IEC 61158-6-23

Edition 1.0 2014-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Industrial communication networks – Fieldbus specifications –
Part 6-23: Application layer protocol specification – Type 23 elements

Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 6-23: Spécification du protocole de la couche application – Éléments de type 23

IEC 61158-6-23:2014-08(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 It also gives information on projects, replaced and withdrawn publications.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 61158-6-23

Edition 1.0 2014-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Industrial communication networks – Fieldbus specifications – Part 6-23: Application layer protocol specification – Type 23 elements

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-23: Spécification du protocole de la couche application – Éléments de type 23

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XH

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-8322-1768-9

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-23: Application layer protocol specification – Type 23 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 61158-6-23 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

FDIS	Report on voting
65C/764/FDIS	65C/774/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 the ISO/IEC Directives, Part 2.

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

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.

0 INTRODUCTION

0.1 General

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 61158-1:2014.

The application protocol provides the application service by making use of the services available from the data-link or other immediately lower 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 application entities (AEs) 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:

– as a guide for implementors and designers;
– for use in the testing and procurement of equipment;
– as part of an agreement for the admittance of systems into the open systems environment;
– 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.

0.2 Patent disclosure

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent concerning Type 23 elements and possibly other types given in 8.1 and 8.2 as follows:

JP 05106658 US 7983177 DE 112006003943.1 KR 10-1029201 TW I338476	[MEC]	Communication management device, communication node, communication system, and data communication method
JP 4503678 DE 112006003895.8 KR 10-1024472 CN 201110218295.6 TW I333356	[MEC]	Communication management device, communication device, and communication method
JP 2010-045463 US 12/774377 DE 112006004225.4 KR 10-1024482 CN 201010148761.3 TW 099112461	[MEC]	Communication node, and token issuing method and token-ring communication method in ring communication system
JP 05127977	[MEC]	Synchronization system, time master nodes, time slave nodes and synchronization method
JP 05106658 US 13/334863 DE 112008004265.9 KR 10-2011-7030535 CN 201210026699.X TW 101108048	[MEC]	Communication management device, communication node, communication system, and data communication method

JP 2011-128274 US 13/325125 DE 112008004268.3 KR 10-2011-7029114 CN 201210127058.3 TW 101102132	[MEC]	Communication management device, communication device, and communication method
JP 05084916 US 13/142244 DE 112008004245.4 KR 10-2011-7014492 CN 200880132546.5 TW 098100145	[MEC]	Communication management device, communication device, and communication method
JP 2011-518195 US 13/377397 DE 112009004913.3 KR 10-2011-7027858 CN 200980159835.9 TW 098119663	[MEC]	Communication managing apparatus, communication nodes, and data communication method
JP 2011-532954	[MEC]	Network performance estimating apparatus, network performance estimating method, network structure recognizing method, communication managing apparatus, and data communication method

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

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

[MEC] Mitsubishi Electric Corporation
Corporate Licensing Division
7-3, Marunouchi 2-chome, Chiyoda-ku,
Tokyo 100-8310, Japan

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 data bases of patents relevant to their standards. Users are encouraged to consult the data bases for the most up to date information concerning patents.

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-23: Application layer protocol specification – Type 23 elements

1 Scope

1.1 General

The Fieldbus Application Layer (FAL) provides user programs with a means to access the fieldbus communication environment. In this respect, the FAL can be viewed as a "window between corresponding application programs."

This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 23 fieldbus. The term "time-critical" is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This standard defines in an abstract way the externally visible behavior provided by the different Types of the fieldbus Application Layer in terms of

a) the abstract syntax defining the application layer protocol data units conveyed between communicating application entities,
b) the transfer syntax defining the application layer protocol data units conveyed between communicating application entities,
c) the application context state machine defining the application service behavior visible between communicating application entities; and
d) the application relationship state machines defining the communication behavior visible between communicating application entities; and.

The purpose of this standard is to define the protocol provided to

a) define the wire-representation of the service primitives defined in IEC 61158-5-23, and
b) define the externally visible behavior associated with their transfer.

This standard specifies the protocol of the IEC fieldbus Application Layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) and the OSI Application Layer Structure (ISO/IEC 9545).

FAL services and protocols are provided by FAL application-entities (AE) contained within the application processes. The FAL AE is composed of a set of object-oriented Application Service Elements (ASEs) and a Layer Management Entity (LME) that manages the AE. The ASEs provide communication services that operate on a set of related application process object (APO) classes. One of the FAL ASEs is a management ASE that provides a common set of services for the management of the instances of FAL classes.

Although these services specify, from the perspective of applications, how request and responses are issued and delivered, they do not include a specification of what the requesting and responding applications are to do with them. That is, the behavioral aspects of the applications are not specified; only a definition of what requests and responses they can send/receive is specified. This permits greater flexibility to the FAL users in standardizing

such object behavior. In addition to these services, some supporting services are also defined in this standard to provide access to the FAL to control certain aspects of its operation.

1.2 Specifications

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-23.

A secondary objective is to provide migration paths from previously-existing industrial communications protocols. It is this latter objective which gives rise to the diversity of protocols standardized in subparts of IEC 61158-6.

1.3 Conformance

This standard does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems.

There is no conformance of equipment to the application layer service definition standard. Instead, conformance is achieved through implementation of this application layer protocol specification.

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 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-5-23, *Industrial communication networks – Fieldbus specifications – Part 5-23: Application layer service definition – Type 23 elements*

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

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

ISO/IEC 8824-1, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*

ISO/IEC 9545, *Information technology – Open Systems Interconnection – Application Layer structure*

ISO/IEC 10731, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

3 Terms, definitions, symbols, abbreviated terms and conventions

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

3.1 Referenced terms and definitions

3.1.1 ISO/IEC 7498-1 terms

For the purposes of this document, the following terms given in ISO/IEC 7498-1 apply:

a) application entity
b) application process
c) application protocol data unit
d) application service element
e) application entity invocation
f) application process invocation
g) application transaction
h) real open system
i) transfer syntax

3.1.2 ISO/IEC 8822 terms

For the purposes of this document, the following terms given in ISO/IEC 8822 apply:

a) abstract syntax
b) presentation context

3.1.3 ISO/IEC 9545 terms

For the purposes of this document, the following terms given in ISO/IEC 9545 apply:

a) application-association
b) application-context
c) application context name
d) application-entity-invocation
e) application-entity-type
f) application-process-invocation
g) application-process-type
h) application-service-element
i) application control service element

3.1.4 ISO/IEC 8824-1 terms

For the purposes of this document, the following terms given in ISO/IEC 8824-1 apply:

a) object identifier
b) type

3.1.5 IEC 61158-1 terms

For the purposes of this document, the following terms given in IEC 61158-1 apply:

a) DLL mapping protocol machine
b) fieldbus application layer
c) FAL service protocol machine
d) protocol data unit

3.2 Type 23 specific terms and definitions

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

3.2.1
cyclic transmission
transmission that is performed periodically used for the link device update

3.2.2
intelligent device station
node capable of performing 1:n bit data and word data cyclic transmission and transient transmission with the master station, and transient transmission with slave stations, excluding remote I/O stations and having client functions and server functions during transient transmission

3.2.3
link bit
link relay bit data that are shared by all the nodes through the cyclic transmission and is used as one bit unit shared memory of the n:n type

3.2.4
link device
link bit, link word, link x and link y or RX, RY, RWr, and RWw

3.2.5
link word
link register two octet unit data that are shared by all the nodes through the cyclic transmission and is used as two octet unit shared memory of the n:n type

3.2.6
link x
link input received bit data that are transmitted from each node through the cyclic transmission and is used as an input shared memory of the 1:n type

3.2.7
link y
link output bit data that are sent to each node through the cyclic transmission and is used as an output shared memory of the 1:n type

3.2.8
local station
node capable of performing n:n bit data and word data cyclic transmission and transient transmission with the master station and other local stations, and transient transmission with slave stations, excluding remote I/O stations and having server functions and client functions during transient transmission

3.2.9
management node
node in which parameters are set

3.2.10
master station
node that has control information (parameters) and manages cyclic transmission

3.2.11
node
element that forms a network and performs data transmission, receiving, and transfer

3.2.12
node-to-node test
physical layer test between two nodes

3.2.13
normal node
node other than a management node

3.2.14
remote device station
node capable of performing 1:n bit data and word data cyclic transmission and transient transmission with the master station, and transient transmission with slave stations, excluding remote I/O stations and having server functions during transient transmission

3.2.15
remote I/O station
node capable of performing 1:n bit data cyclic transmission with the master station

3.2.16
reserve node
node that is not yet connected, but counted in the total node number of the network not performing cyclic transmission, but always regarded as normal from applications

3.2.17
RX
remote input as viewed from the master station with bit data that are periodically updated by cyclic transmission, salve to master, or in local station as viewed from the master station is RY of the local station

3.2.18
RY
remote output as viewed from the master station with bit data that are periodically updated by cyclic transmission, master to salve, or in local station as viewed from the master station is RX of the local station

3.2.19
RWr
remote register (input) as viewed from the master station with word data that are periodically updated by cyclic transmission, slave to master, or in local station as viewed from the master station is RWw of the local station

3.2.20
RWw
remote register (output) as viewed from the master station with word data that are periodically updated by cyclic transmission, master to slave, or in local station as viewed from the master station is RWr of the local station

3.2.21
slave station
node other than the master station

3.2.22
station
node

3.2.23
synchronization manager
node (master station role with one existing per network) that manages synchronization, distributing synchronization timing to other nodes

3.2.24
transient transmission
transmission that is performed upon each request

3.2.25
transient transmission client function
function that issues a transient request

3.2.26
transient transmission server function
function that receives a transient request and issues a response

3.2.27
transmission control manager
node (master station role with one existing per network) that performs token passing management

3.2.28
word
unit representing data, 16 bits in length

3.3 Symbols and abbreviated terms

AE	Application Entity
AL	Application Layer
AP	Application Process
APDU	Application Protocol Data Unit
APO	Application Process Object
AR	Application Relationship
AREP	Application Relationship Endpoint
ASE	Application Service Element
ASN.1	Abstract Syntax Notation 1
CRC	Cyclic Redundancy Check
DLL	Data-link Layer
DMPM	DLL Mapping Protocol Machine
FAL	Fieldbus Application Layer
FSPM	FAL Service Protocol Machine
LB	Link Bit
LSB	Least Significant Bit
LW	Link Word
LX	Link X
LY	Link Y
MSB	Most Significant Bit
OSI	Open Systems Interconnection
PDU	Protocol Data Unit

3.4 Conventions

3.4.1 General concept

The FAL is defined as a set of object-oriented ASEs. Each ASE is specified in a separate subclause. Each ASE specification is composed of three parts: its class definitions, its services, and its protocol specification. The first two are contained in IEC 61158-5-23. The protocol specification for each of the ASEs is defined in this standard.

The class definitions define the attributes of the classes supported by each ASE. The attributes are accessible from instances of the class using the Management ASE services specified in IEC 61158-5-23. The service specification defines the services that are provided by the ASE.

This standard uses the descriptive conventions given in ISO/IEC 10731.

3.4.2 Convention for the encoding of reserved bits and octets

The term "reserved" may be used to describe bits in octets or whole octets. All bits or octets that are reserved should be set to zero at the sending side and shall not be tested at the receiving side except it is explicitly stated or if the reserved bits or octets are checked by a state machine.

The term "reserved" may also be used to indicate that certain values within the range of a parameter are reserved for future extensions. In this case the reserved values should not be used at the sending side and shall not be tested at the receiving side except it is explicitly stated or if the reserved values are check by a state machine.

3.4.3 Conventions for abstract syntax description

This description of FAL Type 23 uses a subset of ASN.1 according to ISO/IEC 8824-1. The following structures are used.

Selective type (CHOICE) – Represents a selection from candidate types

Sequence type (SEQUENCE) – Represents a fixed-order list

as in the following example:

```
DLPDU ::= SEQUENCE {
        preamble        Preamble,
        sfd             SFD,
        destaddr        DestAddr,
        srcaddr         SrcAddr,
        lt              LT,
        dlsdu           FAL-PDU,
        fcs             FCS
}
```

NOTE This example shows that the DLPDU which represents the Ethernet frame is defined as SEQUENCE. The DLPDU consists of Preamble, SFD, DestAddr, SrcAddr, LT, FAL-PDU and FCS.

3.4.4 Conventions for bit description in octets

When identifying each bit in an octet, each bit is identified by a number as shown in Figure 1 and described as Bit n.

MSB LSB

7 6 5 4 3 2 1 0 Bit identification number

Figure 1 – Bit description in octets

When specifying multiple bits sequentially located, the range symbol (..) is used (e.g.: 7..0, specifies bits 7 through 0, inclusive).

When specifying multiple octets, the LSB of the lowest octet is considered 0, and bit identification numbers are assigned in an ascending order.

NOTE For example, when specifying 4 octets, the MSB of the highest octet is Bit 31, the MSB of the second octet is Bit 23, the MSB of the third octet is Bit 15, and the MSB of the lowest octet is Bit 7.

3.4.5 Conventions for state machine descriptions

The state machine description is defined in tabular form as shown in Table 1. The meaning of the elements is shown in Table 2. The conventions used in the state machines are shown in Table 3.

Each row of state table represents a state transition. The first column shows the state transition name or number. The second column shows the current state. The third column shows the events, conditions and actions. The fourth column shows the next state. When an event or condition is fulfilled, the action is performed and the state machine transitions to the next state.

Table 1 – State machine description elements

#	Current state	Event/condition => action	Next state

Table 2 – Description of state machine elements

Heading	Description
#	state transition name or number
Current state	current state
Next state	destination state
Event	description of event
Condition	logical expression representing the condition
=> Action	action performed upon satisfaction of the event or condition

Table 3 – Conventions used in state machines

Notation	Description
=	Substitution of the right side for the left side
==	A logical condition to indicate an item on the left is equal to an item on the right.
!=	A logical condition to indicate an item on the left is not equal to an item on the right.
<	A logical condition to indicate an item on the left is less than the item on the right.
>	A logical condition to indicate an item on the left is greater than the item on the right.
&&	Logical ”AND”

Notation	Description
||	Logical ”OR”
!	Negation operator
+ - * /	Arithmetic operator
;	Breakpoint

4 FAL syntax description

4.1 FALPDU type C abstract syntax

4.1.1 Basic abstract syntax

The definitions of FALPDU are shown below.

```
FAL-PDU ::= CHOICE {
    connect-PDU             [0] Connect-PDU,
    connectAck-PDU          [1] ConnectAck-PDU,
    scan-PDU                [2] Scan-PDU,
    collect-PDU             [3] Collect-PDU,
    select-PDU              [4] Select-PDU,
    launch-PDU              [5] Launch-PDU,
    token-PDU               [6] Token-PDU,
    myStatus-PDU            [7] MyStatus-PDU,
    transient1-PDU          [8] Transient1-PDU,
    dummy-PDU               [9] Dummy-PDU,
    transient2-PDU          [10] Transient2-PDU,
    ntnTest-PDU             [11] NTNTest-PDU,
    cdata-PDU               CData-PDU
}

CData-PDU ::= CHOICE {
    cyclicDataW-PDU         [12] CyclicDataW-PDU,
    cyclicDataB-PDU         [13] CyclicDataB-PDU,
    cyclicDataOut1-PDU      [14] CyclicDataOut1-PDU,
    cyclicDataOut2-PDU      [15] CyclicDataOut2-PDU,
    cyclicDataIn1-PDU       [16] CyclicDataIn1-PDU,
    cyclicDataIn2-PDU       [17] CyclicDataIn2-PDU
}
```

FALARHeader to be used in each PDU are shown as follows.

```
FALARHeader ::= SEQUENCE {
    arFType                 ARFType,
    priority                Priority,
    scanNumber              ScanNumber,
    reserved1               Unsigned8,
    srcNodeNumber           NodeNumber,
    reserved2               Unsigned16,
```

```
        hec                         Hec
    }
```

4.1.2 Connect-PDU

```
    Connect-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        portChoice                  PortChoice,
        padding                     OctetString SIZE(28),
        dcs                         DCS
    }
```

4.1.3 ConnectAck-PDU

```
    ConnectAck-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        portCheckResult             PortCheckResult,
        destPortInfo                DestPortInfo,
        padding                     OctetString SIZE(20),
        dcs                         DCS
    }
```

4.1.4 Scan-PDU

```
    Scan-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        scanState                   ScanState,
        sendTime                    SendTime,
        padding                     OctetString SIZE(20),
        dcs                         DCS
    }
```

4.1.5 Collect-PDU

```
    Collect-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        vendorCode                  VendorCode,
        nodeType                    NodeType,
        netNumber                   NetNumber,
        sendTime                    SendTime,
        loopState                   LoopState,
        parmTypeCyclicStatus        ParmTypeCyclicStatus,
        commonParamId               CommonParamId,
        padding                     OctetString SIZE(8),
        dcs                         DCS
    }

    CommonParamId ::= SEQUENCE {
```

```
        date                    ParamDate,
        timeNodeId              ParamTime,
        checksum                ParamChecksum
    }
```

