liaison montante, il ne doit jamais transmettre des trames de protocole DLR à l'exception des trames Learning_Update entre les ports DLR et les ports de liaison montante. Seul l'appareil passerelle actif doit transmettre les trames Learning_Update aux ports de liaison montante lorsque la transmission de trafic est activée entre les ports DLR et les ports de liaison montante comme indiqué en 10.10.4.

Lorsqu'un changement de passerelle survient, le nouvel appareil passerelle actif doit envoyer sur ses ports de liaison montante le message de notification de changement de topologie qui est approprié au protocole actuellement activé sur ses ports de liaison montante. Cette exigence ne s'applique pas si aucun protocole n'est actuellement activé sur ses ports de liaison montante.

### 10.8.4 Comportement de l'appareil passerelle redondant

#### 10.8.4.1 Généralités

Un appareil passerelle redondant doit mettre en œuvre deux diagrammes d'états indépendants. Un diagramme d'états doit correspondre à la fonction DLR de base, autrement dit un nœud d'anneau Beacon ou un nœud d'anneau Announce ou un superviseur DLR tel qu'approprié selon la capacité de l'appareil. Un second diagramme d'états doit être mis en œuvre pour la fonction passerelle redondante. Le Paragraphe 10.8.4 fournit une vue d'ensemble générale du comportement de l'appareil passerelle redondant. Voir 10.10.4 pour plus d'informations.

#### 10.8.4.2 Variables

Le Tableau 135 récapitule les variables utilisées dans le comportement et les messages de la passerelle redondante. L'objet DLR (voir 7.9) expose ces variables (à l'exception de Gateway State et Active Listen Timeout) par l'intermédiaire des attributs d'objet.

**Tableau 135 – Variables de la passerelle redondante**

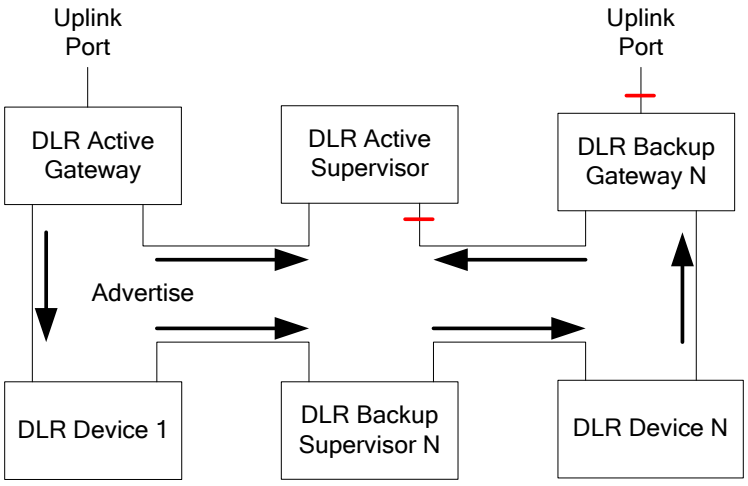
| Variable de la passerelle | Description |
|---|---|
| Gateway State | Etat interne du diagramme d'états des passerelles d'un nœud.<br>IDLE_STATE – état initial des passerelles et lorsque la fonction de passerelle redondante n'a pas été activée.<br>ACTIVE_LISTEN_STATE – lorsqu'une passerelle transmet de manière active des trames Advertise et écoute les trames Advertise pour détecter la présence d'une autre passerelle active tandis que la transmission de trafic est bloquée entre ses ports DLR et les ports de liaison montante.<br>ACTIVE_NORMAL_STATE – fonction normale en mode passerelle active.<br>BACKUP_NORMAL_STATE – fonction normale en mode passerelle de sauvegarde.<br>FAULT_STATE – lorsque le défaut de connexion de liaison montante persiste. |
| Redundant Gateway Enable | Valeur booléenne indiquant si la fonction de passerelle redondante est activée sur un appareil passerelle redondant. La valeur par défaut est FALSE. |
| Gateway Precedence | Valeur de priorité attribuée à une passerelle et transmise dans les trames Advertise. Utilisée pour sélectionner la passerelle active lorsque plusieurs passerelles ont été configurées.<br>La valeur par défaut est 0. Peut être modifiée par l'intermédiaire de l'objet DLR. |
| Advertise Interval | Intervalle selon lequel le nœud de la passerelle active envoie des trames Advertise. Les passerelles doivent prendre en charge une plage comprise entre 1 000 microsecondes et 100 millisecondes. La valeur par défaut doit être 2 000 microsecondes. Les passerelles peuvent prendre en charge un intervalle Advertise inférieur à 1 000 microsecondes, mais ce n'est pas requis. L'intervalle Advertise minimal absolu est de 200 microsecondes. |
| Advertise Timeout | Durée pendant laquelle les nœuds de la passerelle active doivent attendre avant de temporiser la réception des trames Advertise et d'entamer l'action appropriée. Les passerelles doivent prendre en charge une plage comprise entre 2 500 microsecondes et 500 millisecondes. La valeur par défaut doit être 5 000 microsecondes. Les passerelles peuvent prendre en charge un délai d'attente Advertise inférieur à 2 500 microsecondes, mais ce n'est pas requis. Le délai d'attente Advertise minimal absolu est de 500 microsecondes. |
| Learning Update Enable | Valeur booléenne indiquant s'il convient que tous les nœuds du réseau DLR transmettent les trames Learning_Update lorsqu'ils reçoivent une trame Flush_Tables provenant d'une passerelle active. Ce paramètre est encodé par la passerelle active dans sa trame Flush_Tables. Les trames Learning_Update |

| Variable de la passerelle | Description |
|---|---|
| | provenant des appareils DLR accélèrent l'apprentissage de la nouvelle topologie de réseau par l'ensemble des commutateurs non DLR situés en dehors du réseau DLR après le basculement sur la passerelle active. La valeur par défaut est TRUE. |
| Active Listen Timeout | Durée pendant laquelle un appareil passerelle doit attendre à l'état ACTIVE_LISTEN_STATE en mode écoute des trames Advertise à priorité plus élevée provenant d'un autre appareil passerelle. La valeur est égale à la durée Advertise Timeout actuelle. |

#### 10.8.4.3 Démarrage

Un appareil passerelle doit démarrer (à la mise sous tension ou lorsque la fonction de passerelle redondante est activée) à l'état IDLE_STATE tandis que la transmission de trafic est bloquée entre les ports DLR et les ports de liaison montante. Si la fonction de passerelle redondante est activée, il doit passer à l'état ACTIVE_LISTEN_STATE avec la transmission de trafic bloquée entre les ports DLR et les ports de liaison montante. Si la fonction de passerelle redondante est désactivée, il doit permettre la transmission de trafic entre ses ports DLR et les ports de liaison montante.

Immédiatement après le passage à l'état ACTIVE_LISTEN_STATE et une fois tous les intervalles Advertise Interval ensuite, la passerelle doit transmettre les trames Advertise par l'intermédiaire de ses ports DLR (voir Figure 37). Elle doit également écouter les trames Advertise transmises par un autre appareil passerelle actif sur ses ports DLR. Si une trame Advertise est reçue à partir d'un autre appareil passerelle actif présentant une priorité plus élevée ou en cas d'interconnexion, si l'adresse MAC source de l'autre passerelle a un numéro supérieur à sa propre adresse MAC, elle doit passer à l'état BACKUP_NORMAL_STATE avec la transmission de trafic bloquée entre ses ports DLR et les ports de liaison montante, et doit mettre fin à la transmission des trames Advertise.




**Légende**

| Anglais | Français |
|---|---|
| Uplink Port | Port de liaison montante |
| DLR Active Gateway | Passerelle active DLR |
| DLR Active Supervisor | Superviseur actif DLR |
| DLR Backup Gateway N | Passerelle de sauvegarde DLR N |
| Advertise | Advertise |
| DLR Device 1 | Appareil DLR 1 |
| DLR Backup Supervisor N | Superviseur de sauvegarde DLR N |
| DLR Device N | Appareil DLR N |

**Figure 37 – Trame Advertise**

#### 10.8.4.4 Fonctionnement normal

A l'état ACTIVE_LISTEN_STATE, si aucune trame Advertise n'est reçue ou qu'aucune trame Advertise avec une priorité plus élevée (ou un numéro d'adresse MAC source supérieur en cas d'interconnexion) n'est reçue pendant la durée Active Listen Timeout, l'appareil doit passer à l'état ACTIVE_NORMAL_STATE et doit devenir l'appareil passerelle actif avec la transmission de trafic activée entre ses ports DLR et les ports de liaison montante. Immédiatement après le passage à l'état ACTIVE_NORMAL_STATE et une fois tous les intervalles Advertise Interval ensuite, la passerelle doit transmettre les trames Advertise par l'intermédiaire de ses ports DLR.

Immédiatement après le passage à l'état ACTIVE_NORMAL_STATE, un appareil passerelle actif doit transmettre une trame Flush_Tables à l'ensemble des nœuds DLR et doit purger ses propres tableaux de filtrage d'apprentissage d'adresses monodiffusion et multidiffusion. A la réception d'une trame Flush_tables, un nœud DLR doit purger ses tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion. Si le champ Learning Update Enable de la trame Flush_Tables est défini sur TRUE, un nœud DLR doit également transmettre une trame Learning_Update pour accélérer l'apprentissage de la nouvelle topologie de réseau par l'ensemble des commutateurs non DLR situés en dehors du réseau DLR.

A l'état ACTIVE_NORMAL_STATE, si une trame Advertise est reçue à partir d'un autre appareil passerelle actif présentant une priorité plus élevée (ou un numéro d'adresse MAC source supérieur en cas d'interconnexion), le récepteur doit passer à l'état BACKUP_NORMAL_STATE avec la transmission de trafic bloquée entre ses ports DLR et les ports de liaison montante, et doit mettre fin à la transmission des trames Advertise.

#### 10.8.4.5 Détection de défaut de liaison montante

A l'état ACTIVE_LISTEN_STATE ou BACKUP_NORMAL_STATE, si les connexions physiques sont perdues sur l'ensemble de ses ports de liaison montante ou si un défaut de connexion de plus haut niveau est détecté via un protocole facultatif mis en œuvre sur les ports de liaison montante, un appareil passerelle doit passer à l'état FAULT_STATE avec la transmission de trafic bloquée entre ses ports DLR et les ports de liaison montante. Il ne doit pas transmettre de trames Advertise à l'état FAULT_STATE.

A l'état ACTIVE_NORMAL_STATE, si les connexions physiques sont perdues sur l'ensemble de ses ports de liaison montante ou si un défaut de connexion de plus haut niveau est détecté via un protocole facultatif mis en œuvre sur les ports de liaison montante, un appareil passerelle actif doit transmettre une trame Advertise avec l'état FAULT_STATE encodé dans la trame et doit passer à l'état FAULT_STATE avec la transmission de trafic bloquée entre ses ports DLR et les ports de liaison montante. Il ne doit pas transmettre de trames Advertise à l'état FAULT_STATE ensuite.

#### 10.8.4.6 Changement de passerelle

A l'état BACKUP_NORMAL_STATE, si une trame Advertise est reçue en provenance d'une passerelle active avec l'état FAULT_STATE encodé ou si aucune trame Advertise n'est reçue en provenance d'une passerelle active pendant la durée Advertise Timeout, un appareil passerelle de sauvegarde doit passer à l'état ACTIVE_LISTEN_STATE avec la transmission de trafic bloquée entre ses ports DLR et les ports de liaison montante. Il doit alors se comporter comme décrit en 10.8.4.3 pour l'état ACTIVE_LISTEN_STATE.

#### 10.8.4.7 Restauration d'une liaison montante

Lorsqu'une connexion de liaison montante en défaut est restaurée et que la passerelle active présente une priorité plus élevée que la sienne (ou un numéro d'adresse MAC supérieur en cas d'interconnexion), un appareil passerelle doit passer de l'état FAULT_STATE à BACKUP_NORMAL_STATE et doit bloquer la transmission de trafic entre ses ports DLR et les ports de liaison montante.

Lorsqu'une connexion de liaison montante en défaut est restaurée et que la passerelle active présente une priorité moins élevée que la sienne (ou un numéro d'adresse MAC inférieur à sa propre adresse MAC en cas d'interconnexion) ou qu'aucune passerelle active ne transmet de trames Advertise, un appareil passerelle doit passer de l'état FAULT_STATE à ACTIVE_LISTEN_STATE et doit bloquer la transmission de trafic entre ses ports DLR et les ports de liaison montante. Il doit alors se comporter comme décrit en 10.8.4.3 pour l'état ACTIVE_LISTEN_STATE.

#### 10.8.4.8 Défaut de réseau partiel

Un défaut de réseau partiel peut se produire, générant une perte de trafic dans une seule direction. Dans une telle situation, un appareil passerelle de sauvegarde peut diagnostiquer à tort que l'appareil passerelle actif est perdu. Pour empêcher plusieurs appareils passerelle de permettre la transmission de trafic entre leurs ports DLR et les ports de liaison montante, un appareil passerelle actif doit passer à l'état FAULT_STATE et doit bloquer la transmission de trafic entre ses ports DLR et les ports de liaison montante s'il reçoit une trame Advertise à l'état ACTIVE_NORMAL_STATE pour l'état de passerelle d'un autre appareil passerelle actif présentant une priorité moins élevée (ou un numéro d'adresse MAC inférieur en cas d'interconnexion). Il doit indiquer la condition par l'intermédiaire de l'attribut Redundant Gateway Status dans l'objet DLR. Il doit définir les attributs Active Gateway Address et Active Gateway Precedence sur ceux de l'autre passerelle active présentant une priorité inférieure. L'utilisateur doit explicitement supprimer cette condition par l'intermédiaire de l'objet DLR en utilisant le service Clear_Gateway_Partial_Fault.

### 10.9 Messages DLR

#### 10.9.1 Généralités

Les Paragraphes 10.10.2 à 10.9.9 spécifient les messages de protocole DLR. Il convient de remarquer plusieurs éléments d'importance:

a) Les messages DLR, à l'exception de Learning_Update, sont envoyés à l'aide du format de trame IEEE 802.1Q. Les messages doivent être transmis avec la priorité la plus élevée (7), qui doit être conservée, étant donné que les trames DLR sont transférées par le réseau local.

b) Pour différencier les types de données de trame DLR de ceux des données CIP, la convention UINTx est utilisée pour indiquer un nombre x de bits, valeur entière non signée dans les tables inclue en 10.8. Les types de données multi-octets dans les trames DLR doivent être codés, l'octet de poids fort l'étant en premier (format gros-boutiste).

c) Le Tableau 136 spécifie les adresses MAC de destination utilisées pour les messages DLR.

**Tableau 136 – Adresses MAC pour les messages DLR**

| Adresse MAC | Utilisation |
|---|---|
| 01-21-6C-00-00-01 | Beacon |
| 01-21-6C-00-00-02 | Demande Neighbor_Check, réponse Neighbor_Check, Sign_On |
| 01-21-6C-00-00-03 | Announce, Locate_Fault, Flush_Tables |
| 01-21-6C-00-00-04 | Advertise |
| 01-21-6C-00-00-05 | Learning_Update |
| Adresse MAC du superviseur d'anneau actif | Link_Status/Neighbor_Status |

#### 10.9.2 En-tête de trame commune

Les messages DLR, à l'exception de Learning_Update, doivent être envoyés en utilisant le format de trame IEEE 802.1Q (voir Tableau 137).

**Tableau 137 – Format d'en-tête de trame commune IEEE 802.1Q**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 0 | Destination MAC Address | UINT8[6] | |
| 6 | Source MAC Address | UINT8[6] | |
| 12 | IEEE 802.1Q Tag Type | UINT16 | = 0x8100 |
| 14 | IEEE 802.1Q Tag control | UINT16 | = 0xE000 + VLAN_ID (valeur par défaut = 0) |
| 16 | Ring EtherType | UINT16 | = 0x80E1 |
| 18 | Ring Sub-type | UINT8 | = 0x02 |
| 19 | Ring Protocol Version | UINT8 | = 0x01 |

Le Tableau 138 spécifie les champs communs utilisés dans les données utiles du message DLR.

**Tableau 138 – Champs de charge utile des messages DLR**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | Identifie le type de message DLR (voir Tableau 139) |
| 21 | Source Port | UINT8 | Identifie le port de l'appareil à l'origine de ce message:<br>0x00 – Port 1 ou Port 2<br>0x01 – Port 1<br>0x02 – Port 2 |
| 22 | Source IP Address | UINT32 | Adresse IP source du nœud émetteur. Si aucune adresse n'est configurée, la valeur doit être égale à 0 |
| 26 | Sequence Id | UINT32 | ID séquence du message DLR. Lors de l'envoi d'un message, un nœud doit incrémenter l'ID séquence de 1. Lors de la réponse à un message (réponse Neighbor_Check et Sign_On), un nœud doit utiliser l'ID séquence provenant du message de demande d'origine. |
| 30 | Dépendant du type de message DLR | | Voir Tableau 140 à Tableau 148 |
| … | FCS | UINT32 | ISO/CEI 8802-3 FCS |

**Tableau 139 – Types de trames DLR**

| Valeur | Nom | Format de trame |
|---|---|---|
| 0x01 | Beacon | Common Frame Format<br>Voir Tableau 137 et Tableau 138 |
| 0x02 | Neighbor_Check_Request | |
| 0x03 | Neighbor_Check_Response | |
| 0x04 | Link_Status / Neighbor_Status | |
| 0x05 | Locate_Fault | |
| 0x06 | Announce | |
| 0x07 | Sign_On | |
| 0x08 | Advertise | |
| 0x09 | Flush_Tables | |
| 0x0A | Learning_Update | Voir 10.9.12 |

### 10.9.3 Trame Beacon

La trame Beacon est envoyée par le superviseur d'anneau actif afin de déterminer l'état de santé de la topologie en anneau. Le Tableau 140 spécifie le format de la trame Beacon.

**Tableau 140 – Format de la trame Beacon**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x01 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 26 | Sequence Id | UINT32 | |
| 30 | Ring State | UINT8 | Voir Tableau 141 |
| 31 | Supervisor Precedence | UINT8 | A partir de l'objet DLR |
| 32 | Beacon Interval | UINT32 | A partir de l'objet DLR |
| 36 | Beacon Timeout | UINT32 | En microsecondes (µs) |
| 40 | Réservé | UINT8[20] | L'émetteur doit être défini sur zéro, le récepteur doit ignorer |
| 60 | FCS | UINT32 | |

Le Tableau 141 spécifie les valeurs d'état de l'anneau.

**Tableau 141 – Valeurs d'état de l'anneau**

| Ring State | Valeur |
|---|---|
| RING_NORMAL_STATE | 0x01 |
| RING_FAULT_STATE | 0x02 |

### 10.9.4 Demande Neighbor_Check

La demande Neighbor_Check est envoyée par un nœud d'anneau suite à la réception d'une trame Locate_Fault, et par un superviseur actif si une perte Beacon a été détectée. Le format de la demande Neighbor_Check est spécifié dans le Tableau 142.

**Tableau 142 – Format de la demande Neighbor_Check**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x02 |
| 21 | Source Port | UINT8 | = 0x1 ou 0x2 |
| 22 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 26 | Sequence Id | UINT32 | |
| 30 | Réservé | UINT8[30] | L'émetteur doit être défini sur zéro, le récepteur doit ignorer |
| 60 | FCS | UINT32 | |

### 10.9.5 Réponse Neighbor_Check

La trame de réponse Neighbor_Check est envoyée en réponse à la demande Neighbor_Check. Le format de la réponse Neighbor_Check est spécifié dans le Tableau 143.

**Tableau 143 – Format de la réponse Neighbor_Check**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x03 |
| 21 | Source Port | UINT8 | = 0x01 ou 0x02 |
| 22 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 26 | Sequence Id | UINT32 | = ID séquence de la demande Neighbor_Check |
| 30 | Request Source Port | UINT8 | = Port source de la demande Neighbor_Check |
| 31 | Réservé | UINT8[29] | L'émetteur doit être défini sur zéro, le récepteur doit ignorer |
| 60 | FCS | UINT32 | |

### 10.9.6 Link_Status/Neighbor_Status

Une trame Link_Status est envoyée lorsqu'un nœud a détecté un défaut de liaison ou en réponse à une trame Locate_Fault lorsqu'un des ports n'est pas actif. Une trame Neighbor_Status est envoyée lorsqu'un processus Neighbor_Check a été temporisé. Ces trames partagent le même format différencié par la balise Link_Status/Neighbor_Status, comme spécifié dans le Tableau 144.

**Tableau 144 – Format de la trame Link_Status/Neighbor_Status**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x04 |
| 21 | Source Port | UINT8 | = 0 |
| 22 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 26 | Sequence Id | UINT32 | |
| 30 | Etat Liaison/Voisin | UINT8 | Interprété comme un matriciel (voir Tableau 145) |
| 31 | Réservé | UINT8[29] | L'émetteur doit être défini sur zéro, le récepteur doit ignorer |
| 60 | FCS | UINT32 | |

Le Tableau 145 spécifie les valeurs de Link_Status/Neighbor_Status.

**Tableau 145 – Valeurs de Link_Status/Neighbor_Status**

| Bit(s) | Nom | Définition |
|---|---|---|
| 0<br>(poids le plus faible) | Port 1 actif | Défini sur 1 si le port 1 du nœud est actif |
| 1 | Port 2 actif | Défini sur 1 si le port 2 du nœud est actif |
| 2-6 | Réservé | L'émetteur doit être défini sur zéro, le récepteur doit ignorer |
| 7 | Balise Link_Status/Neighbor_Status | Défini sur 0 si la trame est Link_Status<br>Défini sur 1 si la trame est Neighbor_Status |

### 10.9.7 Locate_Fault

La trame Locate_Fault est envoyée par le superviseur d'anneau actif suite à la perte des trames Beacon et au service Verify_Fault_Location envoyé à l'objet DLR. Le format de la trame Locate_Fault est spécifié dans le Tableau 146.

**Tableau 146 – Format de la trame Locate_Fault**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x05 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 26 | Sequence Id | UINT32 | |
| 30 | Réservé | UINT8[30] | L'émetteur doit être défini sur zéro, le récepteur doit ignorer |
| 60 | FCS | UINT32 | |

### 10.9.8 Announce

La trame Announce est envoyée par le superviseur d'anneau actif pour indiquer un changement d'état de l'anneau et notifier la topologie en anneau aux nœuds Announce. Le format de la trame Announce est spécifié dans le Tableau 147.

**Tableau 147 – Format de la trame Announce**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x06 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 26 | Sequence Id | UINT32 | |
| 30 | Ring State | UINT8 | Comme dans le trame Beacon |
| 31 | Réservé | UINT8[29] | L'émetteur doit être défini sur zéro, le récepteur doit ignorer |
| 60 | FCS | UINT32 | |

### 10.9.9 Sign_On

A l'origine, la trame Sign_On est envoyée par le superviseur d'anneau actif, afin d'identifier tous les nœuds d'anneau participants. Le nœud participant de l'anneau retient la trame Sign_On et la transfère uniquement vers le CPU hôte. Le CPU hôte incrémente le nombre de nœuds de la liste, ajoute ses propres adresses à la liste et transmet la trame Sign_On uniquement par l'autre port que le port destinataire. Le format de la trame Sign_On est spécifié dans le Tableau 148.

**Tableau 148 – Format de la trame Sign_On**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x07 |
| 21 | Source Port | UINT8 | = 0x01 ou 0x02 |
| 22 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 26 | Sequence Id | UINT32 | |
| 30 | Number of Nodes in List | UINT16 | |
| 32 | Node 1 MAC Address | UINT8[6] | Le nœud 1 est toujours le superviseur actif |
| 38 | Node 1 IP Address | UINT32 | = 0x0, s'il n'existe aucune adresse IP |
| 42 | Node 2 MAC Address | UINT8[6] | |
| 48 | Node 2 IP Address | UINT32 | = 0x0, s'il n'existe aucune adresse IP |
| ... | Réservé | UINT8[...] | L'émetteur doit être défini sur zéro, le récepteur doit ignorer |
| N | FCS | UINT32 | N doit être rempli à au moins 60, le cas échéant |

### 10.9.10 Advertise

La trame Advertise est envoyée par la passerelle active pour signaler la présence d'une passerelle active et notifier les défauts de passerelles. Le format de la trame Advertise est spécifié dans le Tableau 149.

**Tableau 149 – Format de la trame Advertise**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x08 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 26 | Sequence ID | UINT32 | |
| 30 | Gateway State | UINT8 | Voir Tableau 150 |
| 31 | Gateway Precedence | UINT8 | A partir de l'objet DLR |
| 32 | Advertise Interval | UINT32 | A partir de l'objet DLR |
| 36 | Advertise Timeout | UINT32 | A partir de l'objet DLR |
| 40 | Learning Update Enable | UINT8 | A partir de l'objet DLR |
| 41 | Réservé | UINT8[19] | |
| 60 | FCS | UINT32 | |

**Tableau 150 – Valeurs d'état de la passerelle**

| Etat de la passerelle | Valeur |
|---|---|
| ACTIVE_LISTEN_STATE | 0x01 |
| ACTIVE_NORMAL_STATE | 0x02 |
| FAULT_STATE | 0x03 |

### 10.9.11 Flush_Tables

La trame Flush_Tables est envoyée par la nouvelle passerelle active pour indiquer qu'il convient que les nœuds DLR purgent leurs tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion. Le format de la trame Flush_Tables est spécifié dans le Tableau 151.

**Tableau 151 – Format de la trame Flush_Tables**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 20 | Frame Type | UINT8 | = 0x09 |
| 21 | Source Port | UINT8 | = 0x0 |
| 22 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 26 | Sequence Id | UINT32 | |
| 30 | Learning Update Enable | UINT8 | A partir de l'objet DLR |
| 31 | Réservé | UINT8[29] | |
| 60 | FCS | UINT32 | |

### 10.9.12 Learning_Update

La trame Learning_Update est envoyée par l'ensemble des nœuds DLR après réception d'une trame Flush_Tables en provenance de la passerelle active et si le champ Learning Update Enable est défini sur TRUE pour accélérer l'apprentissage de la nouvelle topologie de réseau par l'ensemble des commutateurs non DLR situés en dehors du réseau DLR. La trame Learning_Update est une trame IEEE 802.3 non balisée comme indiqué ci-dessous, ce qui fait que les commutateurs non DLR du VLAN situés hors du réseau DLR ne la supprimeront pas. Le format de la trame Learning_Update est spécifié dans le Tableau 152.

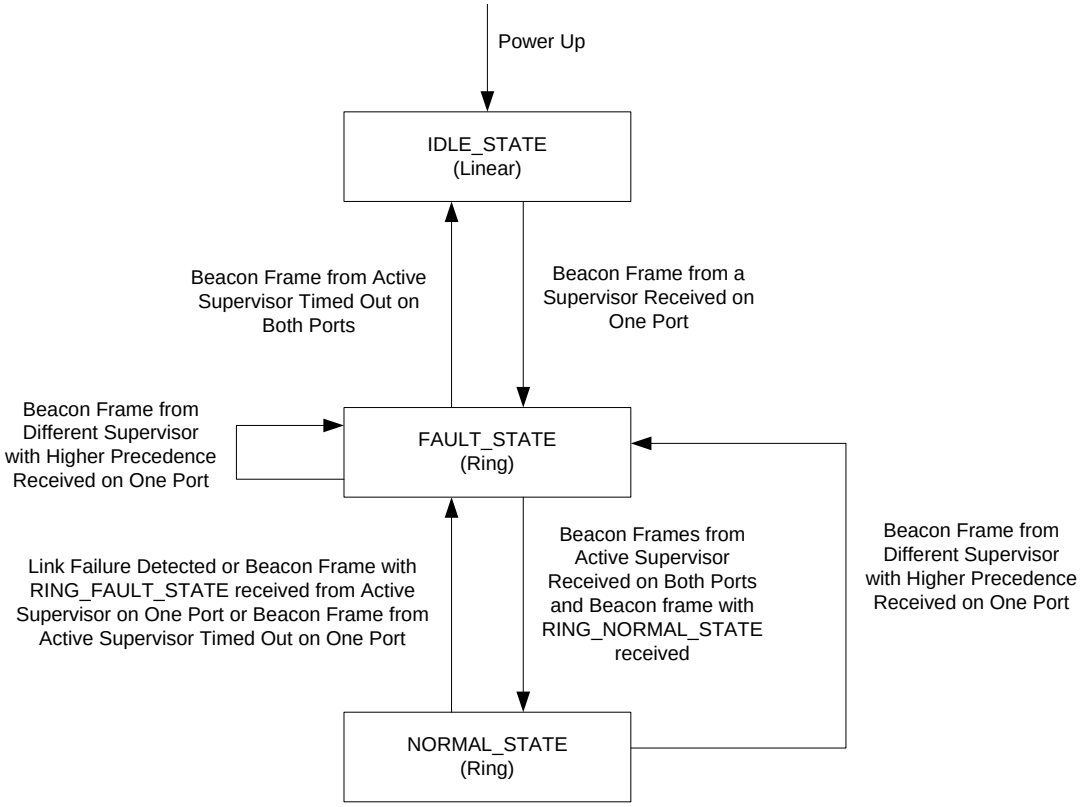
**Tableau 152 – Format de la trame Learning_Update**

| Octet | Champ | Type | Remarques |
|---|---|---|---|
| 0 | Destination MAC Address | UINT8[6] | |
| 6 | Source MAC Address | UINT8[6] | |
| 12 | Ring EtherType | UINT16 | = 0x80E1 |
| 14 | Ring Sub-type | UINT8 | = 0x02 |
| 15 | Ring Protocol Version | UINT8 | = 0x01 |
| 16 | Frame Type | UINT8 | = 0x0A |
| 17 | Source Port | UINT8 | = 0x0 |
| 18 | Source IP Address | UINT32 | = 0x0, si aucune adresse IP n'est associée à la source |
| 22 | Sequence ID | UINT32 | |
| 26 | Réservé | UINT8[34] | |
| 60 | FCS | UINT32 | |

## 10.10 Diagrammes d'états et matrices SEA (State-Event-Action)

### 10.10.1 Nœud d'anneau Beacon

La Figure 38 illustre le diagramme de transition d'états de la trame Beacon en fonction du nœud d'anneau non superviseur.




**Légende**

| Anglais | Français |
|---|---|
| Power up | Mise sous tension |
| (linear) | (linéaire) |
| Beacon frame from active supervisor timed out on both ports | Trame Beacon provenant du superviseur actif temporisée sur deux ports |
| Beacon frame from a supervisor received on one port | Trame Beacon provenant d'un superviseur reçue sur un port |
| Beacon frame from different supervisor with higher precedence received on one port | Trame Beacon provenant de superviseur différent à priorité plus élevée reçue sur un port |
| (ring) | (anneau) |
| Beacon frames from active supervisor received on both ports | Trames Beacon provenant du superviseur actif temporisées sur deux ports |
| Link failure detected or Beacon Frame with RING_FAULT_STATE received from Active supervisor on one port or Beacon Frame from active supervisor timed out on one port | Interruption de liaison détectée ou trame Beacon avec RING_FAULT_STATE reçue du superviseur actif sur un port ou trame Beacon provenant du superviseur actif temporisée sur un port |

**Figure 38 – Diagramme d'états de la trame Beacon en fonction du nœud d'anneau non superviseur.**

Le Tableau 153 spécifie les valeurs de paramètre de la trame Beacon pour un nœud d'anneau non superviseur.

**Tableau 153 – Valeurs de paramètre de la trame Beacon en fonction du nœud d'anneau non superviseur**

| Paramètre | Valeur |
|---|---|
| Active supervisor precedence | Obtenue à partir de la trame Beacon |
| Active supervisor MAC address | Obtenue à partir de la trame Beacon |
| Ring protocol VLAN ID | Obtenue à partir de la trame Beacon |
| Beacon timeout duration | Obtenue à partir de la trame Beacon |
| Neighbor check timeout duration | 100 ms |
| Maximum retry limit for Neighbor_Check request frame | 3 (nombre total de tentatives) |

Les exigences suivantes s'appliquent à la matrice SEA.

- LastBcnRcvPort est une variable sous la forme d'une chaîne de bits permettant de suivre le ou les ports sur lesquels la dernière trame Beacon a été reçue, dont les définitions de bit sont spécifiées dans le Tableau 154.
- L'adresse MAC 11-22-33-44-55-66 doit être codée 0x112233445566 pour comparaison numérique en cas de priorité du superviseur.
- Les nœuds doivent configurer statiquement l'adresse MAC à diffusion individuelle du superviseur d'anneau actif dans une table d'apprentissage MAC à diffusion individuelle à transférer uniquement par les deux ports d'anneau.

**Tableau 154 – Définitions de bit LastBcnRcvPort**

| Bit | Définition |
|---|---|
| 0 | Défini si une trame Beacon du superviseur actif a été reçue sur le port 1 |
| 1 | Défini si une trame Beacon du superviseur actif a été reçue sur le port 2 |
| 2-N | Non utilisé, défini sur 0 |

Le Tableau 155 spécifie la matrice SEA de la trame Beacon en fonction du nœud d'anneau non superviseur.

**Tableau 155 – Matrice SEA de la trame Beacon en fonction du nœud d'anneau non superviseur**

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 1 | Aucun | Mise sous tension | Initialiser et passer à l'état IDLE_STATE |
| 2 | IDLE_STATE | Trame Beacon provenant d'un superviseur d'anneau reçue sur le port 1 | Sauvegarder en tant que priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 1.<br>Passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 3 | IDLE_STATE | Trame Beacon provenant d'un superviseur d'anneau reçue sur le port 2 | Sauvegarder en tant que priorité de superviseur actif, adresse MAC, VLAN ID et délai d'attente Beacon.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 2.<br>Passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 4 | IDLE_STATE | Liaison perdue sur le port 1 | Arrêter le transfert des trames sur le port 1 |
| 5 | IDLE_STATE | Liaison perdue sur le port 2 | Arrêter le transfert des trames sur le port 2 |
| 6 | IDLE_STATE | Liaison rétablie sur le port 1 | Démarrer le transfert des trames sur le port 1 |
| 7 | IDLE_STATE | Liaison rétablie sur le port 2 | Démarrer le transfert des trames sur le port 2 |
| 8 | IDLE_STATE | Trame reçue sur le port 1 ou 2 | Ne pas transférer la trame sur le réseau et la supprimer (s'il est possible de le faire) |
| 9 | IDLE_STATE | Demande Locate_Fault ou Neighbor_Check; ou réponse Neighbor_Check; ou trame Sign_On reçue sur le port 1 ou 2 | Ignorer |
| 10 | FAULT_STATE | Trame Beacon provenant du superviseur d'anneau actif reçue sur le port 1 et LastBcnRcvPort défini sur 1 | Redémarrer le temporisateur de délai d'attente Beacon du port 1 |
| 11 | FAULT_STATE | Trame Beacon provenant du superviseur d'anneau actif reçue sur le port 2 et LastBcnRcvPort défini sur 2 | Redémarrer le temporisateur de délai d'attente Beacon du port 2 |
| 12 | FAULT_STATE | Expiration du temporisateur de délai d'attente Beacon pour le port 1 et LastBcnRcvPort défini sur 1 | Arrêter les temporisateurs de délai d'attente Neighbor_Check sur les deux ports, passer à l'état IDLE_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 13 | FAULT_STATE | Expiration du temporisateur de délai d'attente Beacon pour le port 2 et LastBcnRcvPort défini sur 2 | Arrêter les temporisateurs de délai d'attente Neighbor_Check sur les deux ports, passer à l'état IDLE_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 14 | FAULT_STATE | Trame Beacon provenant du superviseur d'anneau actif reçue sur le port 2 et LastBcnRcvPort défini sur 1 ou 3 | Attribuer la valeur 3 à LastBcnRcvPort<br>Démarrer/redémarrer le temporisateur de délai d'attente Beacon du port 2<br>Si l'état de l'anneau de la trame Beacon est différent de RING_NORMAL_STATE, revenir.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports.<br>Passer à l'état NORMAL_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 15 | FAULT_STATE | Trame Beacon provenant du superviseur d'anneau actif reçue sur le port 1 et LastBcnRcvPort défini sur 2 ou 3 | Attribuer la valeur 3 à LastBcnRcvPort<br>Démarrer/redémarrer le temporisateur de délai d'attente Beacon du port 1<br>Si l'état de l'anneau de la trame Beacon est différent de RING_NORMAL_STATE, revenir.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports.<br>Passer à l'état NORMAL_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 16 | FAULT_STATE | Une trame Beacon provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 1 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Beacon pour les deux ports.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 1.<br>Rester à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 17 | FAULT_STATE | Une trame Beacon provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 2 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Beacon pour les deux ports.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 2.<br>Rester à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 18 | FAULT_STATE | Liaison perdue sur le port 1 et LastBcnRcvPort défini sur 1 | Arrêter le temporisateur de délai d'attente Beacon du port 1 et les temporisateurs de délai d'attente Neighbor_Check des deux ports<br>Passer à l'état IDLE_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 19 | FAULT_STATE | Liaison perdue sur le port 1 et LastBcnRcvPort défini sur 2 | Envoyer la trame Link_Status au superviseur d'anneau actif |
| 20 | FAULT_STATE | Liaison perdue sur le port 2 et LastBcnRcvPort défini sur 2 | Arrêter le temporisateur de délai d'attente Beacon du port 2 et les temporisateurs de délai d'attente Neighbor_Check des deux ports<br>Passer à l'état IDLE_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 21 | FAULT_STATE | Liaison perdue sur le port 2 et LastBcnRcvPort défini sur 1 | Envoyer la trame Link_Status au superviseur d'anneau actif. |
| 22 | FAULT_STATE | Liaison rétablie sur le port 1 | Démarrer le transfert des trames sur le port 1 |
| 23 | FAULT_STATE | Liaison rétablie sur le port 2 | Démarrer le transfert des trames sur le port 2 |
| 24 | FAULT_STATE | Trame Locate_Fault reçue du superviseur d'anneau actif sur le port 1 ou 2 | Si la liaison n'est pas active sur le port 1 ou 2, envoyer la trame Link_Status au superviseur d'anneau actif.<br>Sinon, annuler Neighbor_Status pour le port 1 et 2, envoyer la demande Neighbor_Check pour le port 1 et 2 et démarrer le temporisateur de délai d'attente Neighbor_Check pour le port 1 et 2. |
| 25 | FAULT_STATE | Trame de la demande Neighbor_Check reçue sur le port 1 | Envoyer la trame de réponse Neighbor_Check sur le port 1. |
| 26 | FAULT_STATE | Trame de la demande Neighbor_Check reçue sur le port 2 | Envoyer la trame de réponse Neighbor_Check sur le port 2. |
| 27 | FAULT_STATE | Trame de la réponse Neighbor_Check reçue sur le port 1 | Arrêter le temporisateur de délai d'attente Neighbor_Check sur le port 1 et enregistrer Neighbor_Status sur le port 1 |
| 28 | FAULT_STATE | Trame de la réponse Neighbor_Check reçue sur le port 2 | Arrêter le temporisateur de délai d'attente Neighbor_Check sur le port 2 et enregistrer Neighbor_Status sur le port 2 |
| 29 | FAULT_STATE | Délai d'attente du temporisateur Neighbor_Check sur le port 1 | Si le nombre de nouvelles tentatives n'a pas dépassé la limite maximale, envoyer la demande Neighbor_Check sur le port 1 et démarrer le temporisateur de délai d'attente Neighbor_Check sur le port 1.<br>Sinon, envoyer la trame Neighbor_Status au superviseur d'anneau actif |
| 30 | FAULT_STATE | Délai d'attente du temporisateur Neighbor_Check sur le port 2 | Si le nombre de nouvelles tentatives n'a pas dépassé la limite maximale, envoyer la demande Neighbor_Check sur le port 2 et démarrer le temporisateur de délai d'attente Neighbor_Check sur le port 2.<br>Sinon, envoyer la trame Neighbor_Status au superviseur d'anneau actif |
| 31 | FAULT_STATE | Trame Sign_On reçue sur le port 1 ou 2 | Ignorer |
| 32 | NORMAL_STATE | Liaison perdue sur le port 1 | Envoyer la trame Link_Status au superviseur d'anneau actif.<br>Arrêter le temporisateur de délai d'attente Beacon pour le port 1.<br>Attribuer la valeur 2 à LastBcnRcvPort, passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 33 | NORMAL_STATE | Liaison perdue sur le port 2 | Envoyer la trame Link_Status au superviseur d'anneau actif.<br>Arrêter le temporisateur de délai d'attente Beacon pour le port 2.<br>Attribuer la valeur 1 à LastBcnRcvPort, passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 34 | NORMAL_STATE | Trame Beacon provenant d'un superviseur d'anneau actif à l'état RING_FAULT_STATE reçue sur le port 1 | Si l'état de l'anneau de la trame Beacon précédente reçue sur le port 1 n'était pas RING_NORMAL_STATE, revenir en arrière.<br>Sinon, arrêter le temporisateur de délai d'attente Beacon pour le port 2.<br>Attribuer la valeur 1 à LastBcnRcvPort, passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 35 | NORMAL_STATE | Trame Beacon provenant d'un superviseur d'anneau actif à l'état RING_FAULT_STATE reçue sur le port 2 | Si l'état de l'anneau de la trame Beacon précédente reçue sur le port 2 n'était pas RING_NORMAL_STATE, revenir en arrière.<br>Sinon, arrêter le temporisateur de délai d'attente Beacon pour le port 1.<br>Attribuer la valeur 2 à LastBcnRcvPort, passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 36 | NORMAL_STATE | Temporisateur de délai d'attente Beacon expiré pour le port 1 | Attribuer la valeur 2 à LastBcnRcvPort, passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 37 | NORMAL_STATE | Temporisateur de délai d'attente Beacon expiré pour le port 2 | Attribuer la valeur 1 à LastBcnRcvPort, passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 38 | NORMAL_STATE | Trame Beacon provenant d'un superviseur d'anneau actif reçue sur le port 1 | Redémarrer le temporisateur de délai d'attente Beacon du port 1 |
| 39 | NORMAL_STATE | Trame Beacon provenant d'un superviseur d'anneau actif reçue sur le port 2 | Redémarrer le temporisateur de délai d'attente Beacon du port 2 |
| 40 | NORMAL_STATE | Une trame Beacon provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 1 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Beacon pour les deux ports.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 1.<br>Passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

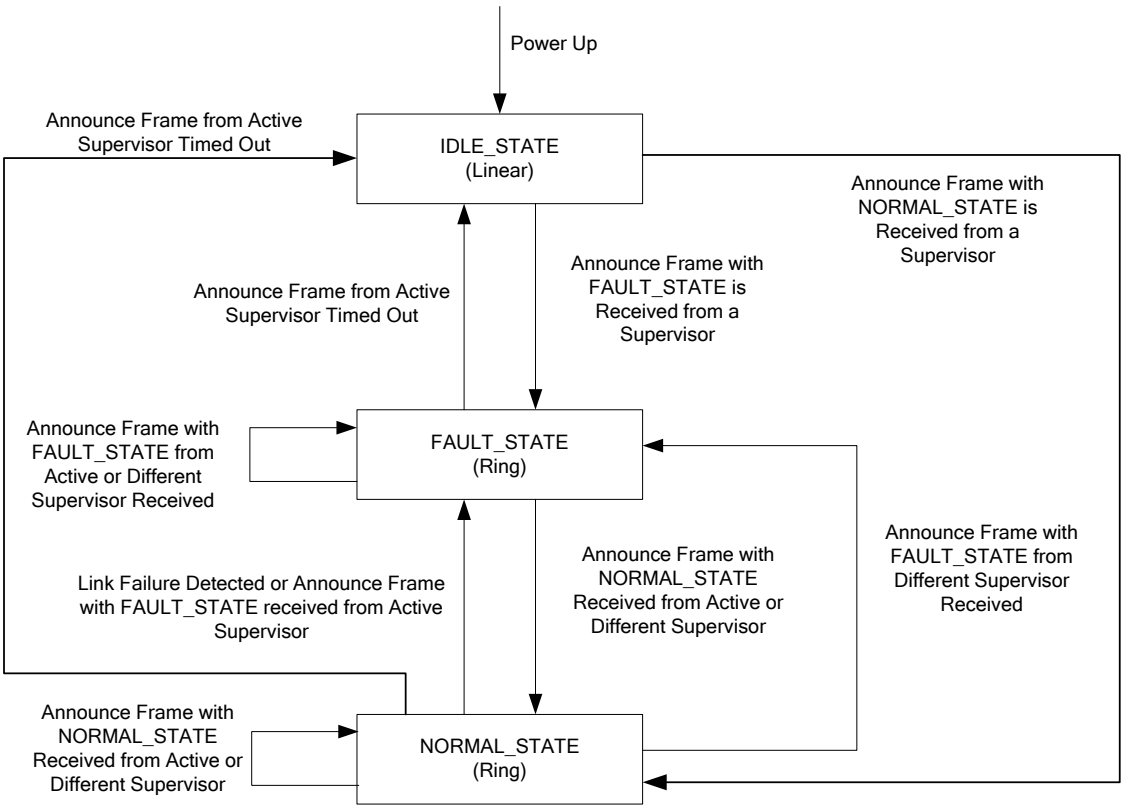
| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 41 | NORMAL_STATE | Une trame Beacon provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 2 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Beacon pour les deux ports.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 2.<br>Passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 42 | NORMAL_STATE | Trame Locate_Fault reçue du superviseur d'anneau actif sur le port 1 ou 2 | Ignorer |
| 43 | NORMAL_STATE | Trame de demande Neighbor_Check ou de réponse Neighbor_Check reçue sur le port 1 ou 2 | Ignorer |
| 44 | NORMAL_STATE | Trame Sign_On reçue sur le port 1 | Ajouter les données de nœud à la liste des participants et envoyer la trame par le port 2 |
| 45 | NORMAL_STATE | Trame Sign_On reçue sur le port 2 | Ajouter les données de nœud à la liste des participants et envoyer la trame par le port 1 |
| 46 | IDLE_STATE | Trame Flush_Tables reçue | Ignorer |
| 47 | NORMAL_STATE | Trame Flush_Tables reçue | Purger les tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion.<br>Si le champ Learning Update Enable de la trame Flush_Tables est défini sur TRUE, envoyer la trame Learning_Update. |
| 48 | FAULT_STATE0 | Trame Flush_Tables reçue | Purger les tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion.<br>Si le champ Learning Update Enable de la trame Flush_Tables est défini sur TRUE, envoyer la trame Learning_Update. |

[a] L'attribut Network Topology doit être mis à jour aux événements 1, 2, 3, 12, 13, 18 et 20 avec les valeurs appropriées.