4.1.6 Select-PDU

```
    Select-PDU ::= SEQUENCE {
        falArHeader             FALARHeader,
        padding                 OctetString SIZE(28),
        dcs                     DCS
    }
```

4.1.7 Launch-PDU

```
    Launch-PDU ::= SEQUENCE {
        falArHeader             FALARHeader,
        padding                 OctetString SIZE(28),
        dcs                     DCS
    }
```

4.1.8 Token-PDU

```
    Token-PDU ::= SEQUENCE {
        falArHeader             FALARHeader,
        padding                 OctetString SIZE(28),
        dcs                     DCS
    }
```

4.1.9 MyStatus-PDU

```
    MyStatus-PDU ::= SEQUENCE {
        falArHeader             FALARHeader,
        reserved1               Unsigned16,
        nodeType                NodeType,
        netNumber               NetNumber,
        reserved2               Unsigned16,
        loopState               LoopState,
        parmTypeCyclicStatus    ParmTypeCyclicStatus,
        commonParamId           CommonParamId,
        inFarNodeMACAddr        MACAddress,
        inFarNodeNumber         NodeNumber,
        reserved3               Unsigned8,
        outFarNodeMACAddr       MACAddress,
        outFarNodeNumber        NodeNumber,
        reserved4               Unsigned8,
        opState                 OpState,
        errorState              ErrorState,
```

```
        errorCode                   ErrorCode,
        vendorCode                  VendorCode,
        deviceType                  DeviceType,
        unitTypeName                UnitTypeName,
        unitTypeCode                UnitTypeCode,
        reserved5                   Unsigned16,
        nodeInfo                    NodeInfo,
        dcs                         DCS
    }
```

4.1.10 Transient1-PDU

```
    Transient1-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        destinationGroup            DestinationGroup,
        seqNumber                   SeqNumber,
        dataId                      TraDataId,
        wholeDataSize               TraWholeDataSize,
        offsetAddr                  TraOffsetAddr,
        dataSize                    TraDataSize,
        dataType                    TraDataType,
        data                        TraData,
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.1.11 Dummy-PDU

```
    Dummy-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        dummyData                   OctetString SIZE(28..1482),
        dcs                         DCS
    }
```

4.1.12 Transient2-PDU

```
    Transient2-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        l                           Length,
        gcnt                        GateCount,
        typeSeqF                    TypeSeqF,
        fno                         FrameSequence,
        dt                          DataFrameType,
        da                          TraDstAddr,
        sa                          TraSrcAddr,
        dat                         TraDstAppType,
        sat                         TraSrcAppType,
        dmf                         TraDstModuleFlag,
```

```
        smf                 TraSrcModuleFlag,
        dna                 TraDstNetAddr,
        ds                  TraDstStaNo,
        did                 TraDstID,
        sna                 TraSrcNetAddr,
        ss                  TraSrcStaNo,
        sid                 TraSrcID,
        l1                  TraCmdLen,
        ct                  TraCmdType,
        rsv                 Unsigned8,
        aps                 TraAppSeq,
        data                [0] TraData OPTIONAL,
        evenPadding         [1] Unsigned8 OPTIONAL,
        dcs                 DCS
}
```

4.1.13 NTNTest-PDU

```
NTNTest-PDU ::= SEQUENCE {
    falArHeader         FALARHeader,
    ntnTestData         NTNTestData,
    dcs                 DCS
}
```

4.1.14 CyclicDataW-PDU

```
CyclicDataW-PDU ::= SEQUENCE {
    falArHeader         FALARHeader,
    seqNumber           SeqNumber,
    byteValidity        ByteValidity,
    dataSize            CycDataSize,
    offsetAddr          CycOffsetAddr,
    exSeqNumber         CycExSeqNumber,
    reserved            Unsigned16,
    wData               CycWData,
    evenPadding         [0] Unsigned8 OPTIONAL,
    dcs                 DCS
}
```

4.1.15 CyclicDataB-PDU

```
CyclicDataB-PDU ::= SEQUENCE {
    falArHeader         FALARHeader,
    seqNumber           SeqNumber,
    byteValidity        ByteValidity,
    dataSize            CycDataSize,
    offsetAddr          CycOffsetAddr,
    reserved1           Unsigned16,
```

```
        reserved2                   Unsigned16,
        bData                       CycBData,
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.1.16 CyclicDataOut1-PDU

```
    CyclicDataOut1-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        seqNumber                   SeqNumber,
        byteValidity                ByteValidity,
        dataSize                    CycDataSize,
        offsetAddr                  CycOffsetAddr,
        reserved1                   Unsigned16,
        reserved2                   Unsigned16,
        out1Data                    CycOut1Data,
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.1.17 CyclicDataOut2-PDU

```
    CyclicDataOut2-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        seqNumber                   SeqNumber,
        byteValidity                ByteValidity,
        dataSize                    CycDataSize,
        offsetAddr                  CycOffsetAddr,
        reserved1                   Unsigned16,
        reserved2                   Unsigned16,
        out2Data                    CycOut2Data,
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.1.18 CyclicDataIn1-PDU

```
    CyclicDataIn1-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        seqNumber                   SeqNumber,
        byteValidity                ByteValidity,
        dataSize                    CycDataSize,
        offsetAddr                  CycOffsetAddr,
        reserved1                   Unsigned16,
        reserved2                   Unsigned16,
        in1Data                     CycIn1Data,
        evenPadding                 [0] Unsigned8 OPTIONAL,
```

```
        dcs                         DCS
    }
```

4.1.19 CyclicDataIn2-PDU

```
    CyclicDataIn2-PDU ::= SEQUENCE {
        falArHeader                 FALARHeader,
        seqNumber                   SeqNumber,
        byteValidity                ByteValidity,
        dataSize                    CycDataSize,
        offsetAddr                  CycOffsetAddr,
        reserved1                   Unsigned16,
        reserved2                   Unsigned16,
        in2Data                     CycIn2Data,
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.2 FALPDU type F abstract syntax

4.2.1 Basic abstract syntax

The definitions of FALPDU are shown below.

```
    FAL-PDU ::= CHOICE {
        f-channelControl-PDU        F-ChannelControl-PDU,
        f-sync-PDU                  F-Sync-PDU,
        f-cyclicData-PDU            F-CData-PDU,
        f-transientData-PDU         F-TraData-PDU
    }

    F-ChannelControl-PDU ::= CHOICE {
        persuasion-PDU              [16] Persuasion-PDU,
        testData-PDU                [17] TestData-PDU,
        testDataAck-PDU             [18] TestDataAck-PDU,
        setup-PDU                   [19] Setup-PDU,
        setupAck-PDU                [20] SetupAck-PDU,
        token-PDU                   [21] F-Token-PDU,
        myStatus-PDU                [32] F-MyStatus-PDU
    }

    F-Sync-PDU ::= CHOICE {
        measure-PDU                 [50] F-Measure-PDU,
        measureAck-PDU              [51] F-Measure-PDU,
        offset-PDU                  [52] F-Offset-PDU,
        update-PDU                  [53] F-Update-PDU
    }
```

```
F-CData-PDU ::= CHOICE {
    cyclicDataRWw-PDU          [130] F-CyclicData-PDU,
    cyclicDataRY-PDU           [131] F-CyclicData-PDU,
    cyclicDataRWr-PDU          [132] F-CyclicData-PDU,
    cyclicDataRX-PDU           [133] F-CyclicData-PDU
}

F-TraData-PDU ::= CHOICE {
    transient1-PDU             [34] Transient1-PDU,
    transientAck-PDU           [35] TransientAck-PDU,
    transient2-PDU             [37] Transient2-PDU,
    paramCheck-PDU             [40] ParamCheck-PDU,
    parameter-PDU              [41] Parameter-PDU,
    timer-PDU                  [44] Timer-PDU
}
```

The following shows the FALARHeader used in each PDU.

```
FALAR-FHeader ::= SEQUENCE {
    arFType                    ARFType,
    dataType                   DataType,
    varField                   CHOICE {
        vField0 [0] SEQUENCE {
            persPriority       PersPriority,
            nodeType           NodeType
        },
        vField1 [1] SEQUENCE {
            reserved1          OCTET STRING (SIZE (4))
        },
        vField2 [2] SEQUENCE {
            nodeId             NodeId,
            reserved2          OCTET STRING (SIZE (2))
        },
        vField3 [3] SEQUENCE {
            nodeId             NodeId,
            syncFlag           SyncFlag,
            nodeType           NodeType
        },
        vField4 [4] SEQUENCE {
            nodeId             NodeId,
            connectionInfo     ConnectionInfo,
            reserved4          OCTET STRING (SIZE (1))
        }
    },
    srcNodeNumber              NodeNumber,
    protocolVerType            ProtocolVerType,
    reserved                   OCTET STRING (SIZE (1)),
```

```
        hec                         Hec
    }
```

4.2.2 Persuasion-PDU

```
    Persuasion-PDU ::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        reserved1                   OCTET STRING (SIZE (1)),
        myPorts                     Unsigned8,
        vendorCode                  VendorCode,
        modelCode                   ModelCode,
        reserved2                   OCTET STRING (SIZE (20)),
        dcs                         DCS
    }
```

4.2.3 TestData-PDU

```
    TestData-PDU ::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        tmMacAddr                   MACAddress,
        srcPort                     PortNumber,
        reserved                    OCTET STRING (SIZE (21)),
        dcs                         DCS
    }
```

4.2.4 TestDataAck-PDU

```
    TestDataAck-PDU ::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        tdSrcMacAddr                MACAddress,
        tdSrcPort                   PortNumber,
        tdRcvPort                   PortNumber,
        reserved1                   OCTET STRING (SIZE (1)),
        myPorts                     Unsigned8,
        tokenKeepTime               Unsigned16,
        reserved2                   OCTET STRING (SIZE (4)),
        myConnectStatus             SEQUENCE {
            port2port1              PortStatus,
            port4port3              PortStatus,
            port6port5              PortStatus,
            port8port7              PortStatus,
            port10port9             PortStatus,
            port12port11            PortStatus,
            port14port13            PortStatus,
            port16port15            PortStatus,
            port18port17            PortStatus,
            port20port19            PortStatus,
            port22port21            PortStatus,
```

```
        port24port23                PortStatus
    },
    dcs                             DCS
}
```

4.2.5 Setup-PDU

```
Setup-PDU ::= SEQUENCE {
    falArHeader                     FALAR-FHeader,
    tokenDstMacAddr                 MACAddress,
    reserved1                       OCTET STRING (SIZE (2)),
    leaveTimerValue                 LeaveTimer,
    portUsage                       PortUsage,
    reserved2                       OCTET STRING (SIZE (1)),
    netBehaviour                    NetworkBehaviour,
    reserved3                       OCTET STRING (SIZE (12)),
    dcs                             DCS
}

NetworkBehaviour ::= SEQUENCE {
    multipleTranmit                 Unsigned8,
    frameInterval                   Unsigned8,
    reserved                        OCTET STRING (SIZE (1)),
    multipleTokens                  Unsigned8
}
```

4.2.6 SetupAck-PDU

```
SetupAck-PDU ::= SEQUENCE {
    falArHeader                     FALAR-FHeader,
    slaveNodeInfo                   SlaveNodeInfo,
    fwVersion                       Version,
    deviceType                      DeviceType,
    reserved1                       OCTET STRING (SIZE (2)),
    vendorCode                      VendorCode,
    modelCode                       ModelCode,
    rySize                          Unsigned16,
    rwwSize                         Unsigned16,
    rxSize                          Unsigned16,
    rwrSize                         Unsigned16,
    reserved2                       OCTET STRING (SIZE (2)),
    availableFuncs                  AvailableFuncs,
    reserved3                       OCTET STRING (SIZE (5)),
    dcs                             DCS
}
```

4.2.7 F-Token-PDU

```
F-Token-PDU ::= SEQUENCE {
    falArHeader             FALAR-FHeader,
    tokenDstMacAddr         MACAddress,
    tokenSeqNumber          Unsigned8,
    reserved1               OCTET STRING (SIZE (1)),
    tokenHopCounter         Unsigned16,
    traAvailHopCounter      Unsigned16,
    traLastHopCounter       Unsigned16,
    traAllows               Unsigned8,
    reserved2               OCTET STRING (SIZE (13)),
    dcs                     DCS
}
```

4.2.8 F-MyStatus-PDU

```
F-MyStatus-PDU ::= SEQUENCE {
    falArHeader             FALAR-FHeader,
    seqNumber               SeqNumber,
    netNumber               NetNumber,
    masterCmd               Unsigned16,
    cyclicStatus            Unsigned16,
    nodeStatus              Unsigned16,
    errorCode               ErrorCode,
    portStatus              SEQUENCE {lower PortStatus,
                            upper PortStatus },
    portStatistics          SEQUENCE {lower PortStatistics,
                            upper PortStatistics },
    portIndex               Unsigned8,
    reserved                OCTET STRING (SIZE (3)),
    cyclicSeqNumber         Unsigned8,
    addrTableDistResult     Unsigned8,
    slaveSpfEventInfo1      Unsigned8,
    slaveSpfEventInfo2      Unsigned16,
    vendorSpfNodeInfo       OCTET STRING (SIZE (4)),
    dcs                     DCS
}
```

4.2.9 Measure-PDU

```
F-Measure-PDU ::= SEQUENCE {
    falArHeader             FALAR-FHeader,
    reserved                OCTET STRING (SIZE (28)),
    dcs                     DCS
}
```

4.2.10 F-Offset-PDU

```
F-Offset-PDU ::= SEQUENCE {
```

```
        falArHeader                 FALAR-FHeader,
        reserved                    OCTET STRING (SIZE (8)),
        syncOffset                  SyncOffset,
        reserved2                   OCTET STRING (SIZE (16)),
        dcs                         DCS
}
```

4.2.11 F-Update-PDU

```
F-Update-PDU ::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        reserved                    OCTET STRING (SIZE (8)),
        syncOffset                  SyncOffset,
        reserved2                   OCTET STRING (SIZE (16)),
        dcs                         DCS
}
```

4.2.12 F-CyclicData-PDU

```
F-CyclicData-PDU ::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        seqNumber                   SeqNumber,
        bothEndsValidity            BothEndsValidity,
        cycDataSize                 Unsigned16,
        offsetAddr                  CycOffsetAddr,
        reserved                    OCTET STRING (SIZE (4)),
        cycData                     CycData,
        dcs                         DCS
}
```

4.2.13 Transient1-PDU

```
Transient1-PDU ::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        traMsgHeader                TraMsgHeader,
        data                        OCTET STRING (SIZE
                                    (12..1466)),
        dcs                         DCS
}

TraMsgHeader ::= SEQUENCE {
        reserved                    OCTET STRING (SIZE (4)),
        seqNumber                   SeqNumber,
        dataId                      TraDataId,
        wholeDataSize               TraWholeDataSize,
        offsetAddr                  TraOffsetAddr,
        dataSize                    TraDataSize,
        dataSubType                 TraDataSubType
```

```
}

FieldSpecificTransient ::= SEQUENCE {
    opHeader                    TraMsgCmdExHeader,
    fSTraData                   CHOICE {
        nodeInfoDist            [721] TraSysNodeInfoDist,
        statisticsGet           [723] TraSysStatisticsGet,
        nodeInfoDetailGet       [724] TraSysNodeInfoDetailGet,
        ...
    }
}

TraMsgCmdExHeader ::= SEQUENCE {
    command                     TraCommand,
    subCommand                  TraSubCommand,
    rtn                         CHOICE {
        reserved                [0] OCTET STRING (SIZE (2)),
        value                   [1] Unsigned16
    },
    reserved1                   OCTET STRING (SIZE (1)),
    destNetNumber               NetNumber,
    destNodeNumber              NodeNumber,
    reserved2                   OCTET STRING (SIZE (5)),
    srcNetNumber                NetNumber,
    srcNodeNumber               NodeNumber,
    reserved3                   OCTET STRING (SIZE (4))
}

TraSysNodeInfoDist ::= SEQUENCE {
    seqNumber                   SeqNumber,
    masterNetNumber             NetNumber,
    masterDeviceType            DeviceType,
    masterModelCode             ModelCode,
    masterVendorCode            VendorCode,
    masterNodeType              NodeType,
    Reserved1                   OCTET STRING (SIZE (1)),
    masterMacAddress            MACAddress
    Reserved2                   OCTET STRING (SIZE (2)),
    dataNum                     Unsigned32,
    messages                    SEQUENCE OF
                                NodeInfoMessage
}

NodeInfoMessage ::= SEQUENCE {
    nodeNumber                  NodeNumber,
    reserved1                   OCTET STRING (SIZE (1)),
    availableFuncs              AvailableFuncs,
```