[b] L'attribut Network Status doit être mis à jour aux événements 1, 2, 3, 8, 12, 13, 14, 15, 18, 20, 32, 33, 34, 35, 36, 37, 40 et 41 avec les valeurs appropriées.

### 10.10.2 Nœud d'anneau Announce

La Figure 39 illustre le diagramme de transition d'états de la trame Announce en fonction du nœud d'anneau non superviseur.




**Légende**

| Anglais | Français |
|---|---|
| Power up | Mise sous tension |
| (linear) | (linéaire) |
| Announce frame from active supervisor timed out | Trame Announce provenant du superviseur actif temporisée |
| Announce frame with NORMAL_STATE is received from a supervisor | Trame Announce avec état NORMAL_STATE est reçue d'un superviseur |
| Announce frame with FAULT_STATE is received from a supervisor | Trame Announce avec état FAULT_STATE est reçue d'un superviseur |
| (ring) | (anneau) |
| Announce frame with FAULT_STATE from Active or Different supervisor received | Trame Announce avec état FAULT_STATE provenant d'un superviseur actif ou différent reçue |
| Link failure detected or Announce frame with FAULT_STATE received from active supervisor | Interruption de liaison détectée ou trame Announce avec état FAULT_STATE reçue du superviseur actif |
| Announce frame with NORMAL_STATE received from active or different supervisor | Trame Announce avec état NORMAL_STATE reçue d'un superviseur actif ou différent |
| Announce frame with FAULT_STATE from different supervisor received | Trame Announce avec état FAULT_STATE provenant d'un superviseur différent reçue |

**Figure 39 – Diagramme de transition d'états de la trame Announce en fonction du nœud d'anneau non superviseur.**

Le Tableau 156 spécifie les valeurs de paramètre de la trame Announce en fonction d'un nœud d'anneau non superviseur.

**Tableau 156 – Valeurs de paramètre de la trame Announce en fonction du nœud d'anneau non superviseur**

| Paramètre | Valeur |
|---|---|
| Active supervisor MAC address | Obtenue à partir de la trame Announce |
| Ring protocol VLAN ID | Obtenue à partir de la trame Announce |
| Announce timeout duration | 2,5 s |
| Neighbor check timeout duration | 100 ms |
| Maximum retry limit for Neighbor_Check request frame | 3 (nombre total de tentatives) |

Les exigences suivantes s'appliquent à la matrice SEA.

- Le nombre de séquences doit être vérifié pour toutes les trames Announce provenant du superviseur d'anneau actif à l'aide d'un entier arithmétique 32 bits non signé modulaire afin de traiter les retournements de numéro de séquence. Seules les trames Announce dont le nombre de séquences est supérieur à la dernière trame Announce doivent être traitées. Les autres doivent être écartées en silence.
- Les nœuds doivent configurer statiquement l'adresse MAC à diffusion individuelle du superviseur d'anneau actif dans une table d'apprentissage MAC à diffusion individuelle à transférer uniquement par les deux ports d'anneau.
- Les nœuds doivent configurer statiquement l'adresse MAC à diffusion individuelle des trames Beacon dans la table d'apprentissage MAC à transférer uniquement par les deux ports d'anneau, et pas par le CPU.

Le Tableau 157 spécifie la matrice SEA de la trame Announce en fonction du nœud d'anneau non superviseur.

**Tableau 157 – Matrice SEA de la trame Announce en fonction du nœud d'anneau non superviseur**

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 1 | Aucun | Mise sous tension | Initialiser et passer à l'état IDLE_STATE |
| 2 | IDLE_STATE | Trame Announce provenant du superviseur d'anneau actif reçue sur le port 1 ou 2 | Sauvegarder le superviseur actif, l'adresse MAC, le VLAN ID et le nombre de séquences.<br>Démarrer le temporisateur de délai d'attente Announce.<br>Passer à l'état d'anneau dans la trame Announce et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 3 | IDLE_STATE | Liaison perdue sur le port 1 | Arrêter le transfert des trames sur le port 1 |
| 4 | IDLE_STATE | Liaison perdue sur le port 2 | Arrêter le transfert des trames sur le port 2 |
| 5 | IDLE_STATE | Liaison rétablie sur le port 1 | Démarrer le transfert des trames sur le port 1 |
| 6 | IDLE_STATE | Liaison rétablie sur le port 2 | Démarrer le transfert des trames sur le port 2 |
| 7 | IDLE_STATE | Trame reçue sur le port 1 ou 2 | Ne pas transférer la trame sur le réseau et la supprimer (s'il est possible de le faire) |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 8 | IDLE_STATE | Demande Locate_Fault ou Neighbor_Check; ou réponse Neighbor_Check; ou trame Sign_On reçue sur le port 1 ou 2 | Ignorer |
| 9 | FAULT_STATE | Trame Announce provenant du superviseur d'anneau actif reçue sur le port 1 ou 2 avec un nombre de séquences supérieur à la dernière trame Announce. | Sauvegarder le nombre de séquences.<br>Mettre à jour le VLAN ID (s'il est différent)<br>Redémarrer le temporisateur de délai d'attente Announce.<br>Si l'état de l'anneau de la trame Announce est RING_NORMAL_STATE et que les liaisons sur les deux ports sont actives, arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports, passer à l'état NORMAL_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle.<br>Sinon, rester à l'état FAULT_STATE |
| 10 | FAULT_STATE | Le temporisateur de délai d'attente Announce a expiré | Arrêter les temporisateurs de délai d'attente Neighbor_Check sur les deux ports, passer à l'état IDLE_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 11 | FAULT_STATE | Une trame Announce provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 1 ou 2 | Redémarrer le temporisateur de délai d'attente Announce et arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports.<br>Sauvegarder le nouveau superviseur actif, l'adresse MAC, le VLAN ID et le nombre de séquences.<br>Si l'état de l'anneau de la trame Announce est RING_NORMAL_STATE et que les liaisons sur les deux ports sont actives, passer à l'état NORMAL_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle.<br>Sinon, rester à l'état FAULT_STATE |
| 12 | FAULT_STATE | Liaison perdue sur le port 1 | SI la liaison sur le port 2 est active, envoyer la trame Link_Status au superviseur d'anneau actif.<br>Sinon, arrêter le temporisateur de délai d'attente Announce, arrêter les temporisateurs de délai d'attente Neighbor_Check sur les deux ports, passer à l'état IDLE_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 13 | FAULT_STATE | Liaison perdue sur le port 2 | SI la liaison sur le port 1 est active, envoyer la trame Link_Status au superviseur d'anneau actif.<br>Sinon, arrêter le temporisateur de délai d'attente Announce, arrêter les temporisateurs de délai d'attente Neighbor_Check sur les deux ports, passer à l'état IDLE_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 14 | FAULT_STATE | Liaison rétablie sur le port 1 | Démarrer le transfert des trames sur le port 1 |
| 15 | FAULT_STATE | Liaison rétablie sur le port 2 | Démarrer le transfert des trames sur le port 2 |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 16 | FAULT_STATE | Trame Locate_Fault reçue du superviseur d'anneau actif | Si les liaisons sont actives sur les deux ports, annuler l'état Neighbor_Status pour les deux ports, envoyer la trame de demande Neighbor_Check_request sur les deux ports et démarrer le temporisateur de délai d'attente Neighbor_Check pour les deux ports.<br>Sinon, envoyer la trame Link_Status au superviseur d'anneau actif |
| 17 | FAULT_STATE | Trame de la demande Neighbor_Check reçue sur le port 1 | Envoyer la trame de réponse Neighbor_Check sur le port 1. |
| 18 | FAULT_STATE | Trame de la demande Neighbor_Check reçue sur le port 2 | Envoyer la trame de réponse Neighbor_Check sur le port 2. |
| 19 | FAULT_STATE | Trame de la réponse Neighbor_Check reçue sur le port 1 | Arrêter le temporisateur de délai d'attente Neighbor_Check sur le port 1 et enregistrer Neighbor_Status sur le port 1 |
| 20 | FAULT_STATE | Trame de la réponse Neighbor_Check reçue sur le port 2 | Arrêter le temporisateur de délai d'attente Neighbor_Check sur le port 2 et enregistrer Neighbor_Status sur le port 2 |
| 21 | FAULT_STATE | Délai d'attente du temporisateur Neighbor_Check sur le port 1 | Si le nombre de nouvelles tentatives n'a pas dépassé la limite maximale, envoyer la demande Neighbor_Check sur le port 1 et démarrer le temporisateur de délai d'attente Neighbor_Check sur le port 1.<br>Sinon, envoyer la trame Neighbor_Status au superviseur d'anneau actif |
| 22 | FAULT_STATE | Délai d'attente du temporisateur Neighbor_Check sur le port 2 | Si le nombre de nouvelles tentatives n'a pas dépassé la limite maximale, envoyer la demande Neighbor_Check sur le port 2 et démarrer le temporisateur de délai d'attente Neighbor_Check sur le port 2.<br>Sinon, envoyer la trame Neighbor_Status au superviseur d'anneau actif |
| 23 | FAULT_STATE | Trame Sign_On reçue sur le port 1 ou 2 | Ignorer |
| 24 | NORMAL_STATE | Liaison perdue sur le port 1 | Envoyer la trame Link_Status au superviseur d'anneau actif.<br>Passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 25 | NORMAL_STATE | Liaison perdue sur le port 2 | Envoyer la trame Link_Status au superviseur d'anneau actif.<br>Passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 26 | NORMAL_STATE | Trame Announce provenant du superviseur d'anneau actif reçue sur le port 1 ou 2 avec un nombre de séquences supérieur à la dernière trame Announce. | Sauvegarder le nombre de séquences.<br>Mettre à jour le VLAN ID (s'il est différent)<br>Redémarrer le temporisateur de délai d'attente Announce.<br>Si l'état de l'anneau de la trame Announce est RING_FAULT_STATE, passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle<br>Sinon, rester à l'état NORMAL_STATE |
| 27 | NORMAL_STATE | Le temporisateur de délai d'attente Announce a expiré | Passer à l'état IDLE_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

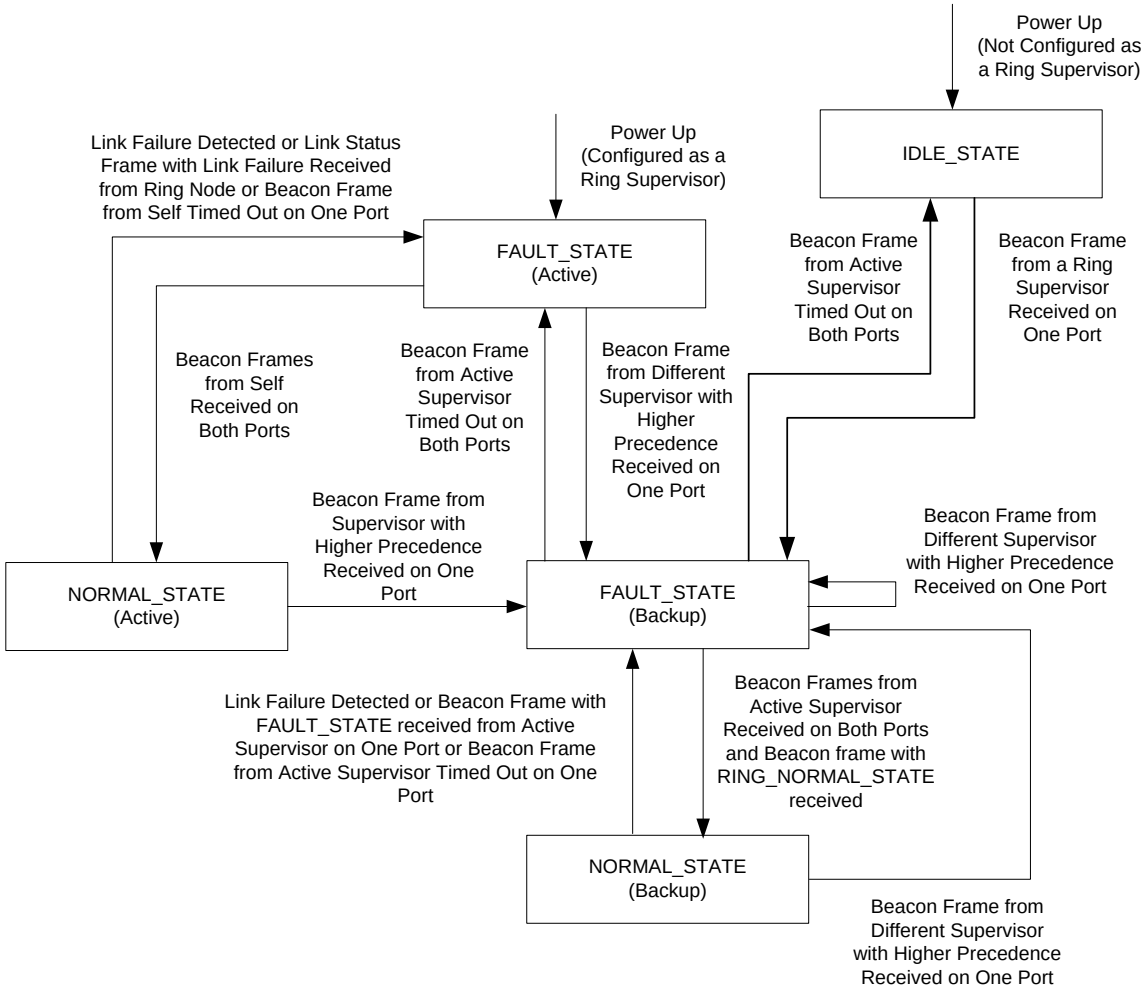
| Evénement n° | Etat actuel | Evénement | Action(s)[a,b] |
|---|---|---|---|
| 28 | NORMAL_STATE | Une trame Announce provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 1 ou 2 | Redémarrer le temporisateur de délai d'attente Announce.<br>Sauvegarder le nouveau superviseur actif, l'adresse MAC, le VLAN ID et le nombre de séquences.<br>Si l'état de l'anneau de la trame Announce est RING_FAULT_STATE, passer à l'état FAULT_STATE et purger la table d'apprentissage d'adresse MAC à diffusion individuelle.<br>Sinon, rester à l'état NORMAL_STATE |
| 29 | NORMAL_STATE | Trame Locate_Fault reçue du superviseur d'anneau actif | Ignorer |
| 30 | NORMAL_STATE | Trame de demande Neighbor_Check ou de réponse Neighbor_Check reçue sur le port 1 ou 2 | Ignorer |
| 31 | NORMAL_STATE | Trame Sign_On reçue sur le port 1 | Ajouter les données de nœud à la liste des participants et envoyer la trame par le port 2 |
| 32 | NORMAL_STATE | Trame Sign_On reçue sur le port 2 | Ajouter les données de nœud à la liste des participants et envoyer la trame par le port 1 |
| 33 | IDLE_STATE | Trame Flush_Tables reçue | Ignorer |
| 34 | NORMAL_STATE | Trame Flush_Tables reçue | Purger les tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion.<br>Si le champ Learning Update Enable de la trame Flush_Tables est défini sur TRUE, envoyer la trame Learning_Update. |
| 35 | FAULT_STATE | Trame Flush_Tables reçue | Purger les tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion.<br>Si le champ Learning Update Enable de la trame Flush_Tables est défini sur TRUE, envoyer la trame Learning_Update. |

[a] L'attribut Network Topology doit être mis à jour aux événements 1, 2, 10, 12, 13 et 27 avec les valeurs appropriées.

[b] L'attribut Network Status doit être mis à jour aux événements 1, 2, 7, 9, 10, 11, 12, 13, 24, 25, 26, 27 et 28 avec la valeur appropriée.

### 10.10.3 Superviseur d'anneau

La Figure 40 illustre le diagramme de transition d'états du superviseur d'anneau.




**Légende**

| Anglais | Français |
|---|---|
| Power up | Mise sous tension |
| (not configured as a Ring supervisor) | (non configuré comme superviseur d'anneau) |
| Link failure detected or link status frame with link failure received from ring node or Beacon frame from self-timed out on one port | Interruption de liaison détectée ou trame Link Status avec interruption de liaison reçue du nœud d'anneau ou trame Beacon provenant de Self temporisée sur un port |
| (configured as a ring supervisor) | (configuré comme un superviseur d'anneau) |
| (active) | (actif) |
| Beacon frame from active supervisor timed out on both ports | Trame Beacon provenant du superviseur actif temporisée sur deux ports |
| Beacon frame from a ring supervisor received on one port | Trame Beacon provenant du superviseur d'anneau reçue sur un port |
| Beacon frame from self-received on both ports | Trame Beacon provenant du self reçue sur deux ports |
| Beacon frame from different supervisor with higher precedence received on one port | Trame Beacon provenant d'un superviseur différent à priorité plus élevée reçue sur un port |
| Beacon frame from supervisor with higher precedence received on one port | Trame Beacon provenant du superviseur à priorité plus élevée reçue sur un port |
| (backup) | (sauvegarde) |
| Link failure detected or Beacon frame with FAULT_STATE received from Active supervisor on one port or Beacon frame from Active timed | Interruption de liaison détectée ou trame Beacon avec FAULT_STATE reçue du superviseur actif sur un port ou trame Beacon provenant du |

| Anglais | Français |
|---|---|
| out on one port | superviseur actif temporisée sur un port |
| Beacon frame from active supervisor received on both ports | Trame Beacon provenant du superviseur actif reçue sur les deux ports |

**Figure 40 – Diagramme de transition d'états du superviseur d'anneau**

Les valeurs de paramètre du nœud de superviseur d'anneau sont spécifiées dans le Tableau 158.

**Tableau 158 – Valeurs de paramètre du nœud de superviseur d'anneau**

| Paramètre | Valeur |
|---|---|
| Active Supervisor Precedence | Valeur par défaut 0 ou valeur configurée |
| Active supervisor MAC address | Conforme à la fabrication |
| Ring protocol VLAN ID | Valeur par défaut 0 ou valeur configurée |
| Beacon interval duration | Valeur par défaut 400 µs ou valeur configurée |
| Beacon timeout duration | Valeur par défaut 1 960 µs ou valeur configurée |
| Announce suppression duration | 2 fois Beacon Timeout |
| Announce interval duration | 1 s |
| Active supervisor precedence (Sauvegarde) | Obtenue à partir de la trame Beacon |
| Active supervisor MAC address (Sauvegarde) | Obtenue à partir de la trame Beacon |
| Ring protocol VLAN ID (Sauvegarde) | Obtenue à partir de la trame Beacon |
| Beacon timeout duration (Sauvegarde) | Obtenue à partir de la trame Beacon |
| Neighbor check timeout duration | 100 ms |
| Maximum retry limit for Neighbor_Check request frame | 3 (nombre total de tentatives) |
| Switchover quiet period duration during ring supervisor switchover | Durée de 1 Beacon Timeout du dernier superviseur d'anneau actif |
| Sign on timeout duration | 60 s |

Les exigences suivantes s'appliquent à la matrice SEA:

- LastBcnRcvPort est une variable sous la forme d'une chaîne de bits permettant de suivre le ou les ports sur lesquels la dernière trame Beacon a été reçue, dont les définitions de bit sont spécifiées dans le Tableau 159.
- L'adresse MAC 11-22-33-44-55-66 doit être codée 0x112233445566 pour comparaison numérique en cas de priorité du superviseur.
- Les nœuds qui ne sont pas configurés en tant que superviseur d'anneau ou ne faisant pas office de superviseur d'anneau de sauvegarde doivent configurer statiquement l'adresse MAC à diffusion individuelle ou le superviseur d'anneau actif en cours dans la table d'apprentissage MAC à diffusion individuelle pour être transféré uniquement par les deux ports d'anneau.
- Si les attributs d'un superviseur actif (beacon interval duration, beacon timeout duration, precedence et VLAN ID, par exemple) sont modifiés, le superviseur actif doit se comporter de manière particulière avant d'appliquer les nouvelles valeurs. Si le superviseur actif était à l'état NORMAL_STATE (Actif), il doit arrêter tous les temporisateurs, laisser le port 2 à l'état bloqué et arrêter l'envoi de trames Beacon et Announce pendant 2 durées (en cours) de délai d'attente Beacon. Si le superviseur actif était à l'état FAULT_STATE (Actif), il doit arrêter tous les temporisateurs, laisser le port 2 à l'état de transfert et arrêter l'envoi de trames Beacon et Announce pendant 2 durées (en cours) de délai d'attente Beacon. A l'issue de la période d'attente, il doit démarrer à partir de l'événement 1 (Tableau 160). Voir également 10.5.2.7 (Modification des paramètres d'anneau).

- Si un superviseur de sauvegarde ne peut pas prendre en charge l'intervalle Beacon et/ou le délai d'attente Beacon en cours, le superviseur de sauvegarde doit indiquer la condition par l'intermédiaire de l'objet DLR et ne doit pas reprendre en tant que superviseur actif en cas de reconfiguration de l'anneau.
- Si l'attribut de priorité d'un superviseur de sauvegarde est reconfiguré en ligne avec une valeur plus élevée que le superviseur d'anneau actif en cours, il doit arrêter tous les temporisateurs et démarrer de l'événement 1 (Tableau 160) immédiatement. Voir également 10.5.2.7 (Modification des paramètres d'anneau).
- Si un nœud non superviseur est activé en tant que superviseur d'anneau en ligne et que la valeur de priorité configurée est supérieure au superviseur d'anneau actif en cours, il doit arrêter tous les temporisateurs et démarrer de l'événement 1 (Tableau 160) immédiatement. Pour toutes les autres modifications, il doit simplement se comporter comme un superviseur de sauvegarde à l'état en cours.
- Le superviseur d'anneau actif ne doit pas transférer des trames multidiffusion avec l'adresse de destination 01:80:C2:00:00:00 (BPDU) d'un port d'anneau à un autre, quel que soit l'état de l'anneau. Toutefois, si le superviseur d'anneau actif contient des ports qui ne sont pas des ports d'anneau, il doit transférer lesdites trames multidiffusion entre ces ports et uniquement entre un port d'anneau et les ports de ce type.

**Tableau 159 – Définitions de bit LastBcnRcvPort**

| Bit | Définition |
|---|---|
| 0 | Défini si une trame Beacon du superviseur actif a été reçue sur le port 1 |
| 1 | Défini si une trame Beacon du superviseur actif a été reçue sur le port 2 |
| 2-N | Non utilisé, défini sur 0 |

Le Tableau 160 spécifie la matrice SEA du nœud de superviseur d'anneau.