```
        reserved2                   OCTET STRING (SIZE (1)),
        netNumber                   NetNumber,
        deviceType                  DeviceType,
        modelCode                   ModelCode,
        vendorCode                  VendorCode,
        nodeType                    NodeType,
        reserved3                   OCTET STRING (SIZE (1)),
        macAddress                  MACAddress,
        reserved4                   OCTET STRING (SIZE (2))
}

TraSysStatisticsGet ::= CHOICE {
        statGetRequest              [0] SEQUENCE {
        },
        statGetResponse             [1] SEQUENCE {
            port1Mib1               Unsigned32,
            port1Mib2               Unsigned32,
            port1Mib3               Unsigned32,
            port1Mib4               Unsigned32,
            port1Mib5               Unsigned32,
            port1Mib6               Unsigned32,
            port1Mib7               Unsigned32,
            reserved                OCTET STRING (SIZE (4)),
            port2Mib1               Unsigned32,
            port2Mib2               Unsigned32,
            port2Mib3               Unsigned32,
            port2Mib4               Unsigned32,
            port2Mib5               Unsigned32,
            port2Mib6               Unsigned32,
            port2Mib7               Unsigned32,
            healthStatusNum         Unsigned32,
            healthStatus            SEQUENCE SIZE (0..128) OF
                                    Unsigned32
        }
}

TraSysNodeInfoDetailGet ::= CHOICE {
        nodeInfoDetailGetRequest    [0] SEQUENCE {
        },
        nodeInfoDetailGetResponse   [1] SEQUENCE {
            rySize                  Unsigned16,
            rwwSize                 Unsigned16,
            rxSize                  Unsigned16,
            rwrSize                 Unsigned16,
            reserved1               OCTET STRING (SIZE (1)),
            ports                   Unsigned8,
            tokenKeepTime           Unsigned16,
```

```
        netBehaviour                NetworkBehaviour,
        nodeInfo                    SlaveNodeInfo,
        fwVersion                   Version,
        deviceType                  DeviceType,
        modelCode                   ModelCode,
        vendorCode                  VendorCode,
        reserved2                   OCTET STRING (SIZE (2)),
        modelName                   OCTET STRING (SIZE (20)),
        vendorName                  OCTET STRING (SIZE (32)),
        contInfo                    Unsigned8,
        contFwVersion               Version,
        contDeviceType              DeviceType,
        contModelCode               ModelCode,
        contVendorCode              VendorCode,
        reserved3                   OCTET STRING (SIZE (2)),
        contModelName               OCTET STRING (SIZE (20)),
        contVendorName              OCTET STRING (SIZE (32)),
        contVendorSpecificInfo      OCTET STRING (SIZE (4))
    }
}
```

4.2.14 TransientAck-PDU

```
TransientAck-PDU ::= SEQUENCE {
    falArHeader         FALAR-FHeader,
    acks                Unsigned32,
    ackData             SEQUENCE OF TraAckData,
    dcs                 DCS
}
```

4.2.15 Transient2-PDU

```
Transient2-PDU ::= SEQUENCE {
    falArHeader         FALAR-FHeader,
    l                   TraLength,
    reserved            OCTET STRING (SIZE (1)),
    tp                  TraType,
    fno                 TraFrameSequence,
    dt                  TraDataFrameType,
    da                  TraDstAddr,
    sa                  TraSrcAddr,
    dat                 TraDstAppType,
    sat                 TraSrcAppType,
    dmf                 TraDstModuleFlag,
    smf                 TraSrcModuleFlag,
    dna                 TraDstNetAddr,
    ds                  TraDstStaNo,
```

```
        did                         TraDstID,
        sna                         TraSrcNetAddr,
        ss                          TraSrcStaNo,
        sid                         TraSrcID,
        l1                          TraCmdLen,
        ct                          TraCmdType,
        dno                         TraDataNo,
        aps                         TraAppSeq,
        rsts                        TraReturnStatus,
        data                        Tra2Data,
        dcs                         DCS
}
```

4.2.16 ParamCheck-PDU

```
ParamCheck-PDU ::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        reserved1                   OCTET STRING (SIZE (4)),
        paramId                     CommonParamId,
        reserved2                   OCTET STRING (SIZE (12)),
        dcs                         DCS
}

CommonParamId ::= SEQUENCE {
        date                        ParamDate,
        timeNodeId                  ParamTime,
        checksum                    ParamChecksum
}
```

4.2.17 Parameter-PDU

```
Parameter-PDU ::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        paramSetFlag                ParamFlag,
        addressOrder                AddressOrder,
        cmdOrder                    CmdOrder,
        cyclicParameter             CyclicParameter,
        dcs                         DCS
}

AddressOrder ::= SEQUENCE {
        assignedNetNumber           NetNumber,
        assignedNodeNumber          NodeNumber
}

CmdOrder ::= SEQUENCE {
        cmd                         Unsigned24,
```

```
        nodeType                    NodeType
    }

    CyclicParameter ::= SEQUENCE {
        paramId                     CommonParamId,
        reserved1                   OCTET STRING (SIZE (2)),
        masterStatus                Unsigned16,
        rySeqNumber                 SeqNumber,
        ryBothEndsValidity          BothEndsValidity,
        ryDataSize                  Unsigned16,
        ryOffset                    Unsigned16,
        reserved2                   OCTET STRING (SIZE (2)),
        rwwSeqNumber                SeqNumber,
        reserved3                   OCTET STRING (SIZE (1)),
        rwwDataSize                 Unsigned16,
        rwwOffset                   Unsigned16,
        reserved4                   OCTET STRING (SIZE (3)),
        rxBothEndsValidity          BothEndsValidity,
        rxDataSize                  Unsigned16,
        rxOffset                    Unsigned32,
        reserved5                   OCTET STRING (SIZE (2)),
        rwrDataSize                 Unsigned16,
        rwrOffset                   Unsigned32,
        reserved6                   OCTET STRING (SIZE (4)),
        masterWatchTimer            Unsigned16,
        reserved7                   OCTET STRING (SIZE (3)),
        cmRyBothEndsValidity        BothEndsValidity,
        cmRyDataSize                Unsigned16,
        cmRyOffset                  Unsigned32,
        reserved8                   OCTET STRING (SIZE (2)),
        cmRwwDataSize               Unsigned16,
        cmRwwOffset                 Unsigned32,
        reserved9                   OCTET STRING (SIZE (1)),
        cmRxBothEndsValidity        BothEndsValidity,
        cmRxDataSize                Unsigned16,
        cmRxOffset                  Unsigned32,
        reserved10                  OCTET STRING (SIZE (2)),
        cmRwrDataSize               Unsigned16,
        cmRwrOffset                 Unsigned32
    }
```

4.2.18 Timer-PDU

```
    Timer-PDU ::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        time                        Timer,
        reserved                    OCTET STRING (SIZE (22)),
```

```
        dcs                                DCS
    }
```

4.3 Data type assignments for type C

Data types used in FALPDU type C abstract syntax are shown as follows.

```
ARFType ::= Unsigned8
DCS ::= Unsigned8
Priority ::= Unsigned8
ScanNumber ::= Unsigned24
NodeNumber ::= Unsigned16
Hec ::= Unsigned32
PortChoice ::= Unsigned32
PortCheckResult ::= Unsigned32
DestPortInfo ::= Unsigned32
ScanState ::= Unsigned32
SendTime ::= Unsigned16
VendorCode ::= Unsigned16
NodeType ::= Unsigned8
NetNumber ::= Unsigned8
LoopState ::= Unsigned8
ParmTypeCyclicStatus ::= Unsigned8
ParamDate ::= Unsigned32
ParamTime ::= Unsigned32
ParamChecksum ::= Unsigned32
OpState ::= Unsigned16
ErrorState ::= Unsigned16
ErrorCode ::= Unsigned16
DeviceType ::= Unsigned16
UnitTypeName ::= VisibleString SIZE(20)
UnitTypeCode ::= Unsigned16
NodeInfo ::= OctetString SIZE(96)
DestinationGroup ::= Unsigned32
SeqNumber ::= Unsigned8
TraDataId ::= Unsigned8
TraWholeDataSize ::= Unsigned16
TraOffsetAddr ::= Unsigned32
TraDataSize ::= Unsigned16
TraDataType ::= Unsigned16
TraData ::= LOctetString SIZE(12..1466)
Length ::= Unsigned16
GateCount ::= Unsigned8
TypeSeqF ::= Unsigned8
FrameSequence ::= Unsigned8
DataFrameType ::= Unsigned8
TraDstAddr ::= Unsigned8
TraSrcAddr ::= Unsigned8
TraDstAppType ::= Unsigned8
TraSrcAppType ::= Unsigned8
TraDstModuleFlag ::= Unsigned8
TraSrcModuleFlag ::= Unsigned8
TraDstNetAddr ::= Unsigned8
TraDstStaNo ::= Unsigned8
TraDstID ::= Unsigned16
TraSrcNetAddr ::= Unsigned8
TraSrcStaNo ::= Unsigned8
TraSrcID ::= Unsigned16
TraCmdLen ::= Unsigned16
TraCmdType ::= Unsigned8
TraAppSeq ::= Unsigned16
Tra2Data ::= LOctetString SIZE(12..1466)
NTNTestData ::= OctetString SIZE(28..1480)
ByteValidity ::= Unsigned8
CycDataSize ::= Unsigned16
CycOffsetAddr ::= Unsigned32
CycExSeqNumber ::= Unsigned16
```

```
CycWData ::= LOctetString SIZE(16..1468)
CycBData ::= LOctetString SIZE(16..1468)
CycOut1Data ::= LOctetString SIZE(16..1468)
CycOut2Data ::= LOctetString SIZE(16..1468)
CycIn1Data ::= LOctetString SIZE(16..1024)
CycIn2Data ::= LOctetString SIZE(16..1024)
```

4.4 Data type assignments for type F

Data types used in FALPDU type F abstract syntax are shown as follows.

```
DCS ::= Unsigned32
ARFType ::= Unsigned8
DataType ::= Unsigned8
NodeNumber ::= Unsigned16
ProtocolVerType ::= Unsigned8
Hec ::= Unsigned32
PersPriority ::= Unsigned24
NodeType ::= Unsigned8
NodeId ::= Unsigned16
ConnectionInfo ::= Unsigned8
VendorCode ::= Unsigned16
ModelCode ::= Unsigned32
PortNumber ::= Unsigned8
TraControl ::= Unsigned8
PortStatus ::= Unsigned8
LeaveTimer ::= Unsigned16
PortUsage ::= Unsigned8
SlaveNodeInfo ::= Unsigned8
Version ::= Unsigned8
DeviceType ::= Unsigned16
AvailableFuncs ::= Unsigned8
SeqNumber ::= Unsigned8
NetNumber ::= Unsigned8
PortStatistics ::= Unsigned8
ErrorCode ::= Unsigned32
ParamFlag ::= Unsigned8
ParamDate ::= Unsigned32
ParamTime ::= Unsigned32
ParamChecksum ::= Unsigned32
Timer ::= Unsigned48
TraDataSubType ::= Unsigned16
TraDataId ::= Unsigned8
TraReturnValue ::= Unsigned16
TraWholeDataSize ::= Unsigned16
TraOffsetAddr ::= Unsigned32
TraDataSize ::= Unsigned16
TraCommand ::= Unsigned8
TraSubCommand ::= Unsigned8
TraLength ::= Unsigned16
TraType ::= Unsigned8
TraFrameSequence ::= Unsigned8
TraDataFrameType ::= Unsigned8
TraDstAddr ::= Unsigned8
TraSrcAddr ::= Unsigned8
TraDstAppType ::= Unsigned8
TraSrcAppType ::= Unsigned8
TraDstModuleFlag ::= Unsigned8
TraSrcModuleFlag ::= Unsigned8
TraDstNetAddr ::= Unsigned8
TraDstStaNo ::= Unsigned8
TraDstID ::= Unsigned16
TraSrcNetAddr ::= Unsigned8
TraSrcStaNo ::= Unsigned8
TraSrcID ::= Unsigned16
TraCmdLen ::= Unsigned16
TraCmdType ::= Unsigned8
TraDataNo ::= Unsigned8
```

```
TraAppSeq ::= Unsigned16
TraReturnStatus ::= Unsigned16
Tra2Data ::= LOctetString (SIZE(0..960))
BothEndsValidity ::= Unsigned8
CycOffsetAddr ::= Unsigned32
CycData ::= LOctetString
OctetString ::= OCTET STRING
BitString8 ::= OCTET STRING (SIZE (1))
BitString16 ::= OCTET STRING (SIZE (2))
BitString32 ::= OCTET STRING (SIZE (3))
Unsigned8 ::= OCTET STRING (SIZE (1))
Unsigned16 ::= OCTET STRING (SIZE (2))
Unsigned24 ::= OCTET STRING (SIZE (3))
Unsigned32 ::= OCTET STRING (SIZE (4))
Unsigned48 ::= OCTET STRING (SIZE (6))
MACAddress ::= OCTET STRING (SIZE (6))
LOctetString ::= OCTET STRING
SyncOffset ::= Unsigned32
SyncFlag ::= Unsigned8
```

5 FAL transfer syntax

5.1 Encoding rules

5.1.1 Unsigned encoding

The fixed-length, unsigned values Unsigned8, Unsigned16, Unsigned24, and Unsigned32 are encoded as unsigned integers of one octet, two octets, three octets, and four octets in length, respectively. Unsigned16, Unsigned24, and Unsigned32 are encoded as big endians, where the most significant octet is regarded as the first octet, the next octet is regarded as the second octet, and the least significant octet is regarded as the last octet.

5.1.2 Octet string encoding

OctetString which has variable length of octets is encoded octet by octet, in sequential order.

5.1.3 SEQUENCE encoding

SEQUENCE (and SEQUENCE OF) encoding is performed in sequence, starting from the initial element. The identifiers and length used in ASN.1 are not used.

5.1.4 LOctetString encoding

LOctetString has variable length and is encoded as little endian, where the lowest, or least significant, octet is ordered as the first octet, and follow in sequential order until the highest, or most significant, octet is ordered last.

5.2 FALPDU type C elements encoding

5.2.1 FALARHeader

5.2.1.1 arFType

This field shows the PDU types described in Table 4.

Table 4 – afFType

Value	Description
0x00	Connect-PDU
0x01	ConnectAck-PDU

Value	Description
0x02	Scan-PDU
0x03	Collect-PDU
0x04	Select-PDU
0x05	Launch-PDU
0x06	Token-PDU
0x07..0x1F	Reserved
0x20	MyStatus-PDU
0x22	Transient1-PDU
0x23	Reserved
0x24	Dummy-PDU
0x25	Transient2-PDU
0x26..0x2E	Reserved
0x2F	NTNTest-PDU
0x30..0x7F	Reserved
0x80	CyclicDataW-PDU
0x81	CyclicDataB-PDU
0x82..0x8B	Reserved
0x8C	CyclicDataOut1-PDU
0x8D	CyclicDataOut2-PDU
0x8E	CyclicDataIn1-PDU
0x8F	CyclicDataIn2-PDU

5.2.1.2 priority

This field shows priorities described in Table 5.

Table 5 – priority

Value	Description
0x00	Select-PDU, Launch-PDU, Token-PDU, Dummy-PDU, CyclicDataW-PDU, CyclicDataB-PDU, CyclicDataOut1-PDU, CyclicDataOut2-PDU, CyclicDataIn1-PDU, CyclicDataIn2-PDU
0x01	Collect-PDU, Connect-PDU, ConnectAck-PDU, Scan-PDU
0x02	MyStatus-PDU
0x03	Transient1-PDU

Value	Description
0x04	Transient2-PDU, NTNTest-PDU

5.2.1.3 scanNumber

This field contains a frame identifier used in Scan-PDU. The number is incremented each time when sending Scan-PDU. The default value of 0x000000 is used in all other PDUs.

5.2.1.4 reserved1

This field is reserved for future use. The value is 0x00.

5.2.1.5 srcNodeNumber

This field contains the identifier number of the source node.

5.2.1.6 reserved2

This field is reserved for future use. The value is 0x0000.

5.2.1.7 hec

This field is an error checking code from DestAddr of DLPDU to reserved2 of FALARHeader. DLPDU definitions are as follows.