**Tableau 160 – Matrice SEA du nœud de superviseur d'anneau**

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 1 | Aucun | Mise sous tension | Initialiser.<br>S'il n'est pas configuré comme un superviseur d'anneau, passer à l'état IDLE_STATE et revenir en arrière.<br>Sinon, attribuer la valeur 0 à LastBcnRcvPort.<br>Activer le transfert de toutes les trames sur les deux ports.<br>Passer à l'état FAULT_STATE (Actif).<br>Envoyer la trame Beacon par les deux ports et démarrer le temporisateur d'intervalle Beacon.<br>Démarrer le temporisateur d'intervalle Announce avec Announce Suppression Duration |
| 2 | IDLE_STATE | Trame Beacon provenant d'un superviseur d'anneau reçue sur le port 1 | Sauvegarder en tant que priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 1.<br>Passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 3 | IDLE_STATE | Trame Beacon provenant d'un superviseur d'anneau reçue sur le port 2 | Sauvegarder en tant que priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 2.<br>Passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 4 | IDLE_STATE | Liaison perdue sur le port 1 | Arrêter le transfert des trames sur le port 1 |
| 5 | IDLE_STATE | Liaison perdue sur le port 2 | Arrêter le transfert des trames sur le port 2 |
| 6 | IDLE_STATE | Liaison rétablie sur le port 1 | Démarrer le transfert des trames sur le port 1 |
| 7 | IDLE_STATE | Liaison rétablie sur le port 2 | Démarrer le transfert des trames sur le port 2 |
| 8 | IDLE_STATE | Trame reçue sur le port 1 ou 2 | Ne pas transférer la trame sur le réseau et la supprimer. |
| 9 | IDLE_STATE | Demande Locate_Fault ou Neighbor_Check; ou réponse Neighbor_Check; ou trame Sign_on ou Link_Status ou Neighbor_Status reçue sur le port 1 ou le port 2; ou demande de service Verify_Fault_Location reçue | Ignorer |
| 10 | FAULT_STATE<br>(Actif) | Temporisateur d'intervalle Beacon expiré | Envoyer la trame Beacon par les deux ports et démarrer le temporisateur d'intervalle Beacon |
| 11 | FAULT_STATE<br>(Actif) | Trame Beacon provenant de self reçue sur le port 1 | Attribuer la valeur (LastBcnRcvPort \| 0x1) à LastBcnRcvPort<br>Redémarrer le temporisateur de délai d'attente Beacon pour le port 1.<br>Si la valeur 3 n'a pas été attribuée à LastBcnRcvPort, revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports.<br>Bloquer tout le trafic sur le port 2, à l'exception des trames de protocole d'anneau particulières.<br>Passer à l'état NORMAL_STATE (Actif) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle<br>Envoyer la trame Announce sur le port 1 et redémarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration.<br>Envoyer la trame Beacon sur les deux ports et démarrer le paramètre Announce interval duration.<br>Envoyer la trame Sign_On sur le port 1 et démarrer le temporisateur de délai d'attente Sign_On |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 12 | FAULT_STATE<br>(Actif) | Trame Beacon provenant de self reçue sur le port 2 | Attribuer la valeur (LastBcnRcvPort \| 0x2) à LastBcnRcvPort<br>Redémarrer le temporisateur de délai d'attente Beacon pour le port 2.<br>Si la valeur 3 n'a pas été attribuée à LastBcnRcvPort, revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports.<br>Bloquer tout le trafic sur le port 2, à l'exception des trames de protocole d'anneau particulières.<br>Passer à l'état NORMAL_STATE (Actif) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle<br>Envoyer la trame Announce sur le port 1 et redémarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration.<br>Envoyer la trame Beacon sur les deux ports et démarrer le paramètre Announce interval duration.<br>Envoyer la trame Sign_On sur le port 1 et démarrer le temporisateur de délai d'attente Sign_On |
| 13 | FAULT_STATE<br>(Actif) | Trame Beacon provenant d'une adresse MAC de superviseur différente du self reçue sur le port 1 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) que celle du self, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter le temporisateur d'intervalle Beacon, les temporisateurs de délai d'attente Beacon des deux ports, les temporisateurs de délai d'attente Neighbor_Check des deux ports et le temporisateur d'intervalle Announce.<br>Sauvegarder la nouvelle priorité de superviseur actif, l'adresse MAC, le VLAN ID et le délai d'attente Beacon.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 1.<br>Passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 14 | FAULT_STATE<br>(Actif) | Trame Beacon provenant d'une adresse MAC de superviseur différente du self reçue sur le port 2 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) que celle du self, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter le temporisateur d'intervalle Beacon, les temporisateurs de délai d'attente Beacon des deux ports, les temporisateurs de délai d'attente Neighbor_Check des deux ports et le temporisateur d'intervalle Announce.<br>Sauvegarder la nouvelle priorité de superviseur actif, l'adresse MAC, le VLAN ID et le délai d'attente Beacon.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 2.<br>Passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 15 | FAULT_STATE (Actif) | Temporisateur de délai d'attente Beacon expiré pour le port 1 | Attribuer la valeur (LastBcnRcvPort & ~0x1) à LastBcnRcvPort |
| 16 | FAULT_STATE (Actif) | Temporisateur de délai d'attente Beacon expiré pour le port 2 | Attribuer la valeur (LastBcnRcvPort & ~0x2) à LastBcnRcvPort |
| 17 | FAULT_STATE (Actif) | Liaison perdue sur le port 1 | Sauvegarder self comme étant le dernier nœud actif du port 1.<br>Arrêter le transfert des trames sur le port 1 |
| 18 | FAULT_STATE (Actif) | Liaison perdue sur le port 2 | Sauvegarder self comme étant le dernier nœud actif du port 2.<br>Arrêter le transfert des trames sur le port 2 |
| 19 | FAULT_STATE (Actif) | Liaison rétablie sur le port 1 | Démarrer le transfert des trames sur le port 1 |
| 20 | FAULT_STATE (Actif) | Liaison rétablie sur le port 2 | Démarrer le transfert des trames sur le port 2 |
| 21 | FAULT_STATE (Actif) | Trame Link_Status reçue sur le port 1 | Si la trame Link_Status ne signale aucune défaillance de liaison, supprimer la trame et revenir en arrière.<br>Sinon, sauvegarder le nœud source en tant que dernier actif sur le port 1 |
| 22 | FAULT_STATE (Actif) | Trame Link_Status reçue sur le port 2 | Si la trame Link_Status ne signale aucune défaillance de liaison, supprimer la trame et revenir en arrière.<br>Sinon, sauvegarder le nœud source en tant que dernier actif sur le port 2 |
| 23 | FAULT_STATE (Actif) | Trame Neighbor_Status reçue sur le port 1 | Si la trame Neighbor_Status ne signale aucune interruption du nœud avoisinant, supprimer la trame et revenir en arrière.<br>Sinon, sauvegarder le nœud source en tant que dernier actif sur le port 1 |
| 24 | FAULT_STATE (Actif) | Trame Neighbor_Status reçue sur le port 2 | Si la trame Neighbor_Status ne signale aucune interruption du nœud avoisinant, supprimer la trame et revenir en arrière.<br>Sinon, sauvegarder le nœud source en tant que dernier actif sur le port 2 |
| 25 | FAULT_STATE (Actif) | Temporisateur d'intervalle Announce expiré | Envoyer la trame Announce par les deux ports et démarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration. |
| 26 | FAULT_STATE (Actif) | Demande de service Verify_Fault_Location reçue | Envoyer la trame Locate_Fault sur le port 1 et sur le port 2.<br>Si la liaison est active sur le port 1, annuler Neighbor_Status pour le port 1, envoyer une trame de demande Neighbor_Check sur le port 1 et démarrer le temporisateur de délai d'attente Neighbor_Check pour le port 1.<br>Sinon, sauvegarder self comme étant le dernier nœud actif du port 1.<br>Si la liaison est active sur le port 2, annuler Neighbor_Status pour le port 2, envoyer une trame de demande Neighbor_Check sur le port 2 et démarrer le temporisateur de délai d'attente Neighbor_Check pour le port 2.<br>Sinon, sauvegarder self comme étant le dernier nœud actif du port 2. |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 27 | FAULT_STATE (Actif) | Trame de la demande Neighbor_Check reçue sur le port 1 | Envoyer la trame de réponse Neighbor_Check sur le port 1. |
| 28 | FAULT_STATE (Actif) | Trame de la demande Neighbor_Check reçue sur le port 2 | Envoyer la trame de réponse Neighbor_Check sur le port 2. |
| 29 | FAULT_STATE (Actif) | Trame de la réponse Neighbor_Check reçue sur le port 1 | Arrêter le temporisateur de délai d'attente Neighbor_Check sur le port 1 et enregistrer Neighbor_Status sur le port 1 |
| 30 | FAULT_STATE (Actif) | Trame de la réponse Neighbor_Check reçue sur le port 2 | Arrêter le temporisateur de délai d'attente Neighbor_Check sur le port 2 et enregistrer Neighbor_Status sur le port 2 |
| 31 | FAULT_STATE (Actif) | Délai d'attente du temporisateur Neighbor_Check sur le port 1 | Si le nombre de nouvelles tentatives n'a pas dépassé la limite maximale, envoyer la demande Neighbor_Check sur le port 1 et démarrer le temporisateur de délai d'attente Neighbor_Check sur le port 1.<br>Sinon, sauvegarder self comme étant le dernier nœud actif du port 1 |
| 32 | FAULT_STATE (Actif) | Délai d'attente du temporisateur Neighbor_Check sur le port 2 | Si le nombre de nouvelles tentatives n'a pas dépassé la limite maximale, envoyer la demande Neighbor_Check sur le port 2 et démarrer le temporisateur de délai d'attente Neighbor_Check sur le port 2.<br>Sinon, sauvegarder self comme étant le dernier nœud actif du port 2. |
| 33 | FAULT_STATE (Actif) | Trame Sign_On reçue sur le port 1 ou 2 | Ignorer |
| 34 | NORMAL_STATE (Actif) | Temporisateur d'intervalle Beacon expiré | Envoyer la trame Beacon par les deux ports et démarrer le temporisateur d'intervalle Beacon |
| 35 | NORMAL_STATE (Actif) | Trame Beacon provenant de self reçue sur le port 1 | Redémarrer le temporisateur de délai d'attente Beacon pour le port 1 |
| 36 | NORMAL_STATE (Actif) | Trame Beacon provenant de self reçue sur le port 2 | Redémarrer le temporisateur de délai d'attente Beacon pour le port 2 |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 37 | NORMAL_STATE<br>(Actif) | Trame Beacon provenant d'une adresse MAC de superviseur différente du self reçue sur le port 1 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter le temporisateur d'intervalle Beacon, les temporisateurs de délai d'attente Beacon pour les deux ports, le temporisateur d'intervalle Announce et le temporisateur de délai d'attente Sign_On.<br>Activer le transfert de toutes les trames sur le port 2.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 1.<br>Passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 38 | NORMAL_STATE<br>(Actif) | Trame Beacon provenant d'une adresse MAC de superviseur différente du self reçue sur le port 2 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter le temporisateur d'intervalle Beacon, les temporisateurs de délai d'attente Beacon pour les deux ports, le temporisateur d'intervalle Announce et le temporisateur de délai d'attente Sign_On.<br>Activer le transfert de toutes les trames sur le port 2.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 2.<br>Passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 39 | NORMAL_STATE<br>(Actif) | Temporisateur de délai d'attente Beacon expiré pour le port 1 | Arrêter le temporisateur de délai d'attente Sign_On.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Activer le transfert de toutes les trames sur le port 2.<br>Passer à l'état FAULT_STATE (Actif) et purger la table d'adresse MAC à diffusion individuelle<br>Envoyer la trame Announce sur les deux ports et redémarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration.<br>Envoyer la trame Beacon et redémarrer le temporisateur d'intervalle Beacon.<br>Envoyer la trame Locate_Fault.<br>Annuler l'état Neighbor_Status pour le port 1 et 2, envoyer la demande Neighbor_Check pour le port 1 et 2 et démarrer le temporisateur de délai d'attente Neighbor_Check pour le port 1 et 2. |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 40 | NORMAL_STATE<br>(Actif) | Temporisateur de délai d'attente Beacon expiré pour le port 2 | Arrêter le temporisateur de délai d'attente Sign_On.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Activer le transfert de toutes les trames sur le port 2.<br>Passer à l'état FAULT_STATE (Actif) et purger la table d'adresse MAC à diffusion individuelle<br>Envoyer la trame Announce sur les deux ports et redémarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration.<br>Envoyer la trame Beacon et redémarrer le temporisateur d'intervalle Beacon.<br>Envoyer la trame Locate_Fault.<br>Annuler l'état Neighbor_Status pour le port 1 et 2, envoyer la demande Neighbor_Check pour le port 1 et 2 et démarrer le temporisateur de délai d'attente Neighbor_Check pour le port 1 et 2 |
| 41 | NORMAL_STATE<br>(Actif) | Liaison perdue sur le port 1 | Sauvegarder self comme étant le dernier nœud actif du port 1.<br>Arrêter le temporisateur de délai d'attente Sign_On.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Activer le transfert de toutes les trames sur le port 2.<br>Passer à l'état FAULT_STATE (Actif) et purger la table d'adresse MAC à diffusion individuelle<br>Envoyer la trame Announce sur les deux ports et redémarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration.<br>Envoyer la trame Beacon et redémarrer le temporisateur d'intervalle Beacon |
| 42 | NORMAL_STATE<br>(Actif) | Liaison perdue sur le port 2 | Sauvegarder self comme étant le dernier nœud actif du port 2.<br>Arrêter le temporisateur de délai d'attente Sign_On.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Activer le transfert de toutes les trames sur le port 2.<br>Passer à l'état FAULT_STATE (Actif) et purger la table d'adresse MAC à diffusion individuelle<br>Envoyer la trame Announce sur les deux ports et redémarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration.<br>Envoyer la trame Beacon et redémarrer le temporisateur d'intervalle Beacon |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 43 | NORMAL_STATE (Actif) | Trame Link_Status reçue sur le port 1 | Si la trame Link_Status ne signale aucune défaillance de liaison, supprimer la trame et revenir en arrière.<br>Sinon, sauvegarder le nœud source en tant que dernier nœud actif sur le port 1.<br>Arrêter le temporisateur de délai d'attente Sign_On.<br>Activer le transfert de toutes les trames sur le port 2.<br>Passer à l'état FAULT_STATE (Actif) et purger la table d'adresse MAC à diffusion individuelle<br>Envoyer la trame Announce sur les deux ports et redémarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration.<br>Envoyer la trame Beacon et redémarrer le temporisateur d'intervalle Beacon |
| 44 | NORMAL_STATE (Actif) | Trame Link_Status reçue sur le port 2 | Si la trame Link_Status ne signale aucune défaillance de liaison, supprimer la trame et revenir en arrière.<br>Sinon, sauvegarder le nœud source en tant que dernier nœud actif sur le port 2.<br>Arrêter le temporisateur de délai d'attente Sign_On.<br>Activer le transfert de toutes les trames sur le port 2.<br>Passer à l'état FAULT_STATE (Actif) et purger la table d'adresse MAC à diffusion individuelle<br>Envoyer la trame Announce sur les deux ports et redémarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration.<br>Envoyer la trame Beacon et redémarrer le temporisateur d'intervalle Beacon |
| 45 | NORMAL_STATE (Actif) | Trame Neighbor_Status reçue sur le port 1 ou 2 | Ignorer |
| 46 | NORMAL_STATE (Actif) | Temporisateur d'intervalle Announce expiré | Envoyer la trame Announce par le port 1 et démarrer le temporisateur d'intervalle Announce avec le paramètre Announce Interval Duration. |
| 47 | NORMAL_STATE (Actif) | Demande de service Verify_Fault_Location reçue | Ignorer |
| 48 | NORMAL_STATE (Actif) | Trame de demande Neighbor_Check ou de réponse Neighbor_Check reçue sur le port 1 ou 2 | Ignorer |
| 49 | NORMAL_STATE (Actif) | Trame Sign_On reçue sur le port 1 ou 2 | Si la première entrée de la liste des participants n'est pas self, supprimer la trame et revenir en arrière.<br>Sinon, sauvegarder le nombre de participants de l'anneau à partir de la trame, et arrêter le temporisateur de délai d'attente Sign_On |
| 50 | NORMAL_STATE (Actif) | Temporisateur de délai d'attente Sign_On expiré | Envoyer la trame Sign_On via le port 1 et démarrer le temporisateur de délai d'attente Sign_On |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 51 | FAULT_STATE<br>(Sauvegarde) | Trame Beacon provenant du superviseur d'anneau actif reçue sur le port 1 et LastBcnRcvPort défini sur 1 | Redémarrer le temporisateur de délai d'attente Beacon du port 1 |
| 52 | FAULT_STATE<br>(Sauvegarde) | Trame Beacon provenant du superviseur d'anneau actif reçue sur le port 2 et LastBcnRcvPort défini sur 2 | Redémarrer le temporisateur de délai d'attente Beacon du port 2 |
| 53 | FAULT_STATE<br>(Sauvegarde) | Expiration du temporisateur de délai d'attente Beacon pour le port 1 et LastBcnRcvPort défini sur 1 | S'il n'est pas configuré en tant que superviseur d'anneau, arrêter les temporisateurs de délai d'attente Neighbor_Check sur les deux ports, passer à l'état IDLE_STATE, purger la table d'apprentissage d'adresse MAC à diffusion individuelle, puis revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports.<br>Attendre la période de calme de permutation.<br>Attribuer la valeur 0 à LastBcnRcvPort.<br>Activer le transfert de toutes les trames sur les deux ports.<br>Passer à l'état FAULT_STATE (Actif).<br>Envoyer la trame Beacon par les deux ports et démarrer le temporisateur d'intervalle Beacon.<br>Démarrer le temporisateur d'intervalle Announce avec Announce Suppression Duration |
| 54 | FAULT_STATE<br>(Sauvegarde) | Expiration du temporisateur de délai d'attente Beacon pour le port 2 et LastBcnRcvPort défini sur 2 | S'il n'est pas configuré en tant que superviseur d'anneau, arrêter les temporisateurs de délai d'attente Neighbor_Check sur les deux ports, passer à l'état IDLE_STATE, purger la table d'apprentissage d'adresse MAC à diffusion individuelle, puis revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports.<br>Attendre la période de calme de permutation.<br>Attribuer la valeur 0 à LastBcnRcvPort.<br>Activer le transfert de toutes les trames sur les deux ports.<br>Passer à l'état FAULT_STATE (Actif).<br>Envoyer la trame Beacon par les deux ports et démarrer le temporisateur d'intervalle Beacon.<br>Démarrer le temporisateur d'intervalle Announce avec Announce Suppression Duration |
| 55 | FAULT_STATE<br>(Sauvegarde) | Trame Beacon provenant du superviseur d'anneau actif reçue sur le port 2 et LastBcnRcvPort défini sur 1 | Attribuer la valeur 3 à LastBcnRcvPort.<br>Démarrer/réinitialiser le temporisateur de délai d'attente Beacon du port 2<br>Si l'état de l'anneau de la trame Beacon n'est pas RING_NORMAL_STATE, revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports.<br>Passer à l'état NORMAL_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 56 | FAULT_STATE<br>(Sauvegarde) | Trame Beacon provenant du superviseur d'anneau actif reçue sur le port 1 et LastBcnRcvPort défini sur 2 | Attribuer la valeur 3 à LastBcnRcvPort.<br>Démarrer/réinitialiser le temporisateur de délai d'attente Beacon du port 1<br>Si l'état de l'anneau de la trame Beacon n'est pas RING_NORMAL_STATE, revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check pour les deux ports.<br>Passer à l'état NORMAL_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 57 | FAULT_STATE<br>(Sauvegarde) | Une trame Beacon provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 1 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Beacon pour les deux ports.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 1.<br>Rester à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 58 | FAULT_STATE<br>(Sauvegarde) | Une trame Beacon provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 2 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Beacon pour les deux ports.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 2.<br>Rester à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 59 | FAULT_STATE<br>(Sauvegarde) | Liaison perdue sur le port 1 et LastBcnRcvPort défini sur 1 | S'il n'est pas configuré en tant que superviseur d'anneau, arrêter les temporisateurs de délai d'attente Neighbor_Check sur les deux ports, arrêter les temporisateurs de délai d'attente Beacon, passer à l'état IDLE_STATE, purger la table d'apprentissage d'adresse MAC à diffusion individuelle, puis revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check des deux ports et les temporisateurs de délai d'attente Beacon des deux ports.<br>Attendre la période de calme de permutation.<br>attribuer la valeur 0 à LastBcnRcvPort.<br>Activer le transfert de toutes les trames sur les deux ports.<br>Passer à l'état FAULT_STATE (Actif).<br>Envoyer la trame Beacon par les deux ports et démarrer le temporisateur d'intervalle Beacon.<br>Démarrer le temporisateur d'intervalle Announce avec Announce Suppression Duration |
| 60 | FAULT_STATE<br>(Sauvegarde) | Liaison perdue sur le port 1 et LastBcnRcvPort défini sur 2 | Envoyer la trame Link Status au superviseur d'anneau actif |
| 61 | FAULT_STATE<br>(Sauvegarde) | Liaison perdue sur le port 2 et LastBcnRcvPort défini sur 2 | S'il n'est pas configuré en tant que superviseur d'anneau, arrêter les temporisateurs de délai d'attente Neighbor_Check sur les deux ports, arrêter les temporisateurs de délai d'attente Beacon, passer à l'état IDLE_STATE, purger la table d'apprentissage d'adresse MAC à diffusion individuelle, puis revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Neighbor_Check des deux ports et les temporisateurs de délai d'attente Beacon des deux ports.<br>Attendre la période de calme de permutation.<br>attribuer la valeur 0 à LastBcnRcvPort.<br>Activer le transfert de toutes les trames sur les deux ports.<br>Passer à l'état FAULT_STATE (Actif).<br>Envoyer la trame Beacon par les deux ports et démarrer le temporisateur d'intervalle Beacon.<br>Démarrer le temporisateur d'intervalle Announce avec Announce Suppression Duration |
| 62 | FAULT_STATE<br>(Sauvegarde) | Liaison perdue sur le port 2 et LastBcnRcvPort défini sur 1 | Envoyer la trame Link Status au superviseur d'anneau actif |
| 63 | FAULT_STATE<br>(Sauvegarde) | Liaison rétablie sur le port 1 | Démarrer le transfert des trames sur le port 1 |
| 64 | FAULT_STATE<br>(Sauvegarde) | Liaison rétablie sur le port 2 | Démarrer le transfert des trames sur le port 2 |
| 65 | FAULT_STATE<br>(Sauvegarde) | Trame Locate_Fault reçue du superviseur d'anneau actif sur le port 1 ou 2 | Si la liaison n'est pas active sur le port 1 ou 2, envoyer la trame Link_Status au superviseur d'anneau actif<br>Sinon, annuler Neighbor_Status pour le port 1 et 2, envoyer la demande Neighbor_Check pour le port 1 et 2 et démarrer le temporisateur de délai d'attente Neighbor_Check pour le port 1 et 2. |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 66 | FAULT_STATE (Sauvegarde) | Trame de la demande Neighbor_Check reçue sur le port 1 | Envoyer la trame de réponse Neighbor_Check sur le port 1. |
| 67 | FAULT_STATE (Sauvegarde) | Trame de la demande Neighbor_Check reçue sur le port 2 | Envoyer la trame de réponse Neighbor_Check sur le port 2. |
| 68 | FAULT_STATE (Sauvegarde) | Trame de la réponse Neighbor_Check reçue sur le port 1 | Arrêter le temporisateur de délai d'attente Neighbor_Check sur le port 1 et enregistrer Neighbor_Status sur le port 1 |
| 69 | FAULT_STATE (Sauvegarde) | Trame de la réponse Neighbor_Check reçue sur le port 2 | Arrêter le temporisateur de délai d'attente Neighbor_Check sur le port 2 et enregistrer Neighbor_Status sur le port 2 |
| 70 | FAULT_STATE (Sauvegarde) | Délai d'attente du temporisateur Neighbor_Check sur le port 1 | Si le nombre de nouvelles tentatives n'a pas dépassé la limite maximale, envoyer la demande Neighbor_Check sur le port 1 et démarrer le temporisateur de délai d'attente Neighbor_Check sur le port 1.<br>Sinon, envoyer la trame Neighbor_Status au superviseur d'anneau actif |
| 71 | FAULT_STATE (Sauvegarde) | Délai d'attente du temporisateur Neighbor_Check sur le port 2 | Si le nombre de nouvelles tentatives n'a pas dépassé la limite maximale, envoyer la demande Neighbor_Check sur le port 2 et démarrer le temporisateur de délai d'attente Neighbor_Check sur le port 2.<br>Sinon, envoyer la trame Neighbor_Status au superviseur d'anneau actif |
| 72 | FAULT_STATE (Sauvegarde) | Trame Sign_On, Link_Status ou Neighbor_Status reçue sur le port 1 ou 2 ou demande de service Verify_Fault_Location reçue | Ignorer |
| 73 | NORMAL_STATE (Sauvegarde) | Liaison perdue sur le port 1 | Envoyer la trame Link Status au superviseur d'anneau actif.<br>Arrêter le temporisateur de délai d'attente Beacon pour le port 1.<br>Attribuer la valeur 2 à LastBcnRcvPort, passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 74 | NORMAL_STATE (Sauvegarde) | Liaison perdue sur le port 2 | Envoyer la trame Link Status au superviseur d'anneau actif.<br>Arrêter le temporisateur de délai d'attente Beacon pour le port 2.<br>Attribuer la valeur 2 à LastBcnRcvPort, passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 75 | NORMAL_STATE (Sauvegarde) | Trame Beacon provenant d'un superviseur d'anneau actif à l'état RING_FAULT_STATE reçue sur le port 1 | Si l'état de l'anneau de la trame Beacon précédente reçue sur le port 1 n'était pas RING_NORMAL_STATE, revenir en arrière.<br>Sinon, arrêter le temporisateur de délai d'attente Beacon pour le port 2.<br>Attribuer la valeur 1 à LastBcnRcvPort, passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 76 | NORMAL_STATE (Sauvegarde) | Trame Beacon provenant d'un superviseur d'anneau actif à l'état RING_FAULT_STATE reçue sur le port 2 | Si l'état de l'anneau de la trame Beacon précédente reçue sur le port 2 n'était pas RING_NORMAL_STATE, revenir en arrière.<br>Sinon, arrêter le temporisateur de délai d'attente Beacon pour le port 1.<br>Attribuer la valeur 2 à LastBcnRcvPort, passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 77 | NORMAL_STATE (Sauvegarde) | Temporisateur de délai d'attente Beacon expiré pour le port 1 | Attribuer la valeur 2 à LastBcnRcvPort, passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 78 | NORMAL_STATE (Sauvegarde) | Temporisateur de délai d'attente Beacon expiré pour le port 2 | Attribuer la valeur 1 à LastBcnRcvPort, passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 79 | NORMAL_STATE (Sauvegarde) | Trame Beacon provenant d'un superviseur d'anneau actif reçue sur le port 1 | Redémarrer le temporisateur de délai d'attente Beacon du port 1 |
| 80 | NORMAL_STATE (Sauvegarde) | Trame Beacon provenant d'un superviseur d'anneau actif reçue sur le port 2 | Redémarrer le temporisateur de délai d'attente Beacon du port 2 |
| 81 | NORMAL_STATE (Sauvegarde) | Une trame Beacon provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 1 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Beacon pour les deux ports.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 1 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 1.<br>Passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |
| 82 | NORMAL_STATE (Sauvegarde) | Une trame Beacon provenant d'une adresse MAC de superviseur d'anneau différente de celle du superviseur d'anneau actif a été reçue sur le port 2 | Si la priorité du nouveau superviseur est plus basse (ou que le numéro d'adresse MAC est inférieur en cas d'attache) à celle du superviseur actif, supprimer la nouvelle trame Beacon et revenir en arrière.<br>Sinon, arrêter les temporisateurs de délai d'attente Beacon pour les deux ports.<br>Sauvegarder la nouvelle priorité de superviseur actif, adresse MAC, VLAN ID et durée de délai d'attente Beacon.<br>Attribuer la valeur 2 à LastBcnRcvPort.<br>Démarrer le temporisateur de délai d'attente Beacon pour le port 2.<br>Passer à l'état FAULT_STATE (Sauvegarde) et purger la table d'apprentissage d'adresse MAC à diffusion individuelle |