```
DLPDU ::= SEQUENCE {
    preamble        Preamble,
    sfd             SFD,
    destaddr        DestAddr,
    srcaddr         SrcAddr,
    lt              LT,
    dlsdu           FAL-PDU,
    fcs             FCS
}

Preamble ::= OctetString SIZE(7)
SFD ::= OctetString SIZE(1)
DestAddr ::= MACAddress
SrcAddr ::= MACAddress
LT ::= Unsigned16
FCS ::= OctetString SIZE(4)
```

The generating polynomial is $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$.

5.2.2 Connect-PDU

5.2.2.1 falArHeader

Refer to 5.2.1.

5.2.2.2 portChoice

This field shows the port types as described in Table 6.

Table 6 – portChoice

Value	Description
0x0	“In” side
0x1	“Out” side
0xF	“Out” side, during Test between nodes

5.2.2.3 padding

This field is padding. The value is 0x00.

5.2.2.4 dcs

This field is an error checking code from DestAddr of DLPDU. For DLPDU definitions, refer to 5.2.1.7.

The generating polynomial is $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$.

5.2.3 ConnectAck-PDU

5.2.3.1 falArHeader

Refer to 5.2.1.

5.2.3.2 portCheckResult

This field shows the check results of In-Out connection The values are specified in Table 7.

Table 7 – portCheckResult

Value	Description
0x0	OK
0x1	NG
0xE	Node-to-node test OK
0xF	Node-to-node test NG

5.2.3.3 dstPortInfo

This field shows the destination port types. The values are specified in Table 8.

Table 8 – dstPortInfo

Value	Description
0	“In” side
1	“Out” side

5.2.3.4 padding

This field is padding. The value is 0x00.

5.2.3.5 dcs

Refer to 5.2.2.4.

5.2.4 Scan-PDU

5.2.4.1 falArHeader

Refer to 5.2.1.

5.2.4.2 scanState

This field shows the status of transmission path. The values are specified in Table 9.

Table 9 – scanState

Value	Description
0	Through status
1	Loopback status of “In” side
2	Loopback status of “Out” side

5.2.4.3 sendTime

This field indicates the sent time.

5.2.4.4 padding

This field is a padding. The value is 0x00.

5.2.4.5 dcs

Refer to 5.2.2.4.

5.2.5 Collect-PDU

5.2.5.1 falArHeader

Refer to 5.2.1.

5.2.5.2 vendorCode

This field shows the vendor code. For vender codes, refer to implementation rules.

5.2.5.3 nodeType

This field shows the node types as described in Table 10.

Table 10 – nodeType

Value	Description
0x00	Management node
0x01	Reserve (for future expansion)
0x02	Normal node
0x10	Reserve (for future expansion)
0x12	Reserve (for future expansion)
0x20	Reserve (for future expansion)
0x21	Master node in the Node-to-node test

5.2.5.4 netNumber

This field contains the identifier of the network to which the node belongs. The range is 1..239.

5.2.5.5 sendTime

Refer to 5.2.4.3.

5.2.5.6 loopState

This field shows the loop status of the node. The values are specified in Table 11.

Table 11 – loopState

Value	Description
0x00	Through
0x12	Loop back to “In” side, “Out” side disconnected
0x13	Loop back to “In” side, In/Out check error in “Out” side
0x14	Loop back to “In” side, In/Out check being performed at “Out” side
0x21	“In” side disconnected, loop back to “Out” side
0x31	In/Out check error at “In” side, loop back to “Out” side
0x41	In/Out check being performed at “In” side, loop back to “Out” side

5.2.5.7 parmTypeCyclicStatus

This field is divided as follows:

Bit 0..5 Cyclic status – follow values in Table 12

Bit 6..7 Parameter setting mode – follow values in Table 13

Table 12 – Cyclic status

Value	Description
0x00	Cyclic transmission is not performed
0x01	Cyclic transmission is performed
0x02	Common parameters not received
0x03	Receiving common parameters
0x04	Common parameters error
0x05	Reserved
0x06	The node number is illegal
0x07	Reserve node setting
0x08	Cyclic stop instruction
0x09	Performing off-line test
0x0A	Monitor timer time out
0x0B	Node number not set
0x0C	The node CPU error
0x0D	The node number duplication
0x0E	The management node duplication
0x0F	The node number and own management node duplication

Value	Description
0x10	Network number error

Table 13 – Parameter setting mode

Value	Description
0x0	Common parameter
0x1	Reserved

5.2.5.8 commonParamId

5.2.5.8.1 date

The bits of this field are defined as follows:

Bit 28..31	Year (hundreds digit)
Bit 24..27	Year (thousands digit)
Bit 20..23	Year (ones digit)
Bit 16..19	Year (tens digit)
Bit 12..15	Month (ones digit)
Bit 8..11	Month (tens digit)
Bit 4..7	Day (ones digit)
Bit 0..3	Day (tens digit)

If all bits are 0, there are no common parameters.

5.2.5.8.2 timeNodeId

The bits of this field are defined as follows:

Bit 28..31	Hours (ones digit)
Bit 24..27	Hours (tens digit)
Bit 20..23	Minutes (ones digit)
Bit 16..19	Minutes (tens digit)
Bit 12..15	Seconds (ones digit)
Bit 8..11	Seconds (tens digit)
Bit 0..7	Node number of setting reference

If all bits are 0, there are no common parameters.

5.2.5.8.3 checksum

This field contains the checksum of “date” and “time_nodeId” in the commandParmId. If all bits are 0, there are no common parameters.

5.2.5.9 padding

This field is 8 octets in length. The value is 0x00 for all 8 octets.

5.2.5.10 dcs

Refer to 5.2.2.4.

5.2.6 Select-PDU

5.2.6.1 falArHeader

Refer to 5.2.1.

5.2.6.2 padding

This field is padding. The value is 0x00.

5.2.6.3 dcs

Refer to 5.2.2.4.

5.2.7 Launch-PDU

5.2.7.1 falArHeader

Refer to 5.2.1.

5.2.7.2 padding

This field is padding. The value is 0x00.

5.2.7.3 dcs

Refer to 5.2.2.4.

5.2.8 Token-PDU

5.2.8.1 falArHeader

Refer to 5.2.1.

5.2.8.2 padding

This field is padding. The value is 0x00.

5.2.8.3 dcs

Refer to 5.2.2.4.

5.2.9 MyStatus-PDU

5.2.9.1 falArHeader

Refer to 5.2.1.

5.2.9.2 reserved1

This field is reserved for future use. The value is 0x0000.

5.2.9.3 nodeType

Refer to 5.2.5.3.

5.2.9.4 netNumber

Refer to 5.2.5.4.

5.2.9.5 reserved2

This field is reserved for future use. The value is 0x0000.

5.2.9.6 loopState

Refer to 5.2.5.6.

5.2.9.7 parmTypeCyclicStatus

Refer to 5.2.5.7.

5.2.9.8 commonParamId

Refer to 5.2.5.8.

5.2.9.9 inFarNodeMACAddr

This field indicates the MAC address of the node connected with “In” side. If the “In” side is not connected correctly, the value is 0.

5.2.9.10 inFarNodeNumber

This field indicates the node number of the node connected with “In” side.

5.2.9.11 reserved3

This field is reserved for future use. The value is 0x00.

5.2.9.12 outFarNodeMACAddr

This field indicates the MAC address of the node connected with “Out” side. If the “Out” side is not connected correctly, the value is 0.

5.2.9.13 outFarNodeNumber

This field indicates the node number of the node connected with “Out” side.

5.2.9.14 reserved4

This field is reserved for future use. The value is 0x00.

5.2.9.15 opState

This field indicates the status of a node. The values are specified in Table 14.

Table 14 – opState

Value	Description
0	Controller does not exist
1	Controller is stopped
2	Controller is operating

5.2.9.16 errorState

This field indicates the error status of a node. The values are specified in Table 15.

Table 15 – errorState

Value	Description
0	No error
1	Minor error
2	Major error
3	Severe error

5.2.9.17 errorCode

This field indicates the error codes of errors that occurred in a node. Error codes and their meanings are not defined here.

5.2.9.18 vendorCode

Refer to 5.2.5.2.

5.2.9.19 deviceType

This field specifies the device type. For device types, refer to device profile.

5.2.9.20 unitTypeName

This field contains the character string for model name.

5.2.9.21 unitTypeCode

This field contains the model name code.

5.2.9.22 reserved5

This field is reserved for future use. The value is 0x0000.

5.2.9.23 nodeInfo

This field indicates the user-defined node status.

5.2.9.24 dcs

Refer to 5.2.2.4.

5.2.10 Transient1-PDU

5.2.10.1 falArHeader

Refer to 5.2.1.

5.2.10.2 destinationGroup

This field specifies the destination group. Each bit represents a group address. Bit 0 shows group address 1. Bit 31 shows group address 32. If no destination group is specified, the values of all bits are 0.

5.2.10.3 seqNumber

Each bit in this field has the following meanings:

Bit 0..6	Shows the number for data fragments.
Bit 7	Shows if the data is the last fragment. The value 0 means that it is not the last PDU. The value 1 means that it is the last PDU of the last fragment.

5.2.10.4 dataId

This field contains the identification number of the transient data. The range of values is 0x00..0xFF. All fragments of a transient data transmission have the same identification number.

5.2.10.5 wholeDataSize

This field specifies the amount of transient data in octets.

5.2.10.6 offsetAddr

This field specifies the offset address. In the initial PDU, the value is 0x0. In subsequent PDUs, the position within the entire transient data is shown as an offset address from the initial PDU.

5.2.10.7 dataSize

This field specifies the fragmented transient data size in octets. The range of values is 0x0000..0x05BA.

5.2.10.8 dataType

This field specifies the data type as described in Table 16.

Table 16 – Data type

Value	Description
0	Parameter delivery
1	Reserved

5.2.10.9 data

5.2.10.9.1 Overview

The data field contains the transient data. The structure depends on the type of transient data. If wholeDataSize exceeds 1466 octets, it is the transient data from offset described in offsetAddr to the data size described in dataSize. If the transient data has less than 16 octets, the padding is filled with 0x00

5.2.10.9.2 Structure for parameter delivery

When the data type is parameter delivery, the data are selected from CPW, CPWC or CPWCR. The structure of CPW is described in Table 17, that of CPWC in Table 18 and that of CPWCR in Table 19. Each field is encoded as LOctetString.

Table 17 – CPW

Field name	Size (octets)	Description
cpfType	4	Frame type. The value is 0.
rcvNodeList	32	Receiving node list. Each bit represents a node, as LSB of the lowest octet representing node 1 and MSB of the lowest octet representing node 8. The value 1 means receiving and value 0 means not receiving.
commonParamId	12	Refer to 5.2.5.8.
cmParam	–	Refer to Table 20.

Table 18 – CPWC

Field name	Size (octets)	Description
cpfType	4	Frame type. The value is 1.
commonParamId	12	Refer to 5.2.5.8.

Table 19 – CPWCR

Field name	Size (octets)	Description
cpfType	4	Frame type. The value is 2.
checkResult	4	The result of common parameter check 0: Common parameters not received 1: Checking common parameters 2: Check OK 3: Check failed
errorCode	4	Error code when check is failed
srcNodeNumber	1	Source node number
padding	3	Padding

Table 20 – cmParam

Field name	Size (octets)	Description	
Parameter Name	8	Specified by the user	
Total Size	2	Data size from start block to end block in octet	
Sum Check Enabled	1	0: Perform sum check 1: Not perform sum check (default)	
Reserved	1	Reserved	
Sum Check Value	4	Check sum value from start block to end block	
Create Time	12	Creation year/month/day/hour/minute/second	
		Size (octets)	**Description**
		1	Last two digits of A.D.
		1	First two digits of A.D.
		2	Month
		2	Day