| Evénement n° | Etat actuel | Evénement | Action(s)[a,b,c,d,e,f,g,h,i,j] |
|---|---|---|---|
| 83 | NORMAL_STATE (Sauvegarde) | Trame Locate_Fault; ou trame de demande Neighbor_Check ou de réponse Neighbor_Check; ou trame Link_Status ou Neighbor_Status reçue sur le port 1 ou le port 2 ou demande de service Verify_Fault_Location reçue | Ignorer |
| 84 | NORMAL_STATE (Sauvegarde) | Trame Sign_On reçue sur le port 1 | Si la première entrée de la liste des participants est self, supprimer la trame et revenir en arrière.<br>Sinon, ajouter les données de nœud à la liste des participants et envoyer la trame sur le port 2 |
| 85 | NORMAL_STATE (Sauvegarde) | Trame Sign_On reçue sur le port 2 | Si la première entrée de la liste des participants est self, supprimer la trame et revenir en arrière.<br>Sinon, ajouter les données de nœud à la liste des participants et envoyer la trame sur le port 1 |
| 86 | IDLE_STATE | Trame Flush_Tables reçue | Ignorer |
| 87 | NORMAL_STATE (Active) | Trame Flush_Tables reçue | Purger les tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion.<br>Si le champ Learning Update Enable de la trame Flush_Tables est défini sur TRUE, envoyer la trame Learning_Update. |
| 88 | FAULT_STATE (Active) | Trame Flush_Tables reçue | Purger les tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion.<br>Si le champ Learning Update Enable de la trame Flush_Tables est défini sur TRUE, envoyer la trame Learning_Update. |
| 89 | NORMAL_STATE (Sauvegarde) | Trame Flush_Tables reçue | Purger les tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion.<br>Si le champ Learning Update Enable de la trame Flush_Tables est défini sur TRUE, envoyer la trame Learning_Update. |
| 90 | FAULT_STATE (Sauvegarde) | Trame Flush_Tables reçue | Purger les tableaux d'apprentissage d'adresses MAC monodiffusion et multidiffusion.<br>Si le champ Learning Update Enable de la trame Flush_Tables est défini sur TRUE, envoyer la trame Learning_Update. |

[a] L'attribut Network Topology doit être mis à jour aux événements 1, 2, 3, 53, 54, 59 et 61 avec les valeurs appropriées.

[b] L'attribut Network Status doit être mis à jour aux événements 1, 2, 3, 8, 11, 12, 13, 14, 37, 38, 39, 40, 41, 42, 43, 44, 53, 54, 55, 56, 57, 58, 59, 61, 73, 74, 75, 76, 77, 78, 81 et 82 avec la valeur appropriée.

[c] L'attribut Ring Supervisor Status doit être mis à jour aux événements 1, 13, 14, 37, 38, 53, 54, 59 et 61 avec la valeur appropriée.

[d] L'attribut Ring Faults Count doit être mis à jour aux événements 1, 13, 14, 37, 38, 39, 40, 41, 42, 43, 44, 53, 54, 59 et 61 avec la valeur appropriée.

[e] L'attribut Last Active Node 1 doit être mis à jour aux événements 1, 11, 12, 13, 14, 17, 21, 23, 26, 31, 41 et 43 avec la valeur appropriée.

[f] L'attribut Last Active Node 2 doit être mis à jour aux événements 1, 11, 12, 13, 14, 18, 22, 24, 26, 32, 42 et 44 avec la valeur appropriée.

[g] L'attribut Ring Protocol Participants Count doit être mis à jour aux événements 1, 13, 14, 37, 38 et 49 avec la valeur appropriée.

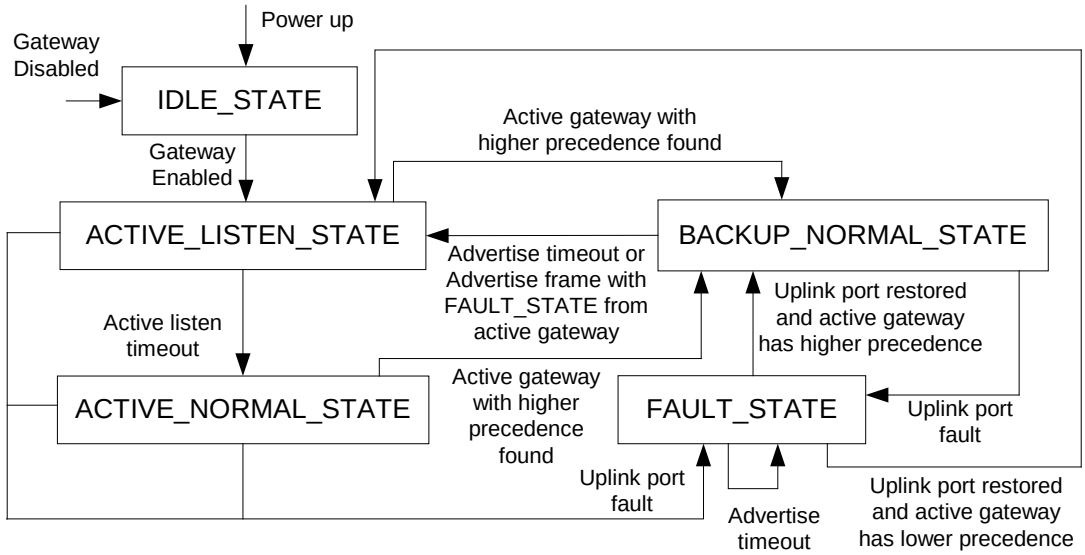
[h] L'attribut Ring Protocol Participants List doit être mis à jour aux événements 1, 13, 14, 37, 38 et 49 avec la valeur appropriée.

[i] Si, dans les événements 11 et 12, les trames Beacon sont reçues en continu pendant plus de 3 durées de délai d'attente consécutives sur le même port sans que l'autre port n'en reçoive, cela constitue une condition de défaut de réseau partiel. Si ce type de condition est détecté, le superviseur d'anneau actif doit bloquer tout le trafic sur le port 2, donnant lieu à une segmentation du réseau et à une mise à jour de l'attribut Network Status. Cette condition implique l'intervention de l'utilisateur. L'emplacement du défaut peut être déterminé grâce au service Verify_Fault_Location.

[j] Si, dans les événements 39, 40, 41, 42, 43 et 44, le superviseur détecte que 5 défauts se sont produits en 30 s, il doit rester à l'état FAULT_STATE et bloquer tout le trafic sur le port 2. Ce type de condition doit être explicitement annulé par l'utilisateur via le service Clear_Rapid_Faults de l'objet DLR.

### 10.10.4 Passerelle redondante

La Figure 41 montre le diagramme de transitions d'états d'une passerelle redondante.




**Légende**

| Anglais | Français |
|---|---|
| Gateway Disabled | Passerelle désactivée |
| Power up | Mise sous tension |
| Gateway enabled | Passerelle activée |
| Active gateway with higher precedence found | Passerelle active avec priorité plus élevée détectée |
| Active listen timeout | Dépassement de délai d'écoute actif |
| Advertise timeout or Advertise frame with FAULT_STATE from active gateway | Dépassement de délai Advertise ou trame Advertise à l'état FAULT_STATE en provenance de la passerelle active |
| Active gateway with higher precedence found | Passerelle active avec priorité moins élevée détectée |
| Uplink port fault | Défaut du port de liaison montante |
| Uplink port restored and active gateway has higher precedence | Port de liaison montante restauré et passerelle active présentant une priorité plus élevée |
| Advertise timeout | Dépassement de délai Advertise |
| Uplink port restored and active gateway has lower precedence | Port de liaison montante restauré et passerelle active présentant une priorité moins élevée |

**Figure 41 – Diagramme de transition d'états de la passerelle redondante**

Le Tableau 161 spécifie les valeurs de paramètre d'un nœud de passerelle redondante.

**Tableau 161 – Valeurs de paramètre du nœud de passerelle redondante**

| Paramètre | Valeur |
|---|---|
| Active gateway Precedence | Valeur par défaut 0 ou valeur configurée |
| Active gateway MAC address | Conforme à la fabrication |
| Advertise Interval duration | Valeur par défaut 2 000 µs ou valeur configurée |
| Advertise Timeout duration | Valeur par défaut 5 000 µs ou valeur configurée |
| Learning Update Enable | Valeur par défaut TRUE ou valeur configurée |
| Active Listen Timeout duration | 1 fois la durée Advertise Timeout |
| Active gateway Precedence (Sauvegarde) | Obtenue à partir de la trame Advertise |
| Active gateway MAC address (Sauvegarde) | Obtenue à partir de la trame Advertise |
| Learning Update Enable (Sauvegarde) | Obtenue à partir de la trame Advertise |
| Advertise Interval duration (Sauvegarde) | Obtenue à partir de la trame Advertise |
| Advertise Timeout duration (Sauvegarde) | Obtenue à partir de la trame Advertise |

Les exigences suivantes s'appliquent à la matrice SEA.

- L'adresse MAC 11-22-33-44-55-66 doit être codée 0x112233445566 pour comparaison numérique en cas de priorité de la passerelle.
- Si les attributs d'une passerelle active (Advertise Interval, Advertise Timeout, Precedence et Learning Update Enable, par exemple) sont modifiés, la passerelle active doit se comporter de manière particulière avant d'appliquer les nouvelles valeurs. Il convient qu'elle bloque la transmission de trafic entre les ports DLR et les ports de liaison montante, arrête tous les temporisateurs et cesse d'envoyer des trames Advertise pendant 1,5 fois la période Advertise Timeout. A l'issue de la période d'attente, il convient qu'elle démarre à partir de l'événement 1 du tableau ci-dessous.
- Si une passerelle de sauvegarde ne peut pas prendre en charge l'intervalle Advertise et/ou le délai d'attente Advertise actif en cours, la passerelle de sauvegarde doit indiquer la condition par l'intermédiaire de l'objet DLR et ne doit pas reprendre en tant que passerelle active en cas de changement de passerelle.
- Si l'attribut de priorité d'une passerelle de sauvegarde est reconfiguré en ligne avec une valeur plus élevée que la passerelle active en cours, il convient qu'elle arrête tous les temporisateurs et démarre à partir de l'événement 1 du tableau ci-dessous immédiatement.
- Contrairement aux autres trames DLR, un appareil passerelle actif doit transmettre les trames Learning_Update aux ports de liaison montante lorsque la transmission de trafic est activée entre les ports DLR et les ports de liaison montante.

- Un appareil passerelle actif à l'état ACTIVE_NORMAL_STATE doit déclarer un défaut de réseau partiel, s'il reçoit une trame Advertise à l'état ACTIVE_NORMAL_STATE pour l'état de passerelle d'un autre appareil passerelle actif présentant une priorité moins élevée (ou un numéro d'adresse MAC inférieur en cas d'interconnexion). Après la déclaration du défaut de réseau partiel, il doit passer à l'état FAULT_STATE, cesser d'envoyer des trames Advertise et bloquer la transmission de trafic entre les ports DLR et les ports de liaison montante. Il doit indiquer la condition par l'intermédiaire de l'attribut Redundant Gateway Status dans l'objet DLR. Il doit définir les attributs Active Gateway Address et Active Gateway Precedence sur ceux de l'autre passerelle active présentant une priorité inférieure. Lorsque l'utilisateur supprime la condition via le service Clear_Gateway_Partial_Fault, la passerelle doit supprimer la condition et démarrer à partir de l'événement 1 du tableau ci-dessous.
- Lorsqu'un changement de passerelle survient, le nouvel appareil passerelle actif doit envoyer sur ses ports de liaison montante le message de notification de changement de topologie qui est approprié au protocole actuellement activé sur ses ports de liaison montante.

Le Tableau 162 spécifie la matrice SEA d'un nœud de passerelle redondante.

**Tableau 162 – Matrice SEA du nœud de passerelle redondante**

| Evénement n° | Etat actuel | Evénement | Action(s)[a] |
|---|---|---|---|
| 1 | Aucun | Mise sous tension | Initialiser<br>Bloquer la transmission du trafic entre les ports DLR et les ports de liaison montante<br>Transition vers IDLE_STATE |
| 2 | IDLE_STATE | Passerelle redondante activée | Démarrer le temporisateur d'écoute active pour la durée Active Listen Timeout<br>Envoyer la trame Advertise avec l'état ACTIVE_LISTEN_STATE pour l'état de passerelle<br>Démarrer le temporisateur d'intervalle Advertise avec la durée Advertise Interval<br>Transition vers ACTIVE_LISTEN_STATE |
| 3 | ACTIVE_LISTEN_STATE | Une trame Advertise en provenance d'une autre passerelle active présentant une priorité plus élevée (ou un numéro d'adresse MAC supérieur en cas d'interconnexion) est reçue et le champ d'état de la passerelle n'est pas défini sur FAULT_STATE | Sauvegarder les champs MAC address, Precedence, Advertise Interval, Advertise Timeout et Learning Update Enable de la passerelle active<br>Arrêter le temporisateur d'intervalle Advertise<br>Arrêter le temporisateur d'écoute active<br>Démarrer le temporisateur de délai d'attente Advertise avec la durée Advertise Timeout<br>Transition vers BACKUP_NORMAL_STATE |
| 4 | ACTIVE_LISTEN_STATE | Temporisateur d'intervalle Advertise expiré | Envoyer la trame Advertise<br>Redémarrer le temporisateur d'intervalle Advertise avec la durée Advertise Interval |

| Evénement n° | Etat actuel | Evénement | Action(s)[a] |
|---|---|---|---|
| 5 | ACTIVE_LISTEN_STATE | Temporisateur d'écoute active expiré | Envoyer la trame Advertise avec l'état ACTIVE_NORMAL_STATE pour l'état de passerelle<br>Redémarrer le temporisateur d'intervalle Advertise avec la durée Advertise Interval<br>Permettre la transmission du trafic entre les ports DLR et les ports de liaison montante<br>Envoyer la trame Send Flush_Tables à l'ensemble des nœuds DLR<br>Envoyer la trame Learning_Update si l'attribut Learning Update Enable de l'objet DLR est défini sur TRUE<br>Transition vers ACTIVE_NORMAL_STATE |
| 6 | ACTIVE_LISTEN_STATE | Connexion de liaison montante ou défaillance de plus haut niveau | Envoyer la trame Advertise avec l'état FAULT_STATE pour l'état de passerelle<br>Arrêter le temporisateur d'intervalle Advertise<br>Arrêter le temporisateur d'écoute active<br>Démarrer le temporisateur de délai d'attente Advertise avec la durée Advertise Timeout<br>Transition vers FAULT_STATE |
| 7 | ACTIVE_NORMAL_STATE | Une trame Advertise en provenance d'une autre passerelle active présentant une priorité plus élevée (ou un numéro d'adresse MAC supérieur en cas d'interconnexion) est reçue et le champ d'état de la passerelle n'est pas défini sur FAULT_STATE | Sauvegarder les champs MAC address, Precedence, Advertise Interval, Advertise Timeout et Learning Update Enable de la passerelle active<br>Arrêter le temporisateur d'intervalle Advertise<br>Démarrer le temporisateur de délai d'attente Advertise avec la durée Advertise Timeout<br>Bloquer la transmission du trafic entre les ports DLR et les ports de liaison montante<br>Transition vers BACKUP_NORMAL_STATE |
| 8 | ACTIVE_NORMAL_STATE | Temporisateur d'intervalle Advertise expiré | Envoyer la trame Advertise<br>Redémarrer le temporisateur d'intervalle Advertise avec la durée Advertise Interval |
| 9 | ACTIVE_NORMAL_STATE | Connexion de liaison montante ou défaillance de plus haut niveau | Envoyer la trame Advertise avec l'état FAULT_STATE pour l'état de passerelle<br>Arrêter le temporisateur d'intervalle Advertise<br>Démarrer le temporisateur de délai d'attente Advertise avec la durée Advertise Timeout<br>Bloquer la transmission du trafic entre les ports DLR et les ports de liaison montante<br>Transition vers FAULT_STATE |
| 10 | FAULT_STATE | La connexion de liaison montante est restaurée et la passerelle active a une priorité plus élevée (ou un numéro d'adresse MAC supérieur en cas d'interconnexion) | Démarrer le temporisateur de délai d'attente Advertise avec la durée Advertise Timeout<br>Transition vers BACKUP_NORMAL_STATE |

| Evénement n° | Etat actuel | Evénement | Action(s)[a] |
|---|---|---|---|
| 11 | FAULT_STATE | La connexion de liaison montante est restaurée et la passerelle active a une priorité moins élevée (ou un numéro d'adresse MAC inférieur en cas d'interconnexion) | Démarrer le temporisateur d'écoute active pour la durée Active Listen Timeout<br>Envoyer la trame Advertise avec l'état ACTIVE_LISTEN_STATE pour l'état de passerelle<br>Démarrer le temporisateur d'intervalle Advertise avec la durée Advertise Interval<br>Transition vers ACTIVE_LISTEN_STATE |
| 12 | FAULT_STATE | Trame Advertise en provenance de la passerelle active reçue | Redémarrer le temporisateur de délai d'attente Advertise avec la durée Advertise Timeout |
| 13 | FAULT_STATE | Expiration du délai d'attente du temporisateur Advertise | Mise à zéro des champs MAC address et Precedence de la passerelle active |
| 14 | FAULT_STATE | Une trame Advertise en provenance d'une autre passerelle active présentant une priorité plus élevée que la passerelle active en cours (ou un numéro d'adresse MAC supérieur en cas d'interconnexion) est reçue et le champ d'état de la passerelle n'est pas défini sur FAULT_STATE | Sauvegarder les champs MAC address, Precedence, Advertise Interval, Advertise Timeout et Learning Update Enable de la passerelle active |
| 15 | BACKUP_NORMAL_STATE | Connexion de liaison montante ou défaillance de plus haut niveau | Transition vers FAULT_STATE |
| 16 | BACKUP_NORMAL_STATE | Trame Advertise en provenance de la passerelle active reçue | Redémarrer le temporisateur de délai d'attente Advertise avec la durée Advertise Timeout |
| 17 | BACKUP_NORMAL_STATE | Expiration du délai d'attente du temporisateur Advertise | Envoyer la trame Advertise avec l'état ACTIVE_LISTEN_STATE pour l'état de passerelle<br>Démarrer le temporisateur d'intervalle Advertise avec la durée Advertise Interval<br>Transition vers ACTIVE_LISTEN_STATE |
| 18 | BACKUP_NORMAL_STATE | Une trame Advertise en provenance d'une autre passerelle active présentant une priorité plus élevée que la passerelle active en cours (ou un numéro d'adresse MAC supérieur en cas d'interconnexion) est reçue et le champ d'état de la passerelle n'est pas défini sur FAULT_STATE | Sauvegarder les champs MAC address, Precedence, Advertise Interval, Advertise Timeout et Learning Update Enable de la passerelle active<br>Redémarrer le temporisateur de délai d'attente Advertise avec la durée Advertise Timeout |

[a] L'attribut Redundant Gateway Status doit être mis à jour sur les événements 2, 3, 6, 7, 9, 10, 11, 15 et 17 avec la valeur appropriée.