Field name	Size (octets)	Description	
		2	Hour
		2	Minute
		2	Second
Begin Marker	4	0x5047532d	
Param Area	0..5 924	Refer to Table 21.	
End Marker	4	0x5047452d	

Table 21 – Details of param area

Field name	Size (octets)	Description
Parameter Availability	2	Bit0: LB/LW Common Memory Area Bit1: LB/LW Common Memory Additional Area Bit8: LX/LY Common Memory Area Bit9: LX/LY Common Memory Additional Area 0 means no setting, 1 means setting available
Common Parameter Version	1	Common parameter version
Reserved	1	Reserved
LB/LW CM Area Offset	2	Offset of LB/LW CM Area
LB/LW CM Additional Area Offset	2	Offset of LB/LW CM Additional Area
LX/LY CM 1 Area Offset	2	Offset of LX/LY CM 1 Area
LX/LY CM 2 Area Offset	2	Offset of LX/LY CM 2 Area
Reserved	24	Reserved
Application Parameters	80	Refer to Table 22
LB/LW CM Area	0 or 1 452	Refer to Table 23 When the Bit0 value of Parameter Availability is 0, the size is 0, when the value is 1, the size is 1 452
LB/LW CM Additional Area	0 or 1 452	Refer to Table 23 When the Bit1 value of Parameter Availability is 0, the size is 0, when the value is 1, the size is 1 452
LX/LY CM 1 Area	0 or 1 452	Refer to Table 24 When the Bit8 value of Parameter Availability is 0, the size is 0, when the value is 1, the size is 1 452
LX/LY CM 2 Area	0 or 1 452	Refer to Table 24 When the Bit9 value of Parameter Availability is 0, the size is 0, when the value is 1, the size is 1 452

Table 22 – Details of application parameters

Field name	Size (octets)	Description
Control Block	4	0x0
Reserved1	1	Reserved 1
Reserved2	1	Reserved 2

Field name	Size (octets)	Description
Total Nodes	1	Total number of nodes
Network Type	1	Network type
Acyclic Times	2	Transient times
Supervisory Period	2	Monitoring time (in 1ms)
Reserved3	20	Reserved 3
Reserved4	16	Reserved 4
Reserved5	16	Reserved 5
Reserved6	16	Reserved 6

Table 23 – Details of LB/LW CM area and LB/LW CM additional area

Field name		Size (octets)	Description
LW CM Head Address		4	LW head relative address
LW CM Total Size		4	Total number of words in LW setting range
LB CM Head Address		2	LB head relative address
LB CM Total Size		2	Total size of LB setting range (in 2 octet units)
LB/LW CM Table List		1 440 (12 x 120)	Parameter table list of LB and LW
	LW CM Head Address Of Node	4	LW head relative address in each node
	LW CM Size	4	Size of LW in each node (in 2 octet units)
	LB CM Head Address Of Node	2	LB head relative address in each node
	LB CM Size	2	Size of LB in each node (in 2 octet units)

Table 24 – Details of LX/LY CM 1 area and LX/LY CM 2 area

Field name		Size (octets)	Description
Master Node Number		1	Master node number
Reserved		3	Reserved
LY CM Head Address		2	LY head relative address
LY CM Total Size		2	Total number of words in LY setting range
LX CM Head Address		2	LX head relative address
LX CM Total Size		2	Total number of words in LX setting range
LX/LY CM Table List		1 440 (12 x 120)	Parameter table list of LX/LY
	LY CM Head Address Sent	2	LY head relative address sent by each Node
	LY CM Size	2	LY size sent by each node (in 2 octet units)
	LX CM Head Address Master Received	2	LX head relative address received by the master node
	LX CM Head Address Received	2	LX head relative address received by each node
	LX CM Size	2	LX size received by each node (in 2 octet units)
	LY CM Head Address Master Sent	2	LY head relative address sent by the master node

5.2.10.10 evenPadding

This field is only used when the data field is an odd number octets. The value is 0x00.

5.2.10.11 dcs

Refer to 5.2.2.4.

5.2.11 Dummy-PDU

5.2.11.1 falArHeader

Refer to 5.2.1.

5.2.11.2 dummyData

This field contains dummy data. The size is 28 to 1 482 octets. The value has no meaning.

5.2.11.3 dcs

Refer to 5.2.2.4.

5.2.12 Transient2-PDU

5.2.12.1 falArHeader

Refer to 5.2.1.

5.2.12.2 l

This field specifies the data length from fno to data (in octets).

5.2.12.3 gcnt

This field represents the maximum relay times as a gate count. The default value is 0x07. The value is decremented through each relay.

5.2.12.4 typeSeqF

This field is divided as follows:

Bit 7..4	Shows the type. The value is 0x00.
Bit 3..0	Represents the sequence number.

5.2.12.5 fno

This field is divided as follows:

Bit 7	Shows the identification of head frame. If the value is 0, it means that the frame is a non-head frame. If the value is 1, it means that the frame is a head frame.
Bit 6..0	Shows the divided frame number. The value 0 means that the frame was not divided. The values from 1 to 7, shows the divided frame number. The divided frame number starts from the same value of the number of division, and is subtracted in sequence. For example, in a 3-divided frame, the order of frame numbers is: 3, 2, 1.

5.2.12.6 dt

This field is divided as follows:

Bit 7	Shows the priority. When the value is 0, it means that the priority is low. When the value is 1, it means that the priority is high.
Bit 6	Shows the presence of response frames. When the value is 0, a response frame is required. When the value is 1, it is not needed.
Bit 5..0	Reserved for future use.

5.2.12.7 da

This field contains the node number of a relay node. When a destination node is in the same network, this field identifies the node number of the destination.

5.2.12.8 sa

This field contains the node number of a relay node. When a destination node is in the same network, this field identifies the node number of the source.

5.2.12.9 dat

This field contains the target application type. The value is fixed as 0x22.

5.2.12.10 sat

This field contains the source application type. The value is fixed as 0x22.

5.2.12.11 dmf

This field contains the destination module flag. The values are described in Table 25.

Table 25 – Destination module flag

Module name	Description
0	Inside the network module
1	Inside the controller

5.2.12.12 smf

This field contains the source module flag. The values are described in Table 25.

5.2.12.13 dna

This field contains the network number of destination node.

5.2.12.14 ds

This field contains the node number of destination node.

5.2.12.15 did

This field is divided as follows:

Bit 10..15	System area (stores the destination node number)
Bit 0..9	Identification number of target. The value is fixed as 0x3FF.

5.2.12.16 sna

This field contains the network number of source node.

5.2.12.17 ss

This field contains the node number of source node.

5.2.12.18 sid

This field is divided as follows:

Bit 10..15	System area (Stores the source node number)
Bit 0..9	Identification number of source. The value is fixed as 0x3FF.

5.2.12.19 l1

This field specifies the data length from ct to data (in octet).

5.2.12.20 ct

This field contains the command type as shown in Table 26.

Table 26 – Command types

Command type	Description
0x00	Unavailable
0x01	Reserved
0x02	Reserved
0x03	Reserved
0x04	Get memory access information
0x05..0x07	Reserved
0x08	RUN
0x09	STOP
0x0A..0x0E	Reserved
0x0F	Reserved
0x10	Read memory
0x11	Reserved
0x12	Write memory
0x13..0x5F	Reserved
0x60..0x7F	Vendor specific

5.2.12.21 rsv

This field is reserved for future use. The value is 0x0.

5.2.12.22 aps

This field is divided as follows:

Bit 8..15	Task number. The range of values is 0..255.
Bit 0..7	Identification number of source application. The range of values is 0..255.

5.2.12.23 data

5.2.12.23.1 Overview

This field contains the transient data. The structure of the transient data depends on ct (refer to 5.2.12.20).

5.2.12.23.2 Get memory access information

When requesting a get memory access information, this area is not used. The structure of the response is shown in Figure 2. The definition of attribute is shown in Figure 3. The definition of the access code is shown in Figure 4.

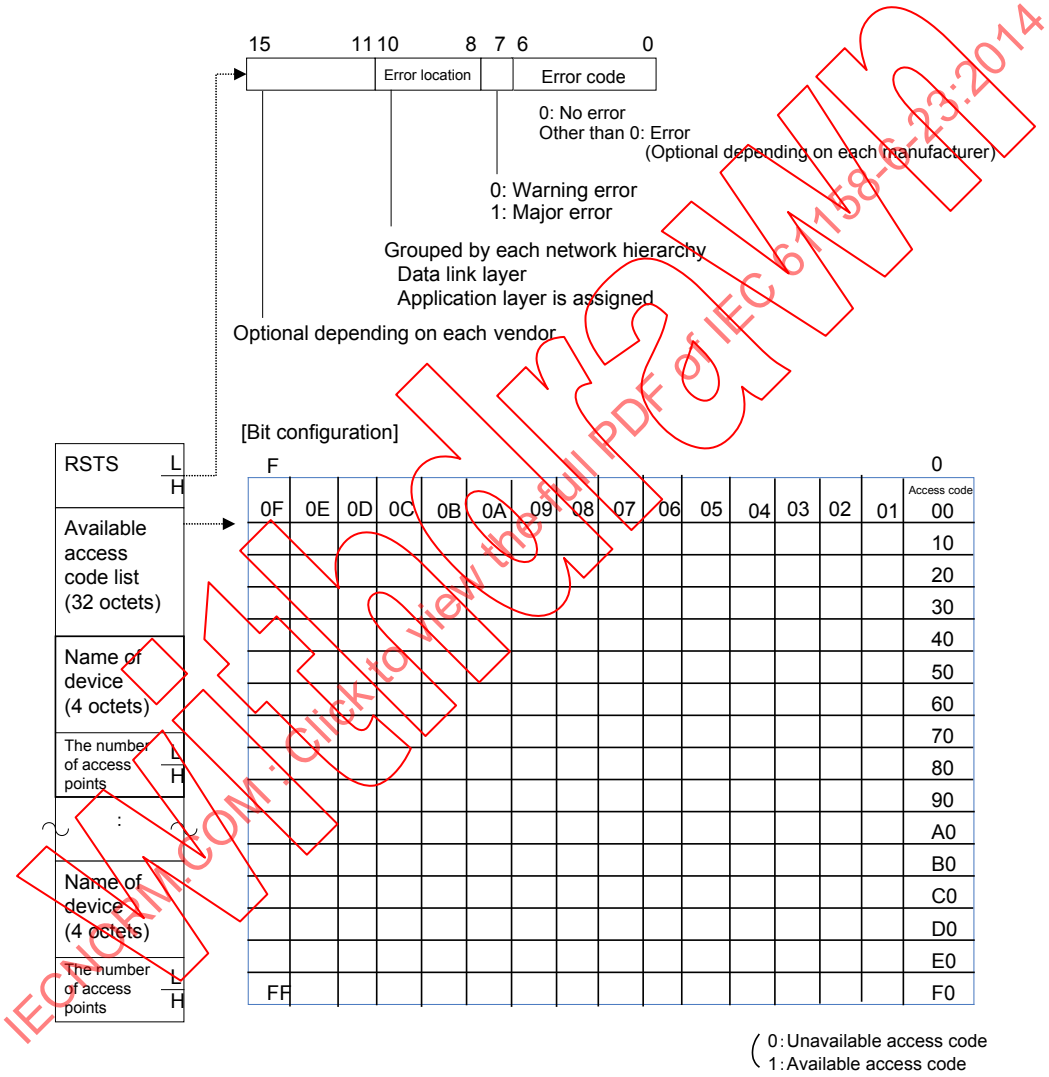


Figure 2 – Structure for memory access information retrieve response

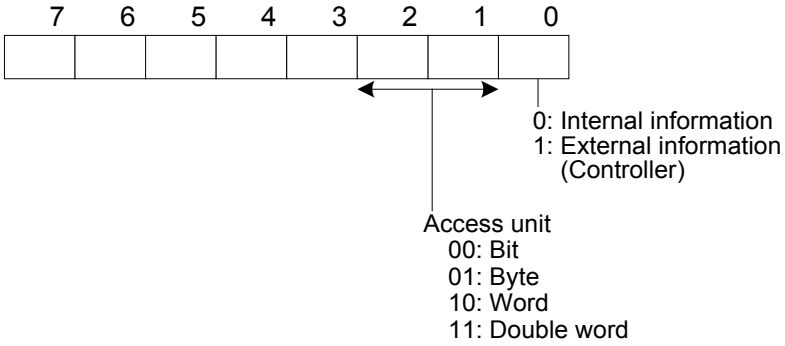


Figure 3 – Attribute definitions

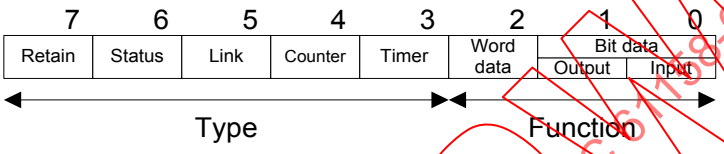


Figure 4 – Access code definitions

An example of a network module memory access code is described in Table 27. An example of a controller memory access code is described in Table 28.

Table 27 – Access codes of network module memory

Memory contents		Access code
Buffer	Standard buffer	0x00
Status buffer	Intelligent device node Auto-refresh buffer	0x40
Link buffer	Random access buffer	0x20
Link Device	Link input	0x21
	Link output	0x22
	Link register	0x24
	Link special relay	0x63
	Link special register	0x64
NOTE Specific fields and property values are vendor dependent.		

Table 28 – Access codes of controller memory

Memory contents	Access code	Type	
		B	W
Input relay	0x01	x	
Output relay	0x02	x	
Special relay	0x43	x	
Special register	0x44		x
Internal relay	0x03	x	
Latch relay	0x83	x	
Timer (contact)	0x09	x	

Memory contents	Access code	Type	
		B	W
Timer (coil)	0x0A	x	
Timer (current value)	0x0C		x
Retentive timer (contact)	0x89	x	
Retentive timer (coil)	0x8A	x	
Retentive timer (current value)	0x8C		x
Counter (contact)	0x11	x	
Counter (coil)	0x12	x	
Counter (current value)	0x14		x
Data register	0x04		x
File register	0x84		x
Link relay	0x23	x	
Link register	0x24		x
Link special relay	0x63	x	
Link special register	0x64		x
NOTE Device memory name, number and access range are PLC dependent.			

5.2.12.23.3 Run

The structure when requesting RUN is shown in Figure 5.

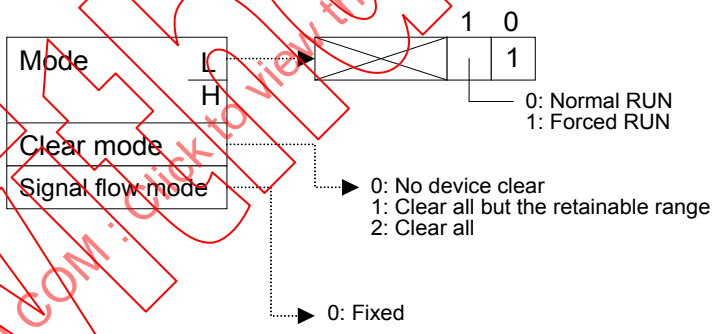


Figure 5 – Structure for RUN request

The structure when responding is shown in Figure 6.

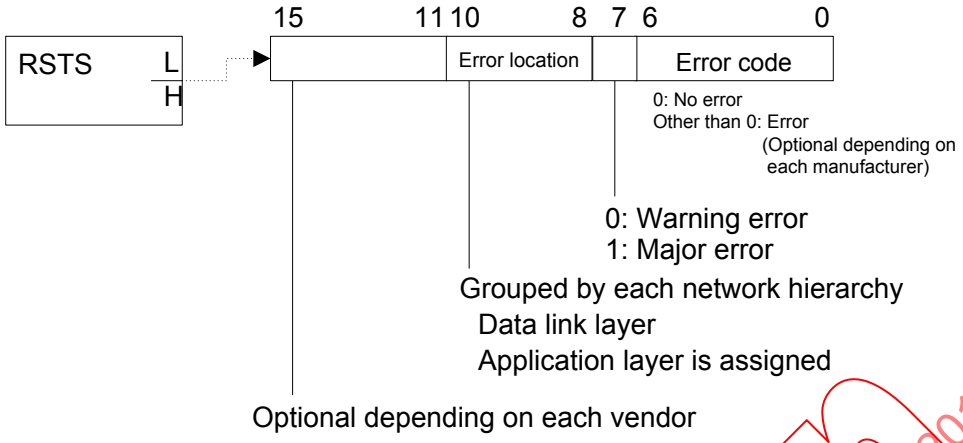


Figure 6 – Structure for RUN response

5.2.12.23.4 Stop

The structure when requesting STOP is shown in Figure 7. The structure when responding is shown in Figure 8.

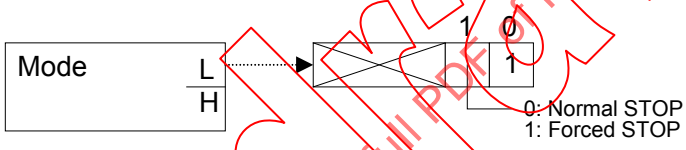


Figure 7 – Structure for STOP request

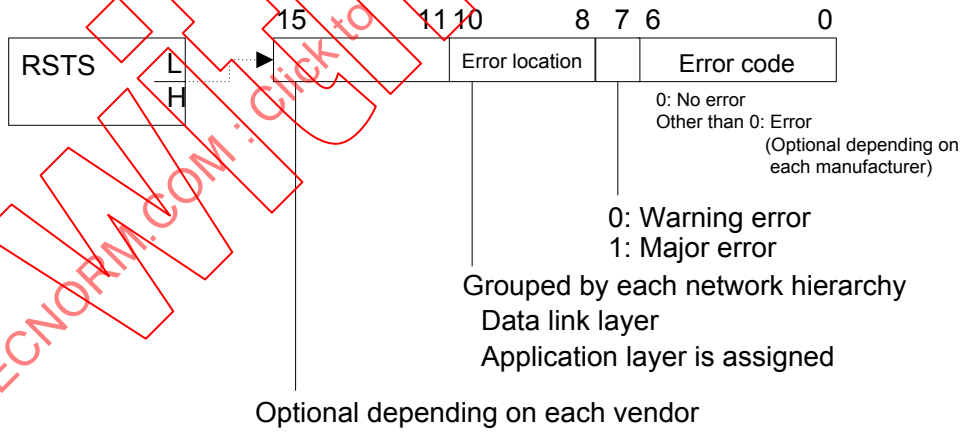


Figure 8 – Structure for STOP response

5.2.12.23.5 Read memory

The structure when requesting batch memory read is shown in Figure 9. The structure when responding is shown in Figure 10.

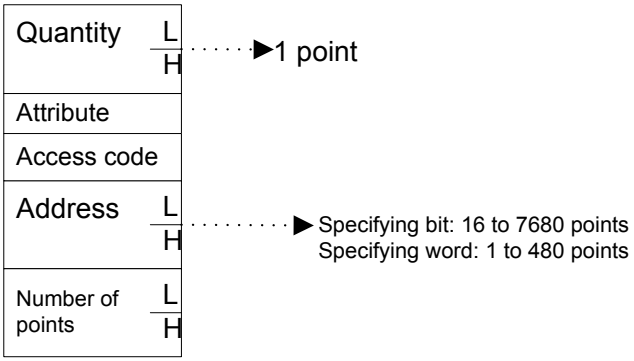


Figure 9 – Structure for batch memory read request

Figure 10 – Structure for batch memory read response

The structure when requesting random memory read is shown in Figure 11. The structure when responding is shown in Figure 12.

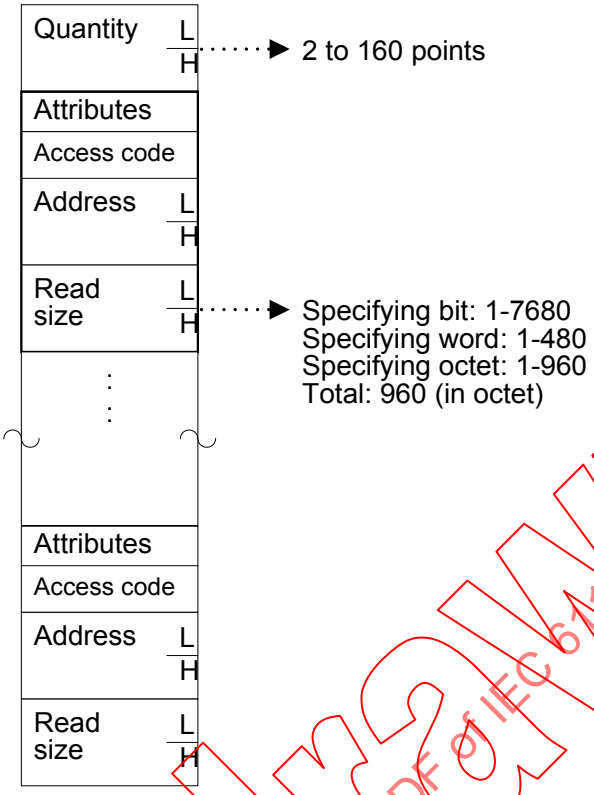


Figure 11 – Structure for random memory read request

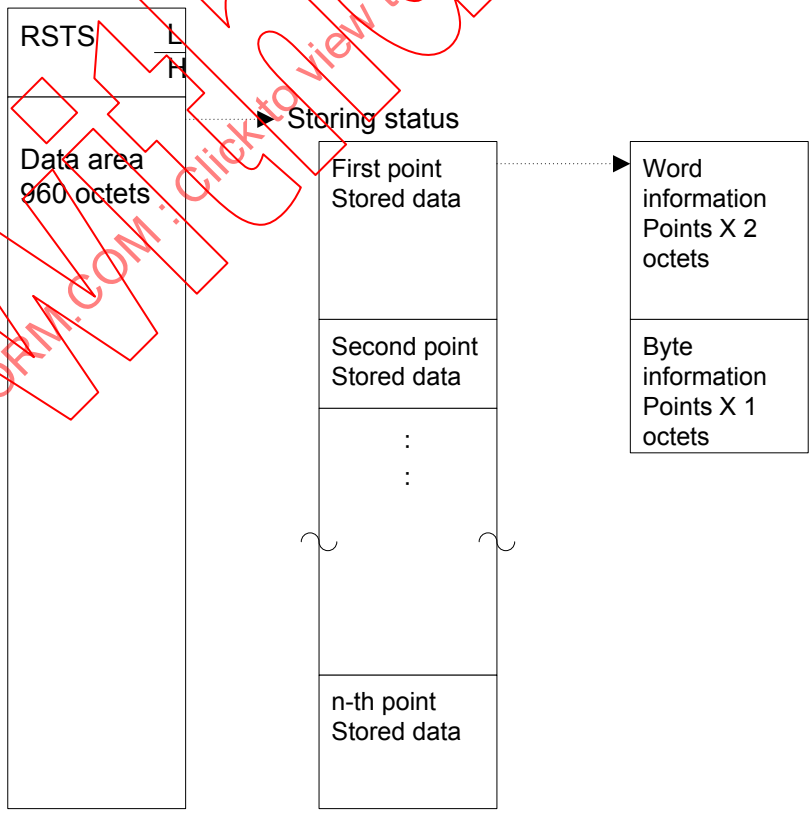


Figure 12 – Structure for random memory read response

5.2.12.23.6 Write memory

The structure when requesting batch memory write is shown in Figure 13. The structure when responding is shown in Figure 14.

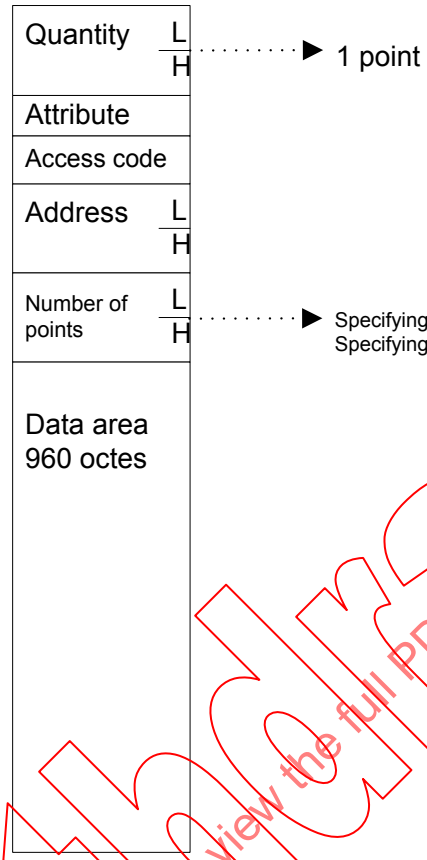


Figure 13 – Structure for batch memory write request

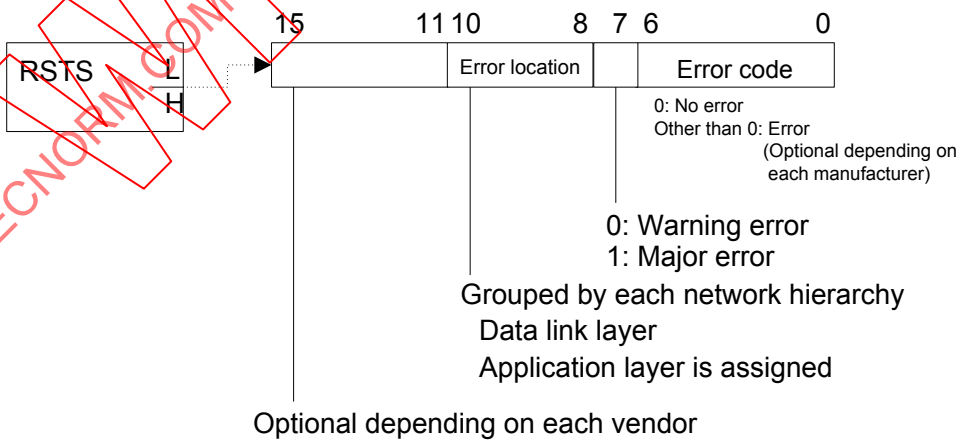


Figure 14 – Structure for batch memory write response

The structure when requesting random memory write is shown in Figure 15. The structure when responding is shown in Figure 16.

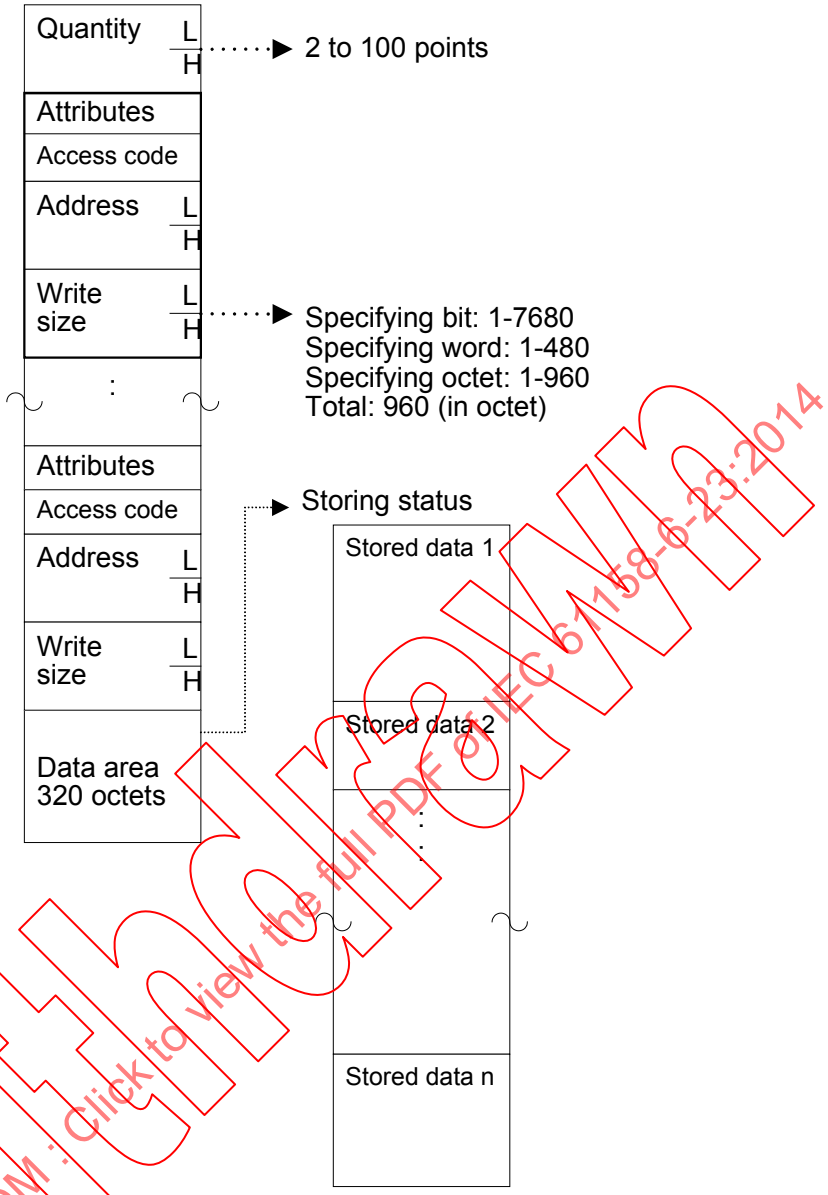


Figure 15 – Structure for random memory write request

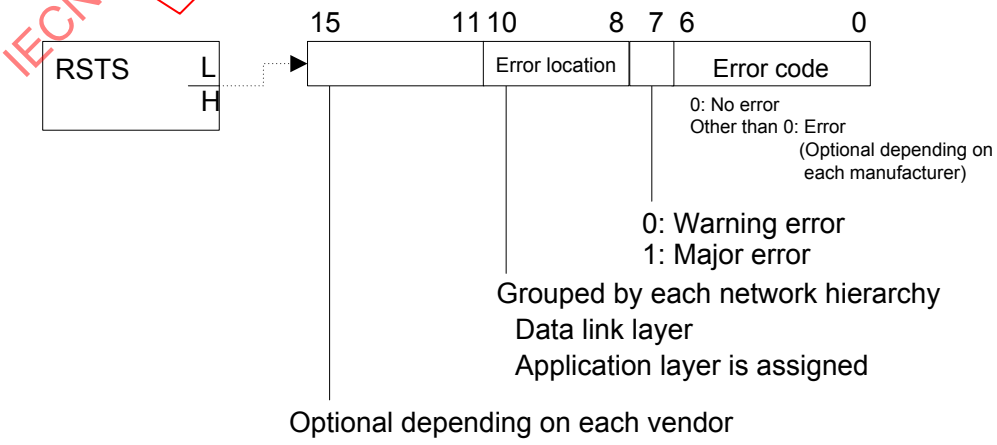


Figure 16 – Structure for random memory write response

5.2.12.24 evenPadding

This field is only used when data field contains an odd number of octets. The value is 0x00.

5.2.12.25 dcs

Refer to 5.2.2.4.

5.2.13 NTNTest-PDU

5.2.13.1 falArHeader

Refer to 5.2.1.

5.2.13.2 ntnTestData

This field contains test dummy data. The size is from 28 to 1 482 octets. The value has no meaning.

5.2.13.3 dcs

Refer to 5.2.2.4.

5.2.14 CyclicDataW-PDU

5.2.14.1 falArHeader

Refer to 5.2.1.

5.2.14.2 seqNumber

Refer to 5.2.10.3.

5.2.14.3 byteValidity

This field specifies if the initial 4 octets and last 4 octets of wData are reflected on shared memory. The bit definitions are shown in Table 29. For each bit, 1 = valid, and 0 = invalid.

Table 29 – byteValidity

Bit	Octet of wData
0	first octet
1	second octet
2	third octet
3	fourth octet
4	fourth octet from end
5	third octet from end
6	second octet from end
7	last octet

5.2.14.4 dataSize

This field specifies the cyclic data size. The range for this value is 0x0..0x16F. The size is specified in the number of groups of 4 octets, and the first 4 octets and the last 4 octets with byteValidity also included in the size.

5.2.14.5 offsetAddr

This field specifies the offset address of LW in octets. The range for this value is 0x0..0x3FFFC with only multiples of 4 permitted. That is, the value of Bit 0..1 is 0.

5.2.14.6 exSeqNumber

This field identifies the sequence number for the fragmented data. The range for this value is 0x0..0xFFFF. When this field is in use, seqNumber value is 0x7F.

5.2.14.7 reserved

This field is reserved for future use. The value is 0x0.

5.2.14.8 wData

This field contains the LW data. If data value is less than 16 octets in length, the padding is filled with 0x00.

5.2.14.9 evenPadding

This field is only used when wData is an odd number of octets in length. The value is 0x00.

5.2.14.10 dcs

Refer to 5.2.2.4.

5.2.15 CyclicDataB-PDU

5.2.15.1 falArHeader

Refer to 5.2.1.

5.2.15.2 seqNumber

Refer to 5.2.14.2

5.2.15.3 byteValidity

Refer to 5.2.14.3.

5.2.15.4 dataSize

Refer to 5.2.14.4.

5.2.15.5 offsetAddr

This field specifies the offset address of LB in octets. The range of the value is 0x0..0x3FFFC with only multiples of 4 permitted. That is, the value of Bit 0..1 is 0.

5.2.15.6 reserved1

This field is reserved for future use. The value is 0x0.

5.2.15.7 reserved2

This field is reserved for future use. The value is 0x0.

5.2.15.8 bData

This field contains the LB data. If data has less than 16 octets, the padding is filled with 0x00.

5.2.15.9 evenPadding

This field is only used when bData is an odd number of octets in length. The value is 0x00.

5.2.15.10 dcs

Refer to 5.2.2.4.

5.2.16 CyclicDataOut1-PDU

5.2.16.1 falArHeader

Refer to 5.2.1.

5.2.16.2 seqNumber

Refer to 5.2.14.2.

5.2.16.3 byteValidity

This field is not used. The value is 0x00.

5.2.16.4 dataSize

Refer to 5.2.14.4.

5.2.16.5 offsetAddr

This field is not used. The value is 0x00.

5.2.16.6 reserved1

This field is reserved. The value is 0x0.

5.2.16.7 reserved2

This field is reserved. The value is 0x0.

5.2.16.8 out1Data

This field contains the LY1 data sent by the master node to all receiving nodes. If the value of the data is less than 16 octets in length, the padding is filled with 0x00.

5.2.16.9 evenPadding

This field is only used when out1Data is an odd number of octets in length. The value is 0x00.

5.2.16.10 dcs

Refer to 5.2.2.4.

5.2.17 CyclicDataOut2-PDU

5.2.17.1 falArHeader

Refer to 5.2.1.

5.2.17.2 seqNumber

Refer to 5.2.14.2

5.2.17.3 byteValidity

This field is not used. The value is 0x00.

5.2.17.4 dataSize

Refer to 5.2.14.4.

5.2.17.5 offsetAddr

This field is not used. The value is 0x00.

5.2.17.6 reserved1

This field is reserved. The value is 0x0.

5.2.17.7 reserved2

This field is reserved. The value is 0x0.

5.2.17.8 out2Data

This field contains the LY2 data sent by the master node to all receiving nodes. If the value of this data is less than 16 octets in length, the padding is filled with 0x00.

5.2.17.9 evenPadding

This field is only used when out2Data is an odd number of octets in length. The value is 0x00.

5.2.17.10 dcs

Refer to 5.2.2.4.

5.2.18 CyclicDataIn1-PDU

5.2.18.1 falArHeader

Refer to 5.2.1.

5.2.18.2 seqNumber

Refer to 5.2.14.2.

5.2.18.3 byteValidity

Refer to 5.2.14.3.

5.2.18.4 dataSize

Refer to 5.2.14.4.

5.2.18.5 offsetAddr

This field specifies the offset address of LX in octets. The range of the value is 0x0..0x3FFFC with only multiples of 4 permitted. That is, the value of Bit 0..1 is 0.

5.2.18.6 reserved1

This field is reserved. The value is 0x0.

5.2.18.7 reserved2

This field is reserved. The value is 0x0.

5.2.18.8 in1Data

This field contains the LX1 data sent by each sending node to the master node. If the value of the data is less than 16 octets in length, the padding is filled with 0x00.

5.2.18.9 evenPadding

This field is only used when in1Data is an odd number of octets in length. The value is 0x00.

5.2.18.10 dcs

Refer to 5.2.2.4.

5.2.19 CyclicDataIn2-PDU

5.2.19.1 falArHeader

Refer to 5.2.1.

5.2.19.2 seqNumber

Refer to 5.2.14.2.

5.2.19.3 byteValidity

Refer to 5.2.14.3.

5.2.19.4 dataSize

Refer to 5.2.14.4.

5.2.19.5 offsetAddr

This field specifies the offset address of LX in octets. The range of the value is 0x0..0x3FFFC with only multiples of 4 permitted. That is, the value of Bit 0..1 is 0.

5.2.19.6 reserved1

This field is reserved. Set to 0x0.

5.2.19.7 reserved2

This field is reserved. Set to 0x0.

5.2.19.8 in2Data

This field contains the LX2 data sent by each sending node to the master node. If the value of the data is less than 16 octets in length, the padding is filled with 0x00.

5.2.19.9 evenPadding

This field is only used when in2Data is an odd number of octets in length. The value is 0x00.

5.2.19.10 dcs

Refer to 5.2.2.4.

5.3 FALPDU type F elements encoding

5.3.1 FALARHeader

5.3.1.1 arFType

This field shows the PDU types described in Table 30.

Table 30 – afFType

Value	Description
0x00..0x06	Used in type C
0x07..0x09	Reserved for future use
0x10	persuasion-PDU
0x11	testData-PDU
0x12	testDataAck-PDU
0x13	setup-PDU
0x14	setupAck-PDU
0x15	token-PDU
0x16..0x1B	Reserved for future use
0x1C	timer-PDU
0x1D..0x1F	Reserved for future use
0x20	myStatus-PDU
0x21	Reserved for future use
0x22	transient1-PDU
0x23	transientAck-PDU
0x24	Used in type C
0x25	transient2-PDU
0x26..0x27	Reserved for future use
0x28	paramCheck-PDU
0x29	parameter-PDU
0x2A..0x3F	Reserved for future use
0x40	measure-PDU
0x41	measureAck-PDU
0x42	offset-PDU
0x43	update-PDU
0x44..0x7F	Reserved for future use
0x80..0x81	Used in type C
0x82	cyclicDataRWw-PDU
0x83	cyclicDataRY-PDU
0x84	cyclicDataRWr-PDU
0x85	cyclicDataRX-PDU
0x86..0x8B	Reserved for future use
0x8C..0x8F	Used in type C

Value	Description
0x90..0xFF	Reserved for future use

5.3.1.2 dataType

This field indicates the data type, the values of which are per arFtype as described in Table 31.

Table 31 – dataType

arFtype	Value	Description
0x10..0x15	0x00	Reserved for future use
	0x01	Transmission control
	0x02..0xFF	Reserved for future use
0x1C	0x00	Reserved for future use
	0x01	Transient transmission (type F specific)
	0x02..0xFF	Reserved for future use
0x20	0x00..0x01	Reserved for future use
	0x02	Transmission control (diagnostics)
	0x03..0xFF	Reserved for future use
0x22	0x00..0x06	Reserved for future use
	0x07	Transient transmission (type F specific)
	0x08	Transient transmission (drive specific)
	0x09	Transient transmission (safety specific)
	0x10..0xFF	Reserved for future use
0x23	0x00..0xFF	Transient transmission response
0x25	0x00..0x03	Reserved for future use
	0x04	Transient transmission (CP8/1, CP8/2 compatible)
	0x05..0xFF	Reserved for future use
0x28-0x29	0x00..0x02	Reserved for future use
	0x03	Cyclic transmission setting
	0x04..0xFF	Reserved for future use
0x82-0x85	0x00	Cyclic transmission
	0x01..0xFF	Reserved for future use

5.3.1.3 varField

5.3.1.3.1 Overview

This field contains the fields listed in Table 32 which are per arFtype as described

Table 32 – varField

arFtype	Field used
persuasion-PDU testDataAck-PDU	persPriority nodeType
testData-PDU setupAck-PDU	reserved1

arFtype	Field used
timer-PDU	
setup-PDU token-PDU cyclicDataRWw-PDU cyclicDataRY-PDU cyclicDataRWr-PDU cyclicDataRX-PDU measure-PDU measureAck-PDU offset-PDU update-PDU	nodeId reserved2
myStatus-PDU	nodeId syncFlag nodeType
transient1-PDU transientAck-PDU transient2-PDU paramCheck-PDU parameter-PDU	nodeId connectionInfo reserved4

5.3.1.3.2 persPriority

This field indicates the priority level when transmission control manager is selected. The range of this value is 0x0..0xFFFFFF. The value 0x0 indicates that the node is not the transmission control manager.

5.3.1.3.3 nodeType

This field indicates the node type. The values are specified in Table 33.

Table 33 – nodeType

Value	Description
0x00..0x2F	Used in type C
0x30	Master station
0x31	Reserved for future use
0x32	Local station
0x33	Intelligent device station
0x34	Remote device station
0x35	Remote I/O station
0x36..0xFF	Reserved for future use

5.3.1.3.4 reserved1

Reserved for future use. The value of each octet is 0x00.

5.3.1.3.5 nodeId

This field contains the node identifier. The transmission control manager determines the node identifiers of nodes other than the transmission control manager, and sets the values using the setup-PDU. The range of values are 0..255. When a node number has not been set, the value is 255.

5.3.1.3.6 reserved2

This field is reserved for future use. The value of each octet is 0x00.

5.3.1.3.7 syncFlag

Each bit has the following meaning:

Bits 7..1	Reserved for future use. The value is 0.
Bit 0	Indicates the synchronization flag. The value is 1 when the frame provides notification of the synchronization timing, and 0 at any other time.

5.3.1.3.8 connectionInfo

This field identifies the f-transientData-PDU to be transmitted when one token is kept. When one token is kept and a duplicate non-cyclic transmission PDU having the same destaddr and the same connectionInfo is received, the receiving node discards it.

5.3.1.3.9 reserved4

This field is reserved for future use. The value of each octet is 0x00.

5.3.1.3.10 srcNodeNumber

This field contains the identification number of the source node.

5.3.1.3.11 protocolVerType

Each bit has the following meaning:

Bits 7..4:	Protocol version. The value is 0.
Bits 3..0:	Protocol type. The values used are in accordance with Table 34.

Table 34 – ProtocolVerType

Value	Description
0x0	type C
0x1	type F
0x2..0xF	Reserved for future use

5.3.1.3.12 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.1.3.13 hec

This field contains an error detection code that targets DestAddr of DLPDU. The DLPDU definitions are as follows.