## 10.11 Analyse de performance

### 10.11.1 Généralités

L'exemple ci-dessous de calcul des performances s'appuie sur les paramètres/hypothèses essentiels présentés dans le Tableau 163.

**Tableau 163 – Paramètres/hypothèses de l'exemple de calcul des performances**

| Variable | Description | Exemple de valeur |
|---|---|---|
| NumNodes | Nombre de nœuds du réseau en anneau | 50 |
| BcnFrmDly | Délai dû à une trame Beacon dans la commutation de message | 7 µs (trame de 64 octets avec surdébit filaire) |
| AvgEipFrmDly | Délai dû à la trame CP 2/2 moyenne | 12 µs (trame de 128 octets avec surdébit filaire) |
| MaxFrmDly | Délai dû à la trame Ethernet de taille maximale | 124 µs (trame de 1 522 octets avec surdébit filaire) |
| MaxDlyNodePrcnt | Pourcentage du nombre de nœuds dans l'anneau sur lesquels la trame Beacon sera retardée par la trame Ethernet de taille maximale | 10 % |
| IntSwtchDly | Délai de commutation interne sur chaque nœud | 5 µs |
| WirePrpgtDly | Délai de propagation du signal sur un support en cuivre de 100 m | 1 µs |
| NodeProcDly | Délai de traitement sur les nœuds en réponse aux trames et événements de l'anneau | 25 µs |
| BcnIntrvl | Intervalle Beacon. Doit être inférieur à la moitié du RPI de connexion le plus rapide | 400 µs |

Afin d'offrir des performances prévisibles en matière de détection des défauts du réseau et de reconfiguration, tous les nœuds placés directement sur l'anneau doivent dédier la file d'attente à priorité la plus élevée aux trames de protocole d'anneau et mettre en œuvre une planification à priorité stricte pour ladite file d'attente. Grâce à ce type de configuration, une trame de protocole d'anneau ne fera l'objet que d'un faible délai de trame de priorité sur chaque nœud. Dans le cadre de l'analyse des performances, supposer que toutes les liaisons du réseau fonctionnent à 100 Mbit/s et en mode duplex simultané.

Les trames Ring Beacon sont composées de 64 octets (séquence de contrôle de trame incluse) et présentent un surdébit filaire de 20 octets, dont 8 octets sont consacrés au préambule et au délimiteur de début de trame et 12 octets à l'espace inter-trame. Les trames Ring Beacon à surdébit filaire de 20 octets prennent environ BcnFrmDly = 7 µs sur le fil.

Les trames Ring Beacon seront supposées être supprimées sur la plupart des nœuds par une trame de priorité plus basse de taille moyenne de 128 octets, séquence de contrôle de trame incluse. Sur certains nœuds d'un réseau accueillant essentiellement un trafic CP 2/2, les trames Ring Beacon ne peuvent pas être supprimées du tout, alors que dans certains autres nœuds, elles peuvent être supprimées par des trames de 256 octets, et dans d'autres encore, elles peuvent l'être par des trames se situant entre ces deux extrêmes, pour une moyenne de 128 octets. Une trame de 128 octets à surdébit filaire de 20 octets prend environ AvgEipFrmDly = 12 µs sur le fil.

Les nœuds sont supposés utiliser l'architecture de commutation de message et présenter un délai de commutation interne moyen de IntSwtchDly = 5 µs. Soit un délai de propagation d'un support en cuivre de 100 mètres de WirePrpgtDly = 1 µs. Le délai total classique des trames Ring Beacon est donc:

TypclDlyPerNode = BcnFrmDly + AvgEipFrmDly + IntSwtchDly + WirePrpgtDly

TypclDlyPerNode = 7 + 12 + 5 + 1 = 25 µs

Les trames Ring Beacon sont supposées être également supprimées sur MaxDlyNodePrcnt = 10 % des nœuds par des trames Ethernet de taille maximale de 1 522 octets chacune ou une certaine combinaison de grandes trames sur plus de 10 % des nœuds, égal à 10 % des nœuds dotés de trames de taille maximale. Ce type de trames peut être présent sur le réseau pour certaines raisons, notamment configuration, IHM ou web, par exemple. Une trame de 1 522 octets à surdébit filaire de 20 octets prend environ MaxFrmDly = 124 µs sur le fil. Le délai total maximal par nœud des trames Ring Beacon sur ces nœuds est donc:

MaxDlyPerNode = BcnFrmDly + MaxFrmDly + IntSwtchDly + WirePrpgtDly

MaxDlyPerNode = 7 + 124 + 5 + 1 = 137 µs

Dans le cas d'un réseau en anneau composé de NumNodes = 50 nœuds, la durée totale maximale du cycle des trames Beacon est donc:

MaxRndTripTime = (NumNodes × (1 – MaxDlyNodePrcnt) × TypclDlyPerNode)

+ (NumNodes × MaxDlyNodePrcnt × MaxDlyPerNode)

MaxRndTripTime = (50 × (1 – 0,1) × 25) + (50 × 0,1 × 137) = 1 810 µs

Pour le même réseau, le délai minimal par nœud est égal à une trame Beacon n'étant pas retardée par une autre trame, et donc:

MinDlyPerNode = BcnFrmDly + IntSwtchDly + PrpgtDly

MinDlyPerNode = 7 + 5 + 1 = 13 µs

Dans le cas d'un réseau en anneau composé de NumNodes = 50 nœuds, la durée totale minimale du cycle des trames Beacon est donc:

MinRndTripTime = NumNodes × MinDlyPerNode

MinRndTripTime = 13 × 50 = 650 µs

En général, un intervalle Beacon plus bas offre des performances de remise de l'anneau plus rapides. L'intervalle Beacon doit être inférieur à la moitié du RPI de connexion le plus rapide afin d'éviter les délais de connexion. Supposer un intervalle Beacon de BcnIntrvl = 400 µs, représentant 1,75 % de la durée de transmission de liaison du réseau, ce qui est adapté à des connexions CIP hautes performances avec RPI de 1 ms, et fonctionne également avec des connexions d'E-S plus lentes.

Pour choisir le délai d'attente Beacon, tenir compte d'une première trame Beacon transmise à l'instant $Tx_1$ avec durée minimale du cycle et arrivant aux ports du superviseur actif à l'instant:

$Rx_1 = Tx_1$ + MinRndTripTime = $Tx_1$ + 650 µs

Soit une deuxième trame Beacon transmise à l'instant:

$Tx_2 = Tx_1$ + BcnIntrvl = $Tx_1$ + 400 µs

est perdue en route suite à la présence d'une trame corrompue.

Une troisième trame Beacon transmise à l'instant $Tx_3 = Tx_2$ + BcnIntrvl = $Tx_2$ + 400 µs avec un retard de cycle maximal au niveau des ports du superviseur actif à l'instant:

$Rx_3 = Tx_3$ + MaxRndTripTime = $Tx_3$ + 1 810 µs

Le délai d'arrivée maximal de la troisième trame Beacon sur les ports du superviseur actif après la première trame Beacon est donc égal à:

$Rx_3 – Rx_1 = (Tx_3 + 1\ 810) – (Tx_1 + 650) = (Tx_1 + 400 + 400 + 1\ 810) – (Tx_1 + 650) = 1\ 960$ µs

Par conséquent, la durée de délai d'attente beacon doit être égale à:

BcnTimeout = 1 960 µs

Soit un délai de traitement de nœud d'extrémité de NodeProcDly = 25 µs en réponse aux trames ou événements de l'anneau.

Pour le réseau décrit, ce qui suit représentera les performances les moins favorables concernant les nœuds d'anneau s'appuyant sur un mécanisme de trame Beacon:

a) Défauts qui sont détectables dans la couche physique. Il s'agit du type de défauts le plus commun. Le scénario le moins favorable a lieu lorsque le défaut se produit à mi-chemin du réseau en anneau par rapport au superviseur d'anneau actif. Les trames Link_Status mettront la moitié d'une durée de cycle pour passer des nœuds avoisinants le défaut au superviseur d'anneau actif. Une autre moitié de durée de cycle sera nécessaire à la trame Beacon à l'état FAULT_STATE pour passer du superviseur actif au nœud le plus éloigné, ou aux nœuds d'anneau pour arriver au bout du délai des trames Beacon, selon ce qui se produit le plus tôt. Le délai de traitement du nœud d'extrémité intervient trois fois: une fois au voisinage du défaut, une fois au niveau du superviseur et une fois au niveau du nœud le plus éloigné. Le délai total le moins favorable est donc:

   PhysclLyrFltDlyBcn = (2 × 0,5 × MaxRndTripTime) + (3 × NodeProcDly)

   PhysclLyrFltDlyBcn = 1 810 + (3 × 25) = 1 885 µs

b) Défauts qui ne sont pas détectables dans la couche physique. Ce type de défaut est relativement rare. Le scénario le moins favorable a lieu lorsque le défaut se produit à mi-chemin du réseau en anneau par rapport au superviseur d'anneau actif. La dernière trame Beacon mettra une moitié de durée de cycle pour passer de l'emplacement du défaut au superviseur d'anneau actif. Une autre période de délai d'attente Beacon sera nécessaire au superviseur actif et aux nœuds d'anneau pour arriver au bout du délai des trames Beacon. Le délai de traitement de nœud d'extrémité intervient une seule fois au niveau de tous les nœuds. Le délai total le moins favorable est donc:

   NonPhysclLyrFltDlyBcn = (0,5 × MaxRndTripTime) + BcnTimeout + NodeProcDly

   NonPhysclLyrFltDlyBcn = (0,5 × 1 810) + 1 960 + 25 = 2 890 µs

c) Restauration du réseau en mode de fonctionnement normal. Il s'agit du total d'un intervalle Beacon à générer, d'une durée de cycle maximale de la trame Beacon et d'une autre durée de moitié de cycle maximale de la trame Beacon à l'état RING_NORMAL_STATE pour atteindre le nœud le plus éloigné. Le délai de traitement de nœud d'extrémité intervient deux fois: une fois au niveau du superviseur et une fois au niveau de tous les nœuds. Le délai total le moins favorable est donc:

   RingRstrDlyBcn = BcnIntrvl + (1,5 × MaxRndTripTime) + (2 × NodeProcDly)

   RingRstrDlyBcn = 400 + (1,5 × 1 810) + (2 × 25) = 3 165 µs

Pour le réseau décrit, ce qui suit représentera les performances les moins favorables concernant les nœuds d'anneau s'appuyant sur un mécanisme de trame Announce:

a) Défauts qui sont détectables dans la couche physique. Il s'agit du type de défauts le plus commun. Le scénario le moins favorable a lieu lorsque le défaut se produit à mi-chemin du réseau en anneau par rapport au superviseur d'anneau actif. Les trames Link_Status mettront la moitié d'une durée de cycle pour passer des nœuds avoisinants le défaut au superviseur d'anneau actif. Une autre moitié de durée de cycle sera nécessaire à la trame Announce à l'état FAULT_STATE pour passer du superviseur d'anneau actif au nœud le plus éloigné. Le délai de traitement du nœud d'extrémité intervient trois fois: une fois au voisinage du défaut, une fois au niveau du superviseur et une fois au niveau du nœud le plus éloigné. Le délai total le moins favorable est donc

   PhysclLyrFltDlyAnnc = (2 × 0,5 × MaxRndTripTime) + (3 × NodeProcDly)

   PhysclLyrFltDlyAnnc = 1 810 + (3 × 25) = 1 885 µs

b) Défauts qui ne sont pas détectables dans la couche physique. Ce type de défaut est relativement rare. Le scénario le moins favorable a lieu lorsque le défaut se produit à mi-chemin du réseau en anneau par rapport au superviseur d'anneau actif. La dernière trame Beacon mettra une moitié de durée de cycle pour passer de l'emplacement du défaut au

superviseur d'anneau actif. Une autre période de délai d'attente Beacon sera nécessaire au superviseur actif pour arriver au bout du délai des trames Beacon. Une autre moitié de durée de cycle sera nécessaire à la trame Announce à l'état FAULT_STATE pour passer du superviseur actif au nœud le plus éloigné. Le délai de traitement de nœud d'extrémité intervient deux fois: une fois au niveau du superviseur et une fois au niveau du nœud d'extrémité. Le délai total le moins favorable est donc:

NonPhysclLyrFltDlyAnnc = (2 × 0,5 × MaxRndTripTime) + BcnTimeout + (2 × NodeProc Dly)

NonPhysclLyrFltDlyAnnc = 1 810 + 1 960 + (2 × 25) = 3 820 µs

c) Restauration du réseau en mode de fonctionnement normal. Il s'agit du total d'un intervalle Beacon à générer, d'une durée de cycle maximale de la trame Beacon et d'une autre durée de cycle maximale de la trame Announce pour atteindre le nœud le plus éloigné. Le délai de traitement de nœud d'extrémité intervient deux fois: une fois au niveau du superviseur et une fois au niveau du nœud d'extrémité. Il s'agit du délai total d'une trame Announce en fonction du nœud pour purger sa table MAC à diffusion individuelle. L'anneau aurait été restauré plus tôt par le calcul des nœuds Beacon. Le délai total le moins favorable est donc:

RingRstrDlyAnnc = BcnIntrvl + (2 × MaxRndTripTime) + (2 × NodeProcDly)

RingRstrDlyAnnc = 400 + (2 × 1 810) + (2 × 25) = 4 070 µs

En s'appuyant sur des calculs analogues, le Tableau 164 indique les paramètres de configuration et numéros de performances les moins favorables pour différents numéros de nœud de réseau en anneau.

NOTE Malgré la validité de ces numéros de performances dans la plupart des cas, des écarts par rapport aux hypothèses présentées ci-dessus impliqueront de les recalculer conformément aux procédures décrites ci-dessus. Par exemple, si une liaison est configurée pour fonctionner à 10 Mbit/s et/ou en mode duplex non simultané, les numéros doivent être ajustés (pour 10 Mbit/s, multiplier par 10).

**Tableau 164 – Exemple de paramètres et de performances d'une configuration en anneau**

| Nombre de nœuds d'anneau | Intervalle Beacon (µs) | Durée cycle[a] (µs) | Délai attente Beacon (µs) | Délai de remise après défauts de la couche physique[a] (µs) | Délai de remise après défauts de la couche non physique des nœuds de trame Beacon (µs) | Délai de remise après défauts de la couche non physique des nœuds de trame Announce (µs) | Délai de restauration d'anneau des nœuds de trame Beacon (µs) | Délai de restauration d'anneau des nœuds de trame Announce (µs) |
|---|---|---|---|---|---|---|---|---|
| 25 | 400 | 905 | 1 380 | 980 | 1 858 | 2335 | 1 808 | 2 260 |
| 50 (taille nominale du réseau) | 400 | 1 810 | 1 960 | 1 885 | 2 890 | 3 820 | 3 165 | 4 070 |
| 100 | 400 | 3 620 | 3 120 | 3 695 | 4 955 | 6 790 | 5 880 | 7 690 |
| 150 | 400 | 5 430 | 4 280 | 5 505 | 7 020 | 9 760 | 8 595 | 11 310 |
| 200 | 400 | 7 240 | 5 440 | 7 315 | 9 085 | 12 730 | 11 310 | 14 930 |
| 250 | 400 | 9 050 | 6 600 | 9 125 | 11 150 | 15 700 | 14 025 | 18 550 |

[a] Identique pour les nœuds de trames Beacon et Announce.

Important: Si l'anneau contient des nœuds non DLR, les délais de remise et de restauration sont identiques à ceux des nœuds de trame Announce, à condition que les nœuds non DLR

soient conformes aux exigences en 10.5. Si les nœuds non DLR ne sont pas conformes aux exigences, les délais de remise et de restauration sont imprévisibles.

En cas d'utilisation de commutateurs non DLR dans l'anneau, une perte de paquets à diffusion individuelle peut se produire, comme décrit en 10.7.3 (Commutateurs non DLR).

### 10.11.2 Performances de changement de passerelle redondante

L'exemple ci-dessous de calcul des performances s'appuie sur les paramètres/hypothèses essentiels présentés dans le Tableau 165.

**Tableau 165 – Variables pour l'analyse des performances**

| Variable | Description | Exemple de valeur |
|---|---|---|
| NodeProcDly | Délai de traitement sur les nœuds en réponse aux trames et événements de l'anneau | 25 µs |
| MaxRndTripTime | 1 810 µs pour 50 nœuds de 10.11.1 | |
| AdvrtsIntrvl | Intervalle Advertise, il convient qu'il soit supérieur à MaxRndTripTime | 2 000 µs |
| AdvrtsTimeout | Délai d'attente Advertise, il convient qu'il soit 2,5 fois égal à AdvrtsIntrvl | 5 000 µs |
| ActiveListenTimeout | 1 fois égal à AdvrtsTimeout | 5 000 µs |
| LearnUpdtPropDly | Délai de propagation de la trame de mise à jour de l'apprentissage via DLR aux commutateurs non DLR situés en dehors du réseau DLR, il convient qu'il soit supérieur à MaxRndTripTime | 5 000 µs |

Performances de changement de passerelle redondante pour trois cas de défaillance, en supposant que Learning Update Enable a été défini sur TRUE, comme suit.

a) Echec de connexion de liaison montante détecté par la passerelle active au niveau de la couche physique: 13,745 ms

Il s'agit du type de défauts le plus commun. La passerelle active détectera la défaillance de la couche physique et enverra une trame Advertise avec l'état FAULT_STATE. Cela peut prendre MaxRndTripTime à la trame Advertise pour atteindre l'appareil passerelle de sauvegarde. L'appareil passerelle de sauvegarde prendra ActiveListenTimeout pour permettre la transmission du trafic et l'envoi de la trame Flush_Tables. La trame Flush_Tables peut prendre MaxRndTripTime pour atteindre le nœud DLR le plus éloigné. Le nœud DLR le plus éloigné enverra la trame Learning_Update qui peut prendre LearnUpdtPropDly pour mettre à jour les commutateurs non DLR situés hors du réseau DLR. Les délais de traitement des nœuds d'extrémité interviennent 5 fois au total. Le délai total le moins favorable est donc:

$$\begin{aligned} \text{GtwyPhysclLyrFltDly} &= (2 \times \text{MaxRndTripTime}) + \text{ActiveListenTimeout} + \\ &\quad \text{LearnUpdtPropDly} + (5 \times \text{NodeProcDly}) \\ &= (2 \times 1\,810) + 5\,000 + 5\,000 + (5 \times 25) \\ &= 13\,745\ \mu\text{s}. \end{aligned}$$

b) Défaillance de la passerelle active: 18,695 ms

La passerelle active pourrait échouer juste après l'envoi de la dernière trame Advertise. Cela peut prendre MaxRndTripTime pour atteindre l'appareil passerelle de sauvegarde. L'appareil passerelle de sauvegarde patientera pendant AdvrtsTimeout avant de déclarer la passerelle active comme étant perdue. L'appareil passerelle de sauvegarde prendra encore ActiveListenTimeout pour permettre la transmission du trafic et l'envoi de la trame Flush_Tables. La trame Flush_Tables peut prendre MaxRndTripTime pour atteindre le

nœud DLR le plus éloigné. Le nœud DLR le plus éloigné enverra la trame Learning_Update qui peut prendre LearnUpdtPropDly pour mettre à jour les commutateurs non DLR situés hors du réseau DLR. Les délais de traitement des nœuds d'extrémité interviennent 3 fois au total. Le délai total le moins favorable est donc:

GtwyFailDly = (2 × MaxRndTripTime) + AdvrtsTimeout + ActiveListenTimeout + LearnUpdtPropDly + (3 × NodeProcDly)
= (2 × 1 810) + 5 000 + 5 000 + 5 000 + (3 × 25)
= 18 695 µs.

c) Détection d'un défaut de liaison montante sur une couche supérieure: 6,013 745 s

Un défaut de liaison montante sur une couche supérieure (ex: autre qu'un défaut de liaison reporté par l'entité PHY) pourrait être détecté par la passerelle active via un protocole (ex: RSTP) sur les ports de liaison montante. Ce cas dépend des performances de détection de défauts de couche supérieure de ce protocole. Par exemple, avec RSTP, le délai de détection d'un défaut de couche supérieure pourrait s'élever à 6 s (défauts typiques) pour les trames RSTP Hello d'expiration. Une fois le défaut détecté, les délais additionnels sont égaux à GtwyPhysclLyrFltDly vu que le même ensemble d'actions a besoin d'être effectué. Le délai total le moins favorable pour RSTP est donc:

GtwyRstpLyrFltDly = RstpFltDly + GtwyPhysclLyrFltDly
= 6 000 000 + 13 745
= 6 013 745 µs.

# Annexe A
(normative)

# Voyants et commutateurs

## A.1 Objectif

Il est recommandé d'adopter une approche commune du comportement des voyants et commutateurs d'état afin d'assurer une interface utilisateur cohérente pour tous les appareils conformes à la présente norme. Ces voyants (en général des DEL) facilitent la maintenance et le diagnostic des problèmes liés aux supports et aux couches physiques et de données.

NOTE Tout au long de l'Annexe A, le terme « réseau » fait référence à la partie du sous-réseau Ph et du sous-réseau DL connectée à l'appareil procédant à l'observation.

## A.2 Voyants

### A.2.1 Exigences générales relatives aux voyants

Les profils CPF 2 comportent différents niveaux d'exigences de prise en charge des voyants d'état. Toutefois, si un appareil ne prend pas en charge les voyants d'état, ils doivent se conformer aux spécifications décrites dans cette annexe pour le CP sélectionné.

Il est vivement recommandé de mettre en œuvre ces voyants dans tous les appareils pour lesquels aucune contrainte pratique ne l'empêche.