```
DLPDU ::= SEQUENCE {
    preamble                Preamble,
    sfd                     SFD,
    destaddr                DestAddr,
    srcaddr                 SrcAddr,
    lt                      LT,
    dlsdu                   FAL-PDU,
    fcs                     FCS
}

Preamble ::= OctetString (SIZE(7))
SFD ::= OctetString (SIZE(1))
DestAddr ::= MACAddress
SrcAddr ::= MACAddress
LT ::= Unsigned16
FCS ::= OctetString (SIZE(4))
```

The generating polynomial is $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x1+1$.

5.3.2 Persuasion-PDU

5.3.2.1 falArHeader

Refer to 5.3.1.

5.3.2.2 reserved1

This field is reserved for future use. The value of each octet is 0x00.

5.3.2.3 myPorts

This field indicates the number of physical communication ports held by the node.

5.3.2.4 vendorCode

This field contains the vendor code.

5.3.2.5 modelCode

This field contains the model code which is vendor specific.

5.3.2.6 reserved2

This field is reserved for future use. The value of each octet is 0x00.

5.3.2.7 dcs

This field contains the error detection code that targets destaddr of DLPDU to the previous field of dcs. For DLPDU definitions, refer to 5.3.1.3.13.

The generating polynomial is $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$.

5.3.3 TestData-PDU

5.3.3.1 falArHeader

Refer to 5.3.1.

5.3.3.2 tmMacAddr

This field contains the MAC address of the transmission control manager.

5.3.3.3 srcPort

This field contains the transmission source port number of the testData-PDU. The value 0 is invalid.

5.3.3.4 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.3.5 dcs

Refer to 5.3.2.7.

5.3.4 TestDataAck-PDU

5.3.4.1 falArHeader

Refer to 5.3.1.

5.3.4.2 tdSrcMacAddr

This field contains the MAC address of the transmission source node of the testData-PDU included as the srcaddr in DLPDU in the received testData-PDU. The target is the testData-PDU that served as the impetus for transmitting the testDataAck-PDU.

5.3.4.3 tdSrcPort

This field contains the source port number of the transmission source node of the testData-PDU that is included as the srcPort in the received testData-PDU.

5.3.4.4 tdRcvPort

This field contains the port number of its own node that received the testDataAck-PDU.

5.3.4.5 reserved1

This field is reserved for future use. The value of each octet is 0x00.

5.3.4.6 myPorts

Refer to 5.3.2.3.

5.3.4.7 tokenKeepTime

This field specifies the maximum value of the duration the node keeps a token after token passing starts. The node notifies the transmission control manager of the token keep time using this field.

Each bit has the following meaning:

Bit 15:	Reserved for future use. The value is 0.
Bits 14..0:	Indicates the time setting. The unit is 1 µs. The range of values is 1..32 767.

NOTE The token keep time start and end points are the same times as those of the token-PDU. If the start point is the receiving start time of the token-PDU, the end point is the transmission start time of the token-PDU. If the

start point is set as the receiving completion time of the token-PDU, the end point is set as the transmission completion time of the token-PDU.

5.3.4.8 reserved2

This field is reserved for future use. The value of each octet is 0x00.

5.3.4.9 myConnectStatus

This field indicates the status of the each port. The field is used to show the status of 24 ports, maximum, using four bits for each port. The field shows the status of port 1 in bits 3..0 and the status of port 2 in bits 7..4 of the first octet. The field shows the status of port 3 in bits 3..0 and the status of port 4 in bits 7..4 of the second octet. Similarly, the field then uses up to 12 octets to show the statuses of 24 ports. Each bit of one octet has the following meaning:

Each bit has the following meaning:

Bits 7..6: Reserved for future use. The value is 0.

Bits 5..4: The values are in accordance with Table 35.

Bits 3..2: Reserved for future use. The value is 0.

Bits 1..0: The values are in accordance with Table 35.

Table 35 – Link status

Value	Description
00b	Link down
01b	Link up (1 Gbps)
10b	Reserved for future use
11b	Reserved for future use

5.3.4.10 dcs

Refer to 5.3.2.7.

5.3.5 Setup-PDU

5.3.5.1 faiArHeader

Refer to 5.3.1.

5.3.5.2 tokenDstMacAddr

This field specifies the MAC address of the next node that will keep the token on the token passing path. The transmission control manager transmits the setup-PDU that set the MAC address of the token-PDU transmission destination in this field to the node that serves as the token-PDU transmission origin. The MAC address specified in this field is kept in the node that received the setup-PDU.

After token passing begins, the transmission control manager becomes the first node to hold the token. The node that holds the token transmits its own data and then transmits the token-PDU with the MAC address specified in this field in the tokenDstMacAddr. The node that receives the token-PDU determines whether or not its own node MAC address and the tokenDstMacAddr of the token-PDU match and, if so, assesses that the token-PDU is addressed to its own node address. A node that receives a token-PDU addressed to itself becomes the node that holds the token.

5.3.5.3 reserved1

This field is reserved for future use. The value of each octet is 0x00.

5.3.5.4 leaveTimerValue

This field specifies the setting value of the LeaveTimer. The transmission control manager enters the LeaveTimer value to be set in a node other than the transmission control manager in this field of the setup-PDU to be transmitted. The value of this field is set in the node that receives the setup-PDU as the LeaveTimer value. The LeaveTimer is used to measure the duration of detection of disconnection of its own node.

The bits of this field have the following meanings:

Bit 15:	Reserved for future use. The value is 0.
Bits 14..0:	Leave Timer value. The value range is 1..32 767. The unit is 400 µs.

5.3.5.5 portUsage

This field indicates the enabled/disabled setting of the transmission/reception function of the port that is associated with the node. A disabled port means that the port does not use DL service. The values used are in accordance with Table 36.

In a node with two ports that receive setting values, the enabled/disabled status of the ports of the node is determined in accordance with Table 36. A node that has three or more ports that received setting values enables all ports without using the specified setting values.

Table 36 – Port enable/disable specification

Value	Description
0x00	Enable both ports
0x01	Enable port 1 only, and disable all other ports.
0x02	Enable port 2 only, and disable all other ports.

5.3.5.6 reserved2

This field is reserved for future use. The value of each octet is 0x00.

5.3.5.7 netBehaviour

5.3.5.7.1 multipleTransmit

This field specifies the number of times the node that is holding the token repeatedly performs transmission of FALPDUs other than the token-PDU. The range of values is 1..255. 0 is not valid.

When this field contains the value 1, the node that holds the token transmits the myStatus-PDU, f-cyclicData-PDU, and f-transientData-PDU, then lastly transmits the token-PDU, causing the node to no longer hold the token. If not applicable, the node skips the f-transientData-PDU. The token holding node repeatedly transmits the myStatus-PDU, f-cyclicData-PDU, and f-transientData-PDU the number of times specified in this field, after which it transmits the token-PDU.

NOTE For example, when an intelligent device station that specifies 2 in this field becomes the token holding node, the station transmits the myStatus-PDU, cyclicRWr-PDU, cyclicRX-PDU, and transient1-PDU, and then once again transmits the myStatus-PDU, cyclicRWr-PDU, cyclicRX-PDU, and transient1-PDU, followed by the token-PDU.

5.3.5.7.2 frameInterval

This field specifies the interval after token-PDU reception to myStatus-PDU transmission. The value range is 1..255. 0 is not valid. 1 indicates that transmission is to be performed based on an interval equivalent to the gap between frames, resulting in the shortest Ethernet frame transmission interval. 2 indicates that transmission is to be performed based on an interval of one frame added to the token-PDU plus the gap between frames before and after that frame. When 2 or higher is specified, the value indicates that transmission is to be performed based on an interval of the token-PDUs of a number of frames equivalent to the specified number minus one, plus the gap between frames before and after each frame.

5.3.5.7.3 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.5.7.4 multipleTokens

This field specifies the number of times transmission of the token-PDU to be transmitted during one token holding period. The value range is 1..255. 0 is invalid and shall not be used.

NOTE For example, when an intelligent device station specifies 2 in this field and becomes the node that holds the token, the station transmits the myStatus-PDU, cyclicRWr-PDU, cyclicRX-PDU, transient1-PDU, and token-PDU, and then transmits the token-PDU once again.

5.3.5.8 reserved3

This field is reserved for future use. The value of each octet is 0x00.

5.3.5.9 dcs

Refer to 5.3.2.7.

5.3.6 SetupAck-PDU

5.3.6.1 falArHeader

Refer to 5.3.1.

5.3.6.2 slaveNodeInfo

Each bit has the following meaning:

Bits 7..2: Reserved for future use. The value is 0.

Bits 1..0: Specifies I/O type.

00 = mixed (input and output share the same address)
01 = input
10 = output
11 = composite (input and output with unique addresses)

5.3.6.3 fwVersion

This field specifies the firmware version.

5.3.6.4 DeviceType

This field specifies the device type.

5.3.6.5 reserved1

This field is reserved for future use. The value of each octet is 0x00.

5.3.6.6 vendorCode

Refer to 5.3.2.4.

5.3.6.7 modelCode

Refer to 5.3.2.5.

5.3.6.8 rySize

This field specifies the RY size (number of octets).

5.3.6.9 rwwSize

This field specifies the RWw size (number of words).

5.3.6.10 rxSize

This field specifies the RX size (number of octets).

5.3.6.11 rwrSize

This field specifies the RWr size (number of words).

5.3.6.12 reserved2

This field is reserved for future use. The value of each octet is 0x00.

5.3.6.13 availableFuncs

Each bit has the following meaning:

Bits 7..2:	Reserved for future use. The value is 0.
Bit 1:	Indicates the presence of the node number setting function from the parameter-PDU. 0 indicates the function is not available, and 1 indicates the function is available. The parameter-PDU for node number setting shall not be sent to a node that does not have the function.
Bit 0:	Indicates the presence of the transient receiving function. 0 indicates that the function does not exist, and 1 indicates that the function exists.

5.3.6.14 reserved3

This field is reserved for future use. The value of each octet is 0x00.

5.3.6.15 dcs

Refer to 5.3.2.7.

5.3.7 F-Token-PDU

5.3.7.1 falArHeader

Refer to 5.3.1.

5.3.7.2 tokenDstMacAddr

This field contains the MAC address of the token-PDU transmission destination node. Each node assesses whether or not the tokenDstMacAddr and the MAC address of its own node

match when the token-PDU is received and, if so, assesses that the token-PDU is addressed to its own node and becomes the node that holds the token.

5.3.7.3 tokenSeqNumber

This field contains the sequence number of the token-PDU. The value range is 1..255.

The transmission control manager assigns a different sequence number for each token passing. Each node discards the second and subsequent token-PDUs having the same tokenSeqNumber values when token-PDUs having the same tokenSeqNumber are received. In nodes other than the transmission control manager, the values shall not be changed. In nodes other than the transmission control manager, the token-PDU having the same value as the tokenSeqNumber value of the token-PDU received is transmitted.

5.3.7.4 reserved1

This field is reserved for future use. The value of each octet is 0x00.

5.3.7.5 tokenHopCounter

This field contains the token passing counter value.

The tokenHopCounter of the token-PDU to be transmitted by the transmission control manager, which has become the token holding node, shall be 1. A node other than the transmission control manager that has become the token holding node shall add 1 to the tokenHopCounter of the received token-PDU and then perform transmission.

5.3.7.6 traAvailHopCounter

This field specifies the minimum value for the token passing counter.

The token holding node uses this field and the tokenHopCounter to assess whether or not transient transmission is possible. The token holding node can perform transient transmission when tokenHopCounter ≥ traAvailHopCounter and traAllows > 0.

After token passing begins, the transmission control manager, which becomes the token holding node for the first time, transmits the token-PDU that specifies the first node to perform transient transmission in the traAvailHopCounter. When the transmission control manager becomes the token holding node for the second and subsequent times, the transmission control manager transmits the token-PDU that sets the same value as the traLastHopCounter of the token-PDU received in the previous token passing in the traAvailHopCounter of the token-PDU.

A node other than the transmission control manager that has become the token holding node transmits the token-PDU that has the same value as the traAvailHopCounter value of the received token-PDU.

5.3.7.7 traLastHopCounter

This field indicates the last transient transmission token passing counter. This is the tokenHopCounter of the token holding node for which the traAllows has been set to 0 as a result of transient transmission execution.

When the token holding node performs transient transmission and traAllows changes to 0, the node shall transmit the token-PDU in which the tokenHopCounter value of the received token-PDU is set in the traLastHopCounter.

5.3.7.8 traAllows

This field specifies the number of times transient transmission is allowed. The number is equivalent to the number of transient transmission frames that may be transmitted by the token holding node.

The token holding node shall not transmit the transientData-PDU when traAllows of the received token-PDU is 0. After transmission of the transientData-PDU, the token holding node shall set into traAllows of the token-PDU to be transmitted a value equivalent to the value minus the number of times the transientData-PDU was transmitted.

5.3.7.9 reserved2

This field is reserved for future use. The value of each octet is 0x00.

5.3.7.10 dcs

Refer to 5.3.2.7.

5.3.8 F-Measure-PDU

5.3.8.1 falArHeader

Refer to 5.3.1.

5.3.8.2 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.8.3 dcs

Refer to 5.3.2.7.

5.3.9 F-Offset-PDU

5.3.9.1 FalArHeader

Refer to 5.3.1.

5.3.9.2 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.9.3 SyncOffset

This field contains the measured transmission path delay value.

5.3.9.4 reserved2

This field is reserved for future use. The value of each octet is 0x00.

5.3.9.5 dcs

Refer to 5.3.2.7.

5.3.10 F-Update-PDU

5.3.10.1 falArHeader

Refer to 5.3.1.

5.3.10.2 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.10.3 syncOffset

Refer to 8.3.9.3.

5.3.10.4 reserved2

This field is reserved for future use. The value of each octet is 0x00.

5.3.10.5 dcs

Refer to 5.3.2.7.

5.3.11 F-MyStatus-PDU

5.3.11.1 falArHeader

Refer to 5.3.1.

5.3.11.2 seqNumber

This field contains a sequential number representing the transmission order of the myStatus-PDU and f-cyclicData-PDU to be transmitted once by the node. Each bit has the following meaning:

Bit 7:	Last frame identification. 0 indicates that frames follow, 1 indicates that no frames follow.
Bits 6..0:	Sequential number. The value of the myStatus-PDU is 0.

5.3.11.3 netNumber

This field contains the network number of the node. The range is 1..239.

5.3.11.4 masterCmd

Each bit has the following meaning:

Bits 15..12:	Reserved for future use. The value is 0.
Bit 11:	Indicates the requirement of the delivery of the address table. 0 indicates delivery required. This field is used only in the myStatus-PDU. For master stations this value is 0.
Bit 10..8:	Sequence number for the transient1-PDU for address table delivery. This field is used only in the myStatus-PDU which is transmitted by the slave stations excluding local stations. For master and local stations this value is 0.
Bit 7:	Reserved for future use. The value is 0.
Bit 6:	Reserved for future use. The value is 0.
Bit 5:	Application error status. 0 indicates no error, and 1 indicates an error exists. Used only with the myStatus-PDU transmitted by the master station. Not used by and set to 0 for slave stations.
Bit 4:	Application operation status. 0 indicates stopped, and 1 indicates running. Used only with the myStatus-PDU transmitted by the master station. The slave station uses 0.
Bit 3:	Cyclic operation instruction. 0 indicates a run instruction, and 1 indicates a stop instruction. Used only with the MyStatus-PDU transmitted by the master station. Used by the master station to assign a cyclic run instruction to slave stations. A slave station that receives the stop instruction discards received cyclicDataRWw-PDUs and cyclicDataRY-PDUs, and stops transmission of

cyclicDataRWr-PDUs and cyclicDataRX-PDUs. Slave stations do not use this field.

Bits 2..1: Reserved for future use. The value is 0.

Bit 0: Master station identification. 0 indicates that the node is not a master station, and 1 indicates that the node is a master station.

5.3.11.5 cyclicStatus

When bit 8 is 1 and bits 9 and 10 are 0, cyclic transmission and reception are performed. Each bit has the following meaning:

Bit 15: Cyclic operation instruction status, node specific. 0 indicates run setting, and 1 indicates stop setting. It is used only with the myStatus-PDU transmitted by the slave station. The slave station reflects bit 19 of the received parameter-PDU cmd. The master station does not use this field.

Bit 14: Cyclic operation instruction status, all nodes. 0 indicates operation setting, and 1 indicates stop setting. It is used only with the myStatus-PDU transmitted by the slave station. The slave station reflects bit 3 of the received myStatus-PDU masterCmd from the master station. The master station does not use this field.

Bit 13: Reserved node setting status. 0 indicates that the node is not a reserved node, and 1 indicates that the node is a reserved node. It is used only with the myStatus-PDU transmitted by the slave station. The slave station reflects the setting value of bit 16 of the received parameter-PDU cmd. The master station does not use this field.

Bit 12: Node number setting status. 0 indicates in range, and 1 indicates out of range. It is used only with the myStatus-PDU transmitted by the slave station. The slave station reflects bit 18 of the received myStatus-PDU cmd. The master station does not use this field.

Bit 11: Cyclic transmission parameter confirmation status. 0 indicates confirmed, and 1 indicates confirmation in progress.

Bits 10..8: Cyclic transmission parameter hold status. The values used are in accordance with Table 37.

Bit 7: Stopped state for own reasons. 0 indicates not stopped, 1 indicates cyclic transmission stopped for reasons other than the above.

NOTE For example, when a stop request is received from the application layer or at startup.

Bit 6: Disconnection status. 0 indicates no disconnection, and 1 indicates disconnection. The master station does not use this field.

Bit 5: Reserved for future use. Uses 0

Bit 4: Node type/number invalid status. 0 indicates valid, and 1 indicates invalid. It is used only with the myStatus-PDU transmitted by the slave station. Not used by and set to 0 for the master station.

Bit 3: Master station duplication status. 0 indicates no duplication, and 1 indicates duplication. Used only with the myStatus-PDU transmitted by the slave station. The slave station does not use this field.

Bit 2: Node number duplication status. 0 indicates no duplication, and 1 indicates duplication. It is used only with the myStatus-PDU transmitted by the slave station. The slave station reflects bit 17 of the received parameter-PDU cmd. The master station does not use this field.

Bit 1: Cyclic transmission continuation not possible error. 0 indicates no error exists, and 1 indicates an error exists that makes it no longer possible to continue cyclic transmission.

NOTE For example, a hardware error or firmware error of its own node.

Bit 0: Reserved for future use. The value is 0.

Table 37 – Cyclic transmission parameter hold status

Value	Description
001b	Received. Parameter normal.
010b	Not received or ID mismatch.
011b	Confirmation in progress.
100b	Received. Parameter error.

5.3.11.6 nodeStatus

Each bit has the following meaning:

Bits 15..12: For future use. The value of bit 15 and bits 13..12 are 0.

Bits 11..10: Detailed application error status. The values are defined in Table 39.

Bits 9..8: Detailed application operation status. The master station and local station use the values in accordance with Table 38. The bits are optional for slave stations (excluding the local station), and shall be set to 0 when not used.

Bit 7: Reserved for future use.

Bit 6: Transient reception enabled/disabled status. 0 indicates reception disabled, and 1 indicates reception enabled.

Bits 5..4: Reserved for future use. The value is 0.

Bit 3: Size error status. 0 indicates normal, and 1 indicates error. It is not used by and always set to 0 for the master station and local station. The slave stations (excluding the local station) set the value to error when data addressed to its own node are not included in the received CyclicDataRY-PDU or CyclicDataRWw-PDU. The value is always set to 0 for slave stations (excluding the local station) that do not have RY or RWw.

Bits 2..0: Reserved for future use. The value is 0.

Table 38 – Detailed application operation status

Value	Description
0	Detailed application operation status notification not supported
1	Application stopped
2	Application running
3	Application user does not exist

Table 39 – Error detection status

Value	Description
0	No error
1	Minor error
2	Major error
3	Severe error

5.3.11.7 errorCode

This field contains the code for errors that have occurred in the node.

5.3.11.8 portStatus

This field indicates the status of four ports, using four bits each. Bits 3..0 and bits 7..4 of the first octet indicate the status of the first port and the status of the second port, respectively, and bits 3..0 and bits 7..4 of the second octet indicate the status of the third port and the status of the fourth port, respectively. The first port is specified in portIndex. Each bit of the first octet has the following meaning:

Bits 7..6:	Reserved for future use. The value is 0.
Bits 5..4:	The values used are in accordance with Table 35.
Bits 3..2:	Reserved for future use. The value is 0.
Bits 1..0:	The values used are in accordance with Table 35.

5.3.11.9 portStatistics

This field contains the statistical information of four ports, using four bits each. Bits 3..0 and bits 7..4 of the first octet indicate the status of the first port and the status of the second port, respectively, and bits 3..0 and bits 7..4 of the second octet indicate the status of the third port and the status of the fourth port, respectively. The first port is specified in portIndex. Each bit of the first octet has the following meaning. The statistical information value of a non-existing port is 0.

Bit 7, 3:	Reserved for future use. The value is 0.
Bit 6, 2:	Indicates the presence of a token-PDU reception timing incorrect error. 0 indicates no error, and 1 indicates error. When the node receives a token-PDU addressed to its own node, begins transmission while holding the token, and then receives a token-PDU having the same tokenSeqNumber, a reception timing incorrect error occurs. The error is cleared when either a token-PDU having a different tokenSeqNumber is received or the node changes to a ChannelGroup undetermined state.
Bit 5, 1:	Indicates detection of transmission authority duplication. 0 indicates no detection, and 1 indicates detection. Detection occurs when the node is holding the token and a PDU other than the persuasion-PDU, testData-PDU, testDataAck-PDU, setup-PDU, setupAck-PDU, and token-PDU is received.
Bit 4, 0:	Indicates the presence of a reception error. 0 indicates no error, and 1 indicates error. An error is indicated when an FCS error, reception undersize error, or reception oversize error occurs.

5.3.11.10 portIndex

This field specifies the port number of the first port of portStatus and portStatistics.

5.3.11.11 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.11.12 cyclicSequenceNumber

This field is used only with the myStatus-PDU transmitted by local stations. The master station and slave station (not local station) do not use the bits, and set them to 0. The bits are defined as follows:

Bit 7:	Head identifier. 1 indicates the head of the cyclic transmission. 0 indicates that the cyclic transmission is at some midpoint.
Bits 6..0:	Sequential number divisor. For value 0, the sequential number is not divided. A divided sequential number is derived through sequential subtraction starting from the value of the number of division. For example, a divisor value of 3 yields a sequential number sequence of: 3, 2, 1.

5.3.11.13 SlaveSpfEventInfo1

This field is used in combination with slaveSpfEventInfo2 when a slave station notifies the master station of a slave-specific event that had occurred. In the myStatus-PDU transmitted by the master station, the field indicates the reception status of the slave-specific event. In the myStatus-PDU transmitted by a slave station, the field indicates the occurrence number of the slave-specific event.

When the master station transmits the field, the value used is in accordance with Table 40. The value is 0x00 at startup, and 0x01 after initialization completion. When the myStatus-PDU is received from a slave station, a check is conducted to see if the received slaveSpfEventInfo1 value has changed from the value previously received and, if so, the value changes to 0x02. Once registration of the detailed code of the slave-specific event indicated by slaveSpfEventInfo2 of the received myStatus-PDU is completed, the value changes to 0x01.

Table 40 – Slave-specific event reception status

Value	Description
0x00	Initial state
0x01	Waiting for reception
0x02	Reception/Registration in progress
0x03..0xFF	Not available for use

When a slave station transmits the field, the value is 0x01..0xFF. When slaveSpfEventInfo1 of the myStatus-PDU received from the master station is 0x01 and slaveSpfEventInfo2 has changed from the previous value, the slave station transmits new slave-specific event information to the master station. When the slave-specific event information is transmitted to the master station, the incremented number is newly assigned to the slave-specific event to be transmitted and registered. The slave station transmits the myStatus-PDU containing the occurrence number assigned to slaveSpfEventInfo1 and the detailed code of the slave-specific event to be registered in slaveSpfEventInfo2.

5.3.11.14 SlaveSpfEventInfo2

This field is used in combination with slave SpfEventInfo1 when a slave station notifies the master station of a slave specific event that has occurred.

In the myStatus-PDU transmitted by the master station, the field contains the slave-specific event reception counter value. The value is 0x0000 at startup and 0x0001 upon initialization completion. The value is incremented upon reception and registration process completion of the slave-specific event.

In the myStatus-PDU transmitted by a slave station, the value specifies the detailed code of the slave-specific event.

For the method of use, refer to 5.3.11.13.

5.3.11.15 vendorSpfNodeInfo

This field contains vendor specific node information.

5.3.11.16 dcs

Refer to 5.3.2.7.

5.3.12 F-CyclicData-PDU

5.3.12.1 Overview

The arFType indicates whether the PDU stores RWw, RY, RWr, or RX. In the following RWw, RY, RWr, and RX descriptions, an arFType of 0x82 (CyclicDataRWw-PDU) refers to RWw, 0x83 (CyclicDataRY-PDU) refers to RY, 0x84 (CyclicDataRWr-PDU) refers to RWr, and 0x85 (CyclicDataRX-PDU) refers to RX.

5.3.12.2 falArHeader

Refer to 5.3.1.

5.3.12.3 seqNumber

Refer to 5.3.11.2.

5.3.12.4 bothEndsValidity

This field indicates whether or not the first four octets and the last four octets of cycData are to be reflected in shared member. Each bit has the following meaning:

Bit 7:	Validity information of the last octet. 1 indicates valid, and 0 indicates invalid.
Bit 6:	Validity information of the second to last octet. 1 indicates valid, and 0 indicates invalid.
Bit 5:	Validity information of the third to last octet. 1 indicates valid, and 0 indicates invalid.
Bit 4:	Validity information of the fourth to last octet. 1 indicates valid, and 0 indicates invalid.
Bit 3:	Validity information of the fourth from the first octet. 1 indicates valid, and 0 indicates invalid.
Bit 2:	Validity information of the third from the first octet. 1 indicates valid, and 0 indicates invalid.
Bit 1:	Validity information of the second from the first octet. 1 indicates valid, and 0 indicates invalid.
Bit 0:	Validity information of the first octet. 1 indicates valid, and 0 indicates invalid.

5.3.12.5 cycDataSize

Each bit has the following meaning:

Bits 15..12:	Reserved for future use.
Bits 11..0:	Indicates the size of RWw, RY, RWr, or RX. Specifies the size in units of four octets. The first four octets and last four octets not to be written in bothEndsValidity are also included in the size.

5.3.12.6 offsetAddr

This field specifies the offset address from the start of RWw, Ry, RWr, or RX. It is specified by a value in units of four octets. That is, the value of bits 1..0 is 0.

5.3.12.7 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.12.8 cycData

This field contains the data of RWw, RY, RWr, or RX. When 16 octets are not filled, the value is padded with 0x00.

5.3.12.9 dcs

Refer to 5.3.2.7.

5.3.13 Transient1-PDU

5.3.13.1 falArHeader

Refer to 5.3.1.

5.3.13.2 traMsgHeader

5.3.13.2.1 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.13.2.2 seqNumber

Each bit of this field has the following meaning:

Bit 7:	Indicates whether or not the PDU is the last PDU. A value of 0 indicates that the PDU is not the last PDU. A value of 1 indicates the PDU is the last PDU.
Bits 6..0:	Indicates the number when the data are divided.

5.3.13.2.3 dataId

This field contains the identification number of transient data. The range is 0x00..0xFF. Divided transient data are given the same identification number.

5.3.13.2.4 wholeDataSize

This field specifies the amount of the transient data, in units of octets.

5.3.13.2.5 offsetAddr

This field specifies the offset address. The field is used to assemble transient data transmitted after being divided. In the first PDU, the value is 0x00. In subsequent PDUs, the position of the data with respect to the entire volume of transient data is specified by an offset address from the start.

5.3.13.2.6 dataSize

This field specifies the transient data size in octet units.

5.3.13.2.7 dataSubType

This field specifies the data sub-type. The values are specified in Table 41. Refer to 5.3.1.2.

Table 41 – dataSupType of dataType (0x07)

Value	Description
0x0000	Not applicable
0x0001	Reserved for future use

Value	Description
0x0002	System specific
0x0003..0xFFFF	Reserved for future use

5.3.13.3 data

5.3.13.3.1 Overview

This field contains the transient data. The structure and size differ according to the dataType of 5.3.1.2, and the dataSubType of 5.3.13.2.7. When the wholeDataSize of 5.3.13.2.4 exceeds 1466 octets, the transient data is of the size indicated by dataSize of 5.3.13.2.6 from the offset indicated by offsetAddr of 5.3.13.2.5. When 16 octets are not filled, the value is padded using 0x00.

5.3.13.3.2 FieldSpecificTransient

5.3.13.3.2.1 opHeader

The structure of the opHeader is shown in Table 42.

Table 42 – FieldSpecificTransient opHeader

Field	Description
command	specifies the type of field-specific command. The values used for each dataType and dataSubType are in accordance with Table 43. For the dataType, refer to 5.3.1.2, and for the dataSubType, refer to 5.3.13.2.7.
subCommand	specifies the sub-command type. The values for each command are in accordance with Table 44.
rtn	not used and contains a value of 0x0000 when the sub-command type is request. The field contains a return value when the sub-command type is response.
reserved1	reserved for future use. The value of each octet is 0x00.
destNetNumber	specifies the destination network number. 0 indicates broadcast. For node information delivery, the value 0 is used.
destNodeNumber	specifies the destination node number. 0xFFFF indicates broadcast. For node information delivery, the value is 0xFFFF.
reserved2	reserved for future use. The value of each octet is 0x00.
srcNetNumber	specifies the transmission source network number.
srcNodeNumber	specifies the transmission source