NOTE Cela parce que les voyants permettent au personnel de maintenance d'identifier rapidement une unité ou un support défectueux (des voyants rouges sont souvent utilisés pour indiquer une anomalie, par exemple).

Deux types de voyants d'état peuvent être fournis:

– un voyant d'état du module;
– un ou plusieurs voyants d'état du réseau.

Ces voyants peuvent parfois être combinés.

D'autres voyants peuvent être présents. Toutefois, les conventions de dénomination et de symbole des voyants standard ne doivent pas être utilisées pour d'autres voyants.

Les appareils de sécurité font l'objet d'exigences supplémentaires (voir la CEI 61784-3-2).

### A.2.2 Exigences communes relatives aux voyants

#### A.2.2.1 Applicabilité des exigences communes

Les exigences communes doivent uniquement concerner les voyants dont les exigences sont spécifiées dans la présente norme.

#### A.2.2.2 Visibilité des voyants

Les voyants doivent être visibles sans retirer les capots ou d'autres parties de l'équipement. Les voyants doivent être visibles aisément à la lumière naturelle. Toutes les étiquettes et icônes connexes doivent être visibles, que le voyant soit allumé ou non.

### A.2.2.3 Vitesse de clignotement du voyant

Sauf spécification contraire, un voyant binaire doit présenter une vitesse de clignotement de (1,0 ± 0,1) Hz. Le voyant doit avoir un cycle continu de (50 ± 5) %.

NOTE Le voyant est à l'état initial (on) pendant environ 0,5 s et à l'état secondaire (off) pendant environ 0,5 s.

### A.2.2.4 Voyant d'état du module

#### A.2.2.4.1 Description

Le voyant bicolore (vert/rouge) doit indiquer l'état de l'ensemble de l'appareil. Il doit indiquer si l'appareil est correctement mis sous tension, configuré et exploité.

NOTE Un appareil doté de plusieurs ports de communication ne comporte qu'un seul voyant d'état du module.

#### A.2.2.4.2 Etats

Les états du voyant d'état du module doivent être conformes au Tableau A.1. Les états du voyant reflètent les états de l'appareil spécifiés dans l'objet Identity (voir la CEI 61158-5-2).

**Tableau A.1 – Voyant d'état du module**

| Etat du voyant | Etat de l'appareil | Cause |
|---|---|---|
| Fixe inactif | Pas d'alimentation | Aucune source d'alimentation n'est appliquée à l'appareil. |
| Fixe vert | Opérationnel | L'appareil fonctionne en condition normale. |
| Vert clignotant | En veille | L'appareil doit être mis en service suite à une configuration manquante, incorrecte ou incomplète. |
| Rouge clignotant | Panne récupérable | L'appareil a détecté une panne récupérable majeure (voir NOTE 1). |
| Rouge fixe | Panne irrécupérable | L'appareil a détecté une panne irrécupérable majeure; peut faire l'objet d'un remplacement. |
| Rouge/vert clignotant | Autotest de l'appareil | L'appareil exécute ses autotests d'alimentation. |
| NOTE 1 Les vitesses de clignotement du voyant sont spécifiées en A.2.2.3. | | |
| NOTE 2 Une configuration incorrecte ou incohérente est considérée comme une panne récupérable. | | |

Tous les états et toutes les couleurs de voyant non défini(e)s dans le Tableau A.1 sont réservés et ne doivent pas être utilisés.

Les appareils de sécurité font l'objet d'exigences supplémentaires (voir la CEI 61784-3-2).

### A.2.2.1 Indication d'état Time Sync

Un appareil doit fournir l'indication de l'état de synchronisation d'horloge. Le Tableau A.2 montre l'état de synchronisation suggéré pour différents indicateurs.

Un appareil est considéré synchronisé lorsque son horloge locale est synchronisée avec l'horloge mère de référence. Cependant, une application peut imposer des exigences de synchronisation plus rigoureuses.

**Tableau A.2 – Indication d'état Time Sync**

| Indicateur d'état visuel | Opération suggérée |
|---|---|
| Voyant d'état du module | Veille (vert clignotant) = non synchronisé<br>Opérationnel (vert fixe) = synchronisé + autres exigences relatives à l'appareil |
| Affichage alphanumérique multicaractères | Il convient qu'un appareil en attente de synchronisation affiche le message « Synching »<br>Il convient qu'un appareil qui est une horloge PTP Master (mère) affiche le message « Time Master »<br>Il convient qu'un appareil qui est une horloge PTP Grandmaster (grand-mère) affiche le message « Time GM » |
| Page Web ou page de propriété (suggestion) | Identité de l'horloge Grandmaster, Parent et Local<br>Qualité de l'horloge Grandmaster et Local<br>Etat de synchronisation<br>Heure système courante<br>Décalage par rapport à l'horloge mère<br>Etat courant des ports |
| NOTE L'indicateur d'état du module est spécifié en A.2.3, A.2.4 ou A.2.5. | |

### A.2.3 Exigences relatives au voyant spécifique au bus de terrain (1)

#### A.2.3.1 Exigences générales (1)

Deux catégories de voyant d'état doivent être proposées pour chaque appareil CP 2/1:

– un voyant d'état du module;
– deux voyants d'état du réseau.

#### A.2.3.2 Voyants à la mise sous tension

A la mise sous tension, les voyants de type 2 doivent s'allumer à l'état d'anomalie (rouge) ou de réinitialisation (inactif). Le réglage des voyants de type 2 à l'état d'anomalie à la mise sous tension est préféré de sorte qu'un état d'anomalie est reflété dans l'éventualité où le module ne peut pas effectuer avec succès l'un de ses essais de mise sous tension. Pendant la mise sous tension (avant le début de l'exécution d'une application en régime établi), les voyants de type 2 doivent être manœuvrés comme suit pour permettre une inspection visuelle de leur propre fonctionnement.

– Pour le voyant d'état du module (Module Status, MS):
  - allumer le voyant MS en vert, tous les autres voyants de type 2 inactifs;
  - garder le voyant MS vert pendant environ 0,25 s;
  - allumer le voyant MS en rouge pendant environ 0,25 s;
  - allumer le voyant MS en vert.
– Pour les voyants d'état du réseau (Network Status, NS):
  - allumer chaque voyant en vert pendant environ 0,25 s;
  - allumer chaque voyant en rouge pendant environ 0,25 s;
  - désactiver chaque voyant.

– Si d'autres voyants de type 2 sont présents:
  - allumer chaque voyant en vert pendant environ 0,25 s;
  - allumer chaque voyant en rouge pendant environ 0,25 s;
  - désactiver chaque voyant.

Les essais d'inspection visuelle concernant les voyants d'état du réseau et les autres indicateurs de type 2 peuvent être effectués de manière séquentielle ou en parallèle. Après la réalisation des essais d'inspection visuelle, le ou les voyants de type 2 doivent se comporter comme indiqué dans la présente norme.

#### A.2.3.3 Voyant d'état du module

Le comportement du voyant d'état du module doit être conforme à A.2.2.4.

L'une des étiquettes ci-dessous doit être apposée sur le voyant d'état du module:

« MS »;

« OK »;

« état »;

« état mod »;

« état module ».

#### A.2.3.4 Voyants d'état du réseau

##### A.2.3.4.1 Description

L'état du réseau doit être indiqué par deux voyants bicolores (rouge/vert). Si l'appareil comporte des PhL (canaux) redondants, chacun des voyants doit être associé à l'un d'eux. Si l'appareil ne prend pas en charge la redondance de canal, l'un des voyants doit être associé au canal unique. Toutefois, deux voyants doivent toujours être requis.

NOTE Ces voyants indiquent un certain nombre de conditions dans le PhL. Lorsqu'ils s'affichent ensemble, ils peuvent refléter l'état de la couche de liaison de données.

En règle générale, les voyants d'état du réseau ne reflètent pas les erreurs reçues sur le port d'accès au réseau (NAP). Toutefois, les nœuds ne comportant qu'un seul NAP (sans autre variante PhL) peuvent utiliser les voyants d'état du réseau pour représenter l'état du NAP. Si ce comportement facultatif est mis en œuvre, l'état des voyants doit refléter celui de la communication sur le port d'accès au réseau, à la place des canaux A et B.

##### A.2.3.4.2 Etats

###### A.2.3.4.2.1 Définitions

Les définitions suivantes doivent être utilisées pour décrire le comportement des voyants:

**fixe** – le voyant doit être allumé en permanence;

**en alternance** – les deux voyants allumés en même temps doivent passer d'un état défini à un autre en étant toujours à l'état opposé, déphasés;

**clignotant** – chaque voyant, affiché indépendamment, doit passer d'un état à l'autre. Si les voyants clignotent, ils doivent le faire ensemble, en phase.

###### A.2.3.4.2.2 Priorité

Les différents états des voyants d'état du réseau doivent être conformes à un schéma de priorité stricte. En présence de plusieurs conditions, les voyants d'état du réseau doivent afficher la priorité la plus élevée (voir Tableau A.3).

**Tableau A.3 – Voyants d'état du réseau**

| Priorité | Etat du voyant | Affichage | Cause |
|---|---|---|---|
| La plus élevée (1) | Fixes inactifs | Affichés ensemble | Réinitialisation ou absence d'alimentation |
| 2 | Fixes rouges | | Interface de liaison défaillante |
| 3 | Rouge/vert en alternance | | autotest |
| 4 | Rouge/inactif en alternance | | Configuration incorrecte du nœud (MAC ID en double, par exemple) |
| 5 | Fixe inactif | Affichés indépendamment | Canal désactivé ou non pris en charge |
| 6 | Rouge/vert clignotant | | Configuration de liaison non valide |
| 7 | Clignotant rouge/inactif | | Défaut de liaison ou aucune DLPDU reçue |
| 8 | Clignotant vert/inactif | | Erreur de canal provisoire ou en écoute seulement |
| La plus basse (9) | Fixe vert | | Fonctionnement normal |

#### A.2.3.4.2.3 Fixe vert

Pendant le fonctionnement normal, si des DLPDU sont reçues sans erreurs détectées, le voyant d'état du réseau d'un canal activé doit être fixe vert.

#### A.2.3.4.2.4 Rouge fixe

En cas d'anomalie sur l'interface de liaison, les deux voyants doivent être fixes rouges.

#### A.2.3.4.2.5 Fixe inactif

Si l'une des entités PhL est désactivée, le voyant du canal désactivé doit être inactif. Un canal peut être désactivé car l'appareil ne dispose que d'une seule PhL ou il dispose de deux PhL existant sur la liaison non redondante. Seule la désactivation de toute l'interface réseau doit provoquer l'extinction des deux voyants (inactifs).

#### A.2.3.4.2.6 Clignotant vert/inactif

Un voyant clignotant vert/inactif doit signifier une erreur de canal provisoire. Les deux canaux doivent être traités indépendamment de ces erreurs. Une erreur se produisant sur le canal A doit être indiquée par l'un des trois événements suivants:

`Ph_status_indication` du canal A n'est pas Normal;
`DLL_badFCS_indication(CHA)` est évalué;
`DLL_event_indication(DLL_EV_errA)` est évalué.

De même, une erreur se produisant sur le canal B doit être indiquée par l'un des trois événements suivants:

`Ph_status_indication` du canal B n'est pas Normal;
`DLL_badFCS_indication(CHB)` est évalué;
`DLL_event_indication(DLL_EV_errB)` est évalué.

Si plusieurs erreurs se produisent dans une DLPDU, une seule erreur de canal doit être déclarée. Il doit exister au maximum un événement d'erreur par DLPDU et par canal. Chaque PhL doit avoir un `Ph_status_indication` indépendant. La couche de liaison de données doit signaler les erreurs sur au moins le canal sélectionné pour la présentation aux niveaux

supérieurs. La couche de liaison de données peut également signaler des erreurs sur l'autre canal.

L'état du voyant vert doit être entré à chaque fois qu'une erreur est détectée. Le voyant clignotant vert doit devenir fixe vert si aucune erreur n'est détectée pendant 4 s ± 1 s sur un canal donné.

Si la couche de liaison de données est en écoute seulement, le voyant d'un canal activé doit clignoter vert à la priorité 8 (voir Tableau A.3).

NOTE L'objet ControlNet peut être utilisé pour forcer la couche de liaison de données à passer à l'état d'écoute seulement.

#### A.2.3.4.2.7 Clignotant rouge/inactif

Un voyant clignotant rouge/inactif doit signifier une grave erreur de liaison. Si la valeur de `Ph_frame_indication` de l'un des canaux reste FALSE pendant 1,0 s ± 0,1 s, le voyant respectif doit conserver l'état clignotant rouge pendant la période de 1,0 s ± 0,1 s qui suit.

Pour les appareils dont les canaux redondants sont activés, si le taux d'erreur de l'un des canaux dépasse celui de l'autre canal de plus d'une erreur par N DLPDU reçus, le voyant du canal dont le taux est le plus élevé doit clignoter en rouge. N doit être compris entre $10^6$ et $2\times10^6$. L'intervalle utilisé pour mesurer le taux d'erreur du canal doit être de $5\times10^6$ DLPDU.

#### A.2.3.4.2.8 Rouge/vert clignotant

Un voyant clignotant rouge/vert doit signifier une configuration de liaison incorrecte. Les voyants d'état du réseau doivent clignoter rouge/vert si le moniteur de connexion au réseau (NAM) indique que le nœud a été invité à utiliser des paramètres de liaison non valides ou non pris en charge.

Pour éventuellement indiquer un dépassement de capacité de réception ou de transmission de la file d'attente, les voyants d'état du réseau doivent clignoter rouge/vert. Les voyants d'état du réseau doivent rester dans cet état au moins 3 s, même si la condition de dépassement de capacité a été supprimée.

#### A.2.3.4.2.9 Rouge/inactif en alternance

Un voyant rouge/inactif en alternance doit signifier une configuration de nœud non valide. Le voyant doit entrer dans cet état si l'une des valeurs ci-dessous est attribuée à `DLL_event_indication`:

`DLL_EV_dupNode`;
`DLL_EV_rogue`.

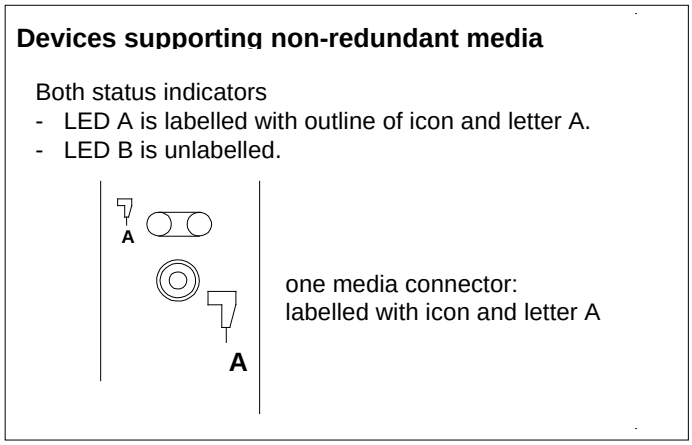
Cet état doit être immédiatement abandonné si la condition d'erreur a été résolue.

#### A.2.3.4.2.10 Rouge/vert en alternance

Un voyant rouge/vert en alternance doit signifier que la couche de liaison de données est en mode d'autotest. Les voyants d'état du réseau doivent rester dans cet état au moins 3 s, même si l'autotest est terminé.

### A.2.3.4.3 Etiquetage

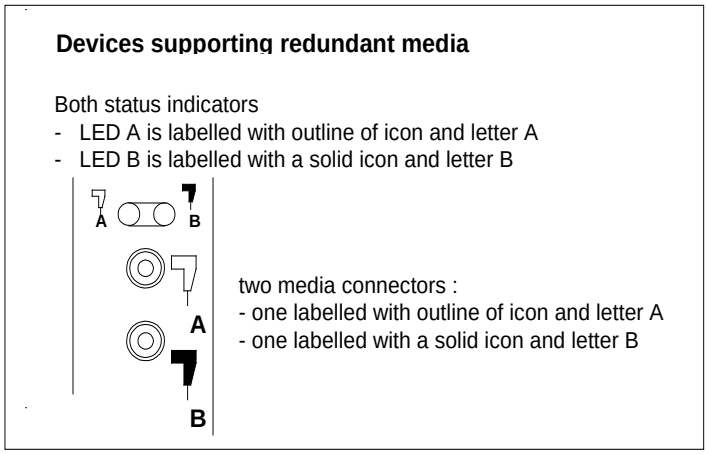
Une étiquette doit être apposée sur les voyants d'état du réseau, en utilisant les symboles présentés dans la Figure A.1 et la Figure A.2. Une étiquette doit être apposée sur le premier canal, représentant le contour d'une icône de canal. Une étiquette doit être apposée sur le deuxième canal, représentant une icône de canal pleine. Une étiquette peut être apposée sur le premier canal et le deuxième canal, avec les lettres A et B majuscules, respectivement.




**Légende**

| Anglais | Français |
|---|---|
| Devices supporting non-redundant media | Appareils prenant en charge des supports non redondants |
| Both status indicators | Deux indicateurs d'état |
| LED A is labeled with outline of icon and letter A | La DEL A est étiquetée avec le contour d'une icône et la lettre A |
| LED B is unlabeled | La DEL B n'est pas étiquetée |
| One media connector labeled with icon and letter A | Un connecteur de support: étiqueté avec une icône et la lettre A |

**Figure A.1 – Etiquetage du voyant d'état du réseau non redondant**




**Légende**

| Anglais | Français |
|---|---|
| Devices supporting redundant media | Appareils prenant en charge des supports redondants |
| Both status indicators | Deux indicateurs d'état |
| LED A is labeled with outline of icon and letter A | La DEL A est étiquetée avec le contour d'une icône et la lettre A |
| LED B is labeled with a solid icon and letter B | La DEL B est étiquetée avec une icône pleine et la lettre B |
| two media connectors:<br>-one labeled with outline of icon and letter B | deux connecteurs de support:<br>- un étiqueté avec contour d'une icône et la lettre A |

| Anglais | Français |
|---|---|
| -one labeled with a solid icon and letter B | - un étiqueté avec une icône pleine et la lettre B |

**Figure A.2 – Etiquetage du voyant d'état du réseau redondant**

### A.2.4 Exigences relatives au voyant spécifique au bus de terrain (2)

#### A.2.4.1 Exigences générales (2)

CP 2/2 n'oblige pas les appareils à disposer de voyants.

Deux types de voyants d'état peuvent être fournis:

– un voyant d'état du module;
– un ou plusieurs voyants d'état du réseau.

NOTE 1 Le consortium de prise en charge du bus de terrain (voir <www.odva.org>) définit un niveau de performances industriel nécessitant un appareil de prise en charge des voyants d'état du module et du réseau, comme indiqué dans la présente norme.

NOTE 2 Un appareil doté de plusieurs ports de communication ne comporte qu'un seul voyant d'état du module, mais plusieurs voyants d'état du réseau (un par port).

NOTE 3 Les appareils sont censés être dotés d'un voyant affichant l'état de la liaison (état de la liaison, tx/rx, collision, par exemple) conformément aux pratiques généralement acceptées de l'industrie (comme ceux utilisés pour les commutateurs, par exemple).

#### A.2.4.2 Voyants à la mise sous tension

Un essai de voyant doit être réalisé à la mise sous tension. Pour permettre un contrôle visuel, la séquence ci-après doit être exécutée:

– premier voyant vert, tous les autres voyants inactifs;
– premier voyant vert pendant environ 0,25 s;
– premier voyant rouge pendant environ 0,25 s;
– premier voyant vert;
– deuxième voyant (le cas échéant) vert pendant environ 0,25 s;
– deuxième voyant (le cas échéant) rouge pendant environ 0,25 s;
– deuxième voyant (le cas échéant) inactif.

En présence d'autres voyants, chacun d'eux doit être soumis à essai en séquence, comme indiqué par le deuxième voyant ci-dessus. En présence d'un voyant d'état du module, il doit s'agir du premier voyant de la séquence, suivi d'un voyant d'état du réseau présent. A l'issue de cet essai à la mise sous tension, le/les voyant(s) doi(ven)t reprendre leur état de fonctionnement normal.

#### A.2.4.3 Voyant d'état du module

Le comportement du voyant d'état du module doit être conforme à A.2.2.4.

L'une des étiquettes ci-dessous doit être apposée sur le voyant d'état du module:

« MS »;
« Mod »;
« Etat mod »;
« Etat module ».

### A.2.4.4 Voyants d'état du réseau

#### A.2.4.4.1 Description

L'état du réseau doit être indiqué par un voyant bicolore (rouge/vert) représentant l'état de l'interface réseau. Les appareils dotés de plusieurs ports de communication physique mais d'une seule adresse IP ne doivent comporter qu'un seul voyant d'état du réseau (par exemple, un appareil à 2 ports qui prend en charge DLR). Les appareils qui prennent en charge plusieurs adresses IP peuvent utiliser un voyant d'état du réseau pour chaque interface d'adresse IP ou un seul voyant pour toutes les interfaces (par comportement dans le Tableau A.4).

#### A.2.4.4.2 Etats

Les états du voyant d'état du réseau doivent être conformes au Tableau A.4.

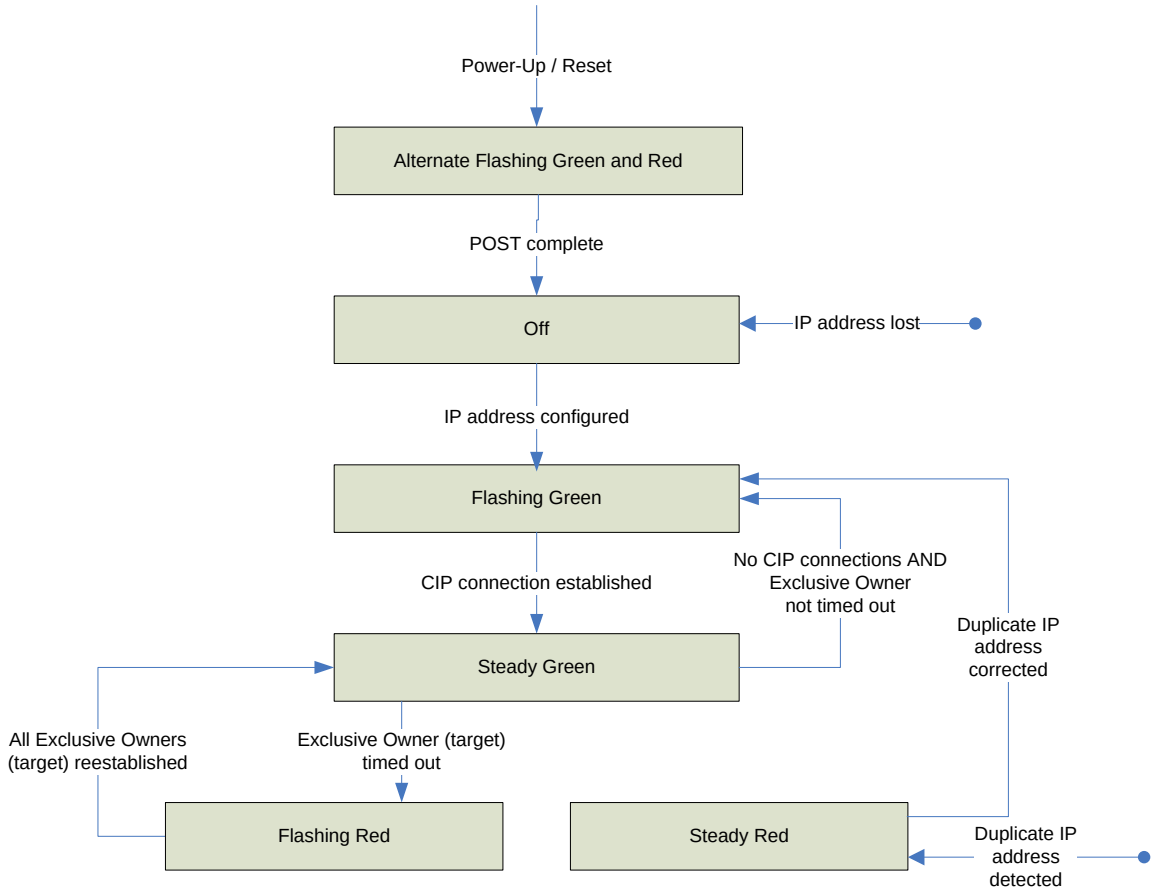
**Tableau A.4 – Voyant d'état du réseau**

| Etat du voyant | Récapitulatif | Cause |
|---|---|---|
| Fixe inactif | Pas d'alimentation pas d'adresse IP | L'appareil est hors tension ou est sous tension mais ne dispose d'aucune adresse IP (attribut de configuration d'interface de l'objet interface TCP/IP). |
| Vert clignotant | Pas de connexion | Une adresse IP est configurée mais aucune connexion de type 2 n'a été établie et une connexion de propriétaire exclusif n'a pas expiré. |
| Fixe vert | Connecté | Au moins une connexion de type 2 (toutes les classes de transport) a été établie et une connexion de propriétaire exclusif n'a pas expiré. |
| Rouge clignotant | Délai de connexion | Une connexion de propriétaire exclusif ayant pour cible cet appareil a expiré. Le voyant d'état du réseau doit devenir à nouveau fixe vert seulement lorsque toutes les connexions de propriétaire exclusif expirées sont rétablies.<br>Les appareils qui prennent en charge une seule connexion de propriétaire exclusif doivent passer à l'état fixe vert lorsque la prochaine connexion de propriétaire exclusif est établie.<br>Les appareils qui prennent en charge plusieurs connexions de propriétaire exclusif doivent retenir les informations sur le chemin de connexion O->T lorsqu'une connexion de propriétaire exclusif expire. Le voyant DEL du réseau doit passer de l'état rouge clignotant à l'état fixe vert lorsque toutes les connexions des points de connexion O->T ayant précédemment expirés sont rétablies.<br>Les délais de connexion autres que ceux des connexions de propriétaire exclusif ne doivent pas entraîner l'état rouge clignotant.<br>L'état rouge clignotant s'applique uniquement aux connexions cible. Les auteurs et les routeurs de type 2 ne doivent pas passer à cet état lorsqu'une connexion d'auteur ou routée de type 2 expire. |
| Rouge fixe | IP en double | Pour les appareils qui prennent en charge la détection des adresses IP en double, l'appareil a détecté qu'au moins l'une de ses adresses IP est déjà utilisée. |
| Clignotant vert/rouge | Autotest | L'appareil exécute ses autotests de mise sous tension (POST). Pendant les autotests POST, le voyant d'état du réseau doit clignoter vert et rouge en alternance. |

Lorsqu'un seul voyant est utilisé pour représenter plusieurs interfaces d'adresse IP, l'état de n'importe quelle interface doit être suffisant pour modifier l'état du voyant (par comportement dans le tableau):

– passage au vert clignotant lorsqu'une interface reçoit une adresse IP;
– passage au vert fixe lorsqu'une connexion de type 2 est établie sur l'une des interfaces (et la connexion de propriétaire exclusif n'a pas expiré);
– passage au rouge clignotant lorsqu'une connexion de type 2 de propriétaire exclusif expire sur l'une des interfaces;
– passage au rouge fixe lorsqu'une des interfaces détecte un conflit au niveau de l'adresse IP.

La Figure A.3 représente un diagramme d'état du voyant d'état du réseau.




**Légende**

| Anglais | Français |
|---|---|
| Power-Up / Reset | Mise sous tension / Réinitialisation |
| Alternate Flashing green and red | Clignotant vert et rouge en alternance |
| POST complete | Autotests POST terminés |
| Off | Hors-tension |
| IP address lost | Adresse IP perdue |
| IP address configured | Adresse IP configurée |
| Flashing green | Vert clignotant |
| CIP connection established | Connexion CIP établie |
| No CIP connections AND exclusive owner not | Aucune connexion CIP ET connexion de |

| Anglais | Français |
|---|---|
| timed out | propriétaire exclusif non expirée |
| Steady green | Vert fixe |
| Duplicate IP address corrected | Adresse IP en double corrigée |
| All exclusive owners (target) reestablished | Toutes les connexions de propriétaire exclusif (cible) rétablies |
| Exclusive owner (target) timed out | Connexion de propriétaire exclusif (cible) expirée |
| Flashing red | Rouge clignotant |
| Steady red | Rouge fixe |
| Duplicate IP address detected | Adresse IP en double détectée |

**Figure A.3 – Diagramme d'état du voyant d'état du réseau**

#### A.2.4.4.3 Etiquetage

L'une des étiquettes ci-dessous doit être apposée sur le voyant d'état du réseau:

« NS »;

« Rés »;

« Etat rés »;

« Etat du réseau ».

### A.2.5 Exigences relatives au voyant spécifique au bus de terrain (3)

#### A.2.5.1 Exigences générales (3)

CP 2/3 n'oblige pas les appareils à disposer de voyants.

Si un appareil prend en charge les voyants, il est recommandé de fournir:

– un voyant d'état du module;
– un voyant d'état;

ou un voyant d'état du réseau/module combiné.

Un appareil peut également prendre en charge les voyants d'état E-S.

En cas d'hésitation entre l'activation d'un voyant rouge ou vert, le voyant doit être activé rouge.

#### A.2.5.2 Voyants à la mise sous tension

##### A.2.5.2.1 Voyants d'état du module et du réseau à la mise sous tension

Un essai de voyant doit être réalisé à la mise sous tension. Pour permettre un contrôle visuel, la séquence ci-après doit être exécutée:

– voyant d'état du réseau inactif;
– voyant d'état du module vert pendant environ 0,25 s;
– voyant d'état du module rouge pendant environ 0,25 s;
– voyant d'état du module vert;
– voyant d'état du réseau vert pendant environ 0,25 s;
– voyant d'état du réseau rouge pendant environ 0,25 s;
– voyant d'état du réseau inactif.

Si un appareil est doté d'autres voyants, il convient de soumettre chacun d'eux en séquence.

#### A.2.5.2.2 Voyants d'état du module/réseau combinés à la mise sous tension

Un essai de voyant doit être réalisé à la mise sous tension. Pour permettre un contrôle visuel, la séquence ci-après doit être exécutée:

– voyant d'état du module/réseau combiné vert pendant environ 0,25 s;
– voyant d'état du module/réseau combiné rouge pendant environ 0,25 s;
– voyant d'état du module/réseau combiné inactif.

Si un appareil est doté d'autres voyants, il convient de soumettre chacun d'eux en séquence.

### A.2.5.3 Voyant d'état du module

Le comportement du voyant d'état du module doit être conforme à A.2.2.4.

Il convient d'apposer l'une des étiquettes ci-dessous sur le voyant d'état du module:

« MS »;

« Etat module ».

### A.2.5.4 Voyant d'état du réseau

#### A.2.5.4.1 Description

Le voyant bicolore (vert/rouge) indique l'état de la liaison de communication.

#### A.2.5.4.2 Etats

Les états du voyant d'état du réseau doivent être conformes au Tableau A.5. Voir le diagramme d'états d'accès à la liaison de la CEI 62026-3:2008, 5.4 pour comparer le voyant d'état du réseau au diagramme d'états d'accès à la liaison.

**Tableau A.5 – Voyant d'état du réseau**

| Etat du voyant | Récapitulatif | Cause |
|---|---|---|
| Fixe inactif | Pas d'alimentation/ hors ligne | L' appareil n'est pas en ligne.<br>– L'appareil n'a pas encore terminé l'essai Dup_MAC_ID.<br>– Impossible de mettre l'appareil sous tension, consulter le voyant d'état du module.<br>– Pas d'alimentation réseau |
| Vert clignotant[a] | En ligne, non connecté | L'appareil est en ligne mais n'a établi aucune connexion.<br>– L'appareil a réussi l'essai Dup_MAC_ID, est en ligne, mais n'a pas établi de connexion avec les autres nœuds.<br>– Pour un appareil de Groupe 2 uniquement, cela signifie que cet appareil n'est pas attribué à un maître.<br>– Pour un appareil UCMM, cela signifie que l'appareil n'a établi aucune connexion |
| Fixe vert | Liaison OK, en ligne, connecté | L'appareil est en ligne et a établi des connexions.<br>– Pour un appareil de Groupe 2 uniquement, cela signifie que cet appareil est attribué à un maître.<br>– Pour un appareil UCMM, cela signifie que l'appareil a établi une ou plusieurs connexions. |
| Rouge clignotant[a] | Délai de connexion | Une ou plusieurs connexions d'E-S sont à l'état de délai d'attente |

| Etat du voyant | Récapitulatif | Cause |
|---|---|---|
| Rouge fixe | Interruption de liaison critique | Echec de l'appareil de communication. L'appareil a détecté une erreur qui l'empêche de communiquer sur le réseau (MAC ID en double ou bus inactif). |
| Rouge/vert clignotant[b] | Echec de la communication et réception d'une demande Identifiy_Comm_Fault (protocole long) | Echec de communication spécifique de l'appareil. L'appareil a détecté une erreur d'accès au réseau et se trouve à l'état de défaut de communication. L'appareil a par la suite reçu et accepté une demande d'identification de communication défaillante (message de protocole long). |

[a] Les vitesses de clignotement du voyant sont indiquées en A.2.2.3.

[b] Les vitesses de clignotement du voyant sont données dans la CEI 62026-3:2008, 5.2.4.5.4.

Tous les états et toutes les couleurs de voyant non défini(e)s dans le Tableau A.6 sont réservés et ne doivent pas être utilisés.

Les appareils de sécurité font l'objet d'exigences supplémentaires (voir la CEI 61784-3-2).

#### A.2.5.4.3 Etiquetage

Il convient d'apposer l'une des étiquettes ci-dessous sur le voyant d'état du réseau:

« NS »;

« Etat du réseau ».

### A.2.5.5 Voyant d'état du module/réseau combiné

#### A.2.5.5.1 Description

Le voyant bicolore (vert/rouge) indique un appareil et un état de communication limités. Le voyant d'état du module/réseau combiné indique si l'appareil est alimenté et fonctionne correctement.

#### A.2.5.5.2 Etats

Les états du voyant d'état du module/réseau combiné doivent être conformes au Tableau A.6. Voir le diagramme d'états d'accès à la liaison de la CEI 62026-3:2008, 5.4 pour comparer le voyant d'état du module/réseau combiné au diagramme d'état d'accès à la liaison.

**Tableau A.6 – Voyant d'état du module/réseau combiné**

| Etat du voyant | Récapitulatif | Cause |
|---|---|---|
| Fixe inactif | Pas d'alimentation/ appareil hors ligne | L' appareil n'est pas en ligne.<br>– L'appareil n'a pas encore terminé l'essai Dup_MAC_ID.<br>– L'appareil n'est probablement pas alimenté. |
| Fixe vert | Appareil opérationnel et en ligne, connecté | L'appareil fonctionne normalement et est en ligne avec des connexions établies.<br>– Pour un appareil de Groupe 2 uniquement, cela signifie que cet appareil est attribué à un maître.<br>– Pour un appareil UCMM, cela signifie que l'appareil a établi une ou plusieurs connexions. |
| Vert clignotant[a] | Appareil opérationnel et en ligne et non connecté ou appareil en ligne et devant être mis en service | L'appareil fonctionne normalement et est en ligne avec aucune connexion établie.<br>– L'appareil a réussi l'essai Dup_MAC_ID, est en ligne, mais n'a pas établi de connexion avec les autres nœuds.<br>– Pour un appareil de Groupe 2 uniquement, cela signifie que cet appareil n'est pas attribué à un maître.<br>– Pour un appareil UCMM, cela signifie que l'appareil n'a établi aucune connexion.<br>– Configuration manquante, incomplète ou incorrecte |
| Rouge clignotant[a] | Défaut mineur et/ou délai de connexion et/ou pas d'alimentation réseau | Une ou plusieurs des conditions ci-dessous:<br>– Panne récupérable.<br>– Une ou plusieurs connexions d'E-S sont à l'état de délai d'attente<br>– Pas d'alimentation réseau |
| Rouge fixe | Défaut critique ou interruption de liaison critique | L'appareil a détecté une panne irrécupérable; peut faire l'objet d'un remplacement.<br>Echec de l'appareil de communication. L'appareil a détecté une erreur qui l'empêche de communiquer sur le réseau (MAC ID en double ou bus inactif). |
| Rouge/vert clignotant[b] | Echec de la communication et réception d'une demande Identifiy_Comm_Fault (protocole long) | Echec de communication spécifique de l'appareil. L'appareil a détecté une erreur d'accès au réseau et se trouve à l'état de défaut de communication. L'appareil a par la suite reçu et accepté une demande d'identification de communication défaillante (message de protocole long). |

[a] Les vitesses de clignotement du voyant sont indiquées en A.2.2.3.

[b] Les vitesses de clignotement du voyant sont données dans la CEI 62026-3:2008, 5.2.4.5.4.

Tous les états et toutes les couleurs de voyant non défini(e)s dans le Tableau A.6 sont réservés et ne doivent pas être utilisés.

Un voyant d'état du module/réseau combiné ne doit pas être utilisé pour les appareils de sécurité.

#### A.2.5.5.3 Etiquetage

Il convient d'apposer l'une des étiquettes ci-dessous sur le voyant d'état du module/réseau combiné:

« MNS »;

« Etat Mod/Rés »;

« Etat du module/réseau ».

### A.2.5.6 Voyant d'état E-S

#### A.2.5.6.1 Description

Le voyant bicolore (vert/rouge) donne des informations relatives aux états des entrées et/ou sorties.

Le voyant d'état E-S a pour objet de signaler à l'utilisateur que l'appareil contrôle des sorties et que des entrées ou des sorties sont actives (sorties actives, productions d'entrée) ou défaillantes. Le voyant a pour objet de refléter le mode/l'état des entrées et des sorties, pas nécessairement la condition active/inactive des points d'E-S eux-mêmes.

#### A.2.5.6.2 Etats

La définition spécifique des couleurs et états du voyant d'état E-S réside dans les définitions individuelles des objets d'application correspondants précisant l'utilisation de ce voyant.

Le Tableau A.7 donne les lignes directrices régissant l'utilisation du voyant d'état E-S.

**Tableau A.7 – Voyant d'état E-S**

| Etat du voyant | Récapitulatif | Cause |
|---|---|---|
| Fixe inactif | Sortie(s) inactive(s)<br>Entrée(s) inactive(s) | Toutes les sorties sont inactives.<br>Toutes les entrées sont inactives. |
| Fixe vert | Sortie(s) active(s)<br>Entrée(s) active(s) | Une ou plusieurs sorties sont actives et sous contrôle, et aucune sortie n'est « défaillante ».<br>Une ou plusieurs entrées sont actives et génèrent des données, et aucune entrée n'est « défaillante ». |
| Vert clignotant | Sortie(s) en veille | Une ou plusieurs sorties sont en veille et aucune sortie n'est active ni « défaillante » |
| Rouge clignotant | Sortie(s) défaillante(s)<br>Entrée(s) défaillante(s) | Une ou plusieurs sorties sont « défaillantes », probablement à l'état d'anomalie.<br>Une ou plusieurs entrées sont « défaillantes », probablement à l'état d'anomalie. |
| Rouge fixe | Sortie(s) forcées<br>Défaut irrécupérable de l'entrée | Une ou plusieurs sorties sont forcées (probablement un défaut irrécupérable).<br>Une ou plusieurs entrées font l''objet d'un défaut irrécupérable |
| NOTE Les vitesses de clignotement du voyant sont spécifiées en A.2.2.3. | | |

Tous les états et toutes les couleurs de voyant non défini(e)s dans le Tableau A.7 sont réservés et ne doivent pas être utilisés.

#### A.2.5.6.3 Etiquetage

Il convient d'apposer l'une des étiquettes ci-dessous sur le voyant d'état E-S:

« E-S »;

« E/S »;

« Etat E-S ».

## A.3 Commutateurs

### A.3.1 Exigences communes du commutateur

#### A.3.1.1 Généralités

Tous les commutateurs doivent se comporter de la même manière pour la même fonction. Par exemple, tous les commutateurs d'adresse réseau des appareils doivent se comporter de la même manière.

Si un appareil est doté de commutateurs, il est recommandé qu'il puisse permettre de déterminer à distance leurs paramètres à l'aide des services de messagerie.

Les appareils de sécurité font l'objet d'exigences supplémentaires (voir la CEI 61784-3-2).

#### A.3.1.2 Remplacement logiciel des commutateurs

Si un commutateur accessible par l'utilisateur (MAC ID ou commutateur de vitesse en bits, par exemple) peut être remplacé par un logiciel, un voyant doit indiquer s'il peut l'être ou pas.

Si un attribut peut être configuré avec un commutateur qui ne peut pas être remplacé par un logiciel, l'appareil doit répondre par l'erreur « Attribute not settable » (l'attribut ne peut pas être défini) à un service qui tente d'affecter une valeur à l'attribut.

#### A.3.1.3 Durées de validité des commutateurs

Les commutateurs doivent se conformer à l'une des règles ci-après:

– le commutateur doit être actif à tout moment;
– le commutateur ne doit être actif qu'à la mise sous tension.

### A.3.2 Exigences du commutateur spécifique au bus de terrain (1)

#### A.3.2.1 Commutateurs d'adresse réseau

##### A.3.2.1.1 Apparence

Si le MAC ID peut être défini à l'aide des commutateurs, ils doivent être indiqués au format décimal. Cela est possible à l'aide d'un commutateur rotatif, à molette, à poussoir ou autre. Le chiffre le plus significatif doit se trouver à gauche ou en haut du chiffre de poids faible.

##### A.3.2.1.2 Lecture du commutateur

Les commutateurs d'adresse réseau ne doivent être lus qu'à la mise sous tension. Si les commutateurs d'adresse réseau sont modifiés après la mise sous tension, un défaut mineur doit être détecté et le voyant d'état du module doit indiquer un défaut non critique (clignotant rouge). Le MAC ID du nœud utilisé pour la couche de liaison de données doit conserver la valeur lue pendant la mise sous tension. Le paramètre en cours des commutateurs d'adresse réseau doit être enregistré dans l'objet ControlNet, de manière à permettre à un autre nœud de diagnostiquer à distance le défaut non critique.

##### A.3.2.1.3 Etiquetage

L'une des étiquettes ci-dessous doit être apposée sur les commutateurs d'adresse réseau:

« NA »;

« adresse »;

« adresse rés».

### A.3.3 Exigences du commutateur spécifique au bus de terrain (2)

Il n'existe aucune exigence relative au commutateur pour CP 2/2.

### A.3.4 Exigences du commutateur spécifique au bus de terrain (3)

#### A.3.4.1 Commutateurs d'adresse réseau

##### A.3.4.1.1 Apparence

Si des commutateurs DIP sont utilisés pour définir le MAC ID, ils doivent se présenter au format binaire. Si des commutateurs rotatifs, à molette ou à poussoir sont utilisés, le format décimal est obligatoire. Le chiffre le plus significatif doit toujours se trouver à gauche ou en haut de l'appareil lorsque l'utilisateur tente de configurer les commutateurs.

##### A.3.4.1.2 Etiquetage

L'une des étiquettes ci-dessous doit être apposée sur les commutateurs d'adresse réseau:

« NA »;

« Adresse nœud ».

#### A.3.4.2 Commutateurs de vitesse en bits

##### A.3.4.2.1 Apparence

Si des commutateurs sont utilisés pour définir la vitesse en bits CP 2/3, ils doivent être codés comme indiqué dans le Tableau A.8.

**Tableau A.8 – Codage du commutateur de vitesse en bits**

| Vitesse en bits | Paramètres du commutateur |
|---|---|
| 125 Kbit/s | 0 |
| 250 Kbit/s | 1 |
| 500 Kbit/s | 2 |

##### A.3.4.2.2 Etiquetage

L'une des étiquettes ci-dessous doit être apposée sur les commutateurs de vitesse en bits:

« DR »;

« Vitesse données ».

# Bibliographie

NOTE Toutes les parties de la série CEI 61158, ainsi que la CEI 61784-1 et la CEI 61784-2 font l'objet d'une maintenance simultanée. Les références croisées à ces documents dans le texte se rapportent donc aux éditions datées figurant dans cette liste de références bibliographiques.

CEI 61158-1:2014, *Réseaux de communication industriels – Partie 1: Spécifications des bus de terrain – Partie 1: Présentation et lignes directrices des séries CEI 61158 et CEI 61784*

CEI 61158-2:2014, *Réseaux de communication industriels – Partie 2: Spécifications des bus de terrain – Partie2: Spécification et définition des services de la couche physique*

CEI 61784-1:2014, *Réseaux de communication industriels – Profils – Partie 1: Profils de bus de terrain*

CEI 61784-2:2014, *Réseaux de communication industriels – Profils – Partie 2: Profils de bus de terrain supplémentaires pour les réseaux en temps réel basés sur l'ISO/CEI 8802-3*

ISO/CEI 8802 (toutes les parties), *Technologies de l'information – Télécommunications et échange d'information entre systèmes – Réseaux locaux et métropolitains – Exigences spécifiques*

ISO/CEI 9314-2, *Systèmes de traitement de l'information – Interface de données distribuées sur fibre (FDDI) – Partie 2: Mécanisme d'accès au support de l'anneau à jeton (MAC)*

ISO/CEI 10731, *Technologies de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base – Conventions pour la définition des services OSI*

ANSI X3.66, *Advanced data communication control procedures (ADCCP)*

ANSI X3.159, *Information Systems – Programming Language C*

ANSI X3J16, *Programming Language C++*

IETF RFC 791, *Internet Protocol*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 793, *Transmission Control Protocol*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1350, *The TFTP Protocol (Revision 2)*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 4541, *Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 4702, *The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option*, disponible à l'adresse <http://www.ietf.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 1: Common Industrial Protocol (CIP™) – Edition 3.13, November 2012*, disponible à l'adresse <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 2: EtherNet/IP™ Adaptation of CIP – Edition 1.14, November 2012*, disponible à l'adresse <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 3: DeviceNet™ Adaptation of CIP – Edition 1.12, November 2011*, disponible à l'adresse <http://www.odva.org>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 4: ControlNet Adaptation of CIP – Edition 1.7, April 2011*, disponible à l'adresse <http://www.odva.org>

_____

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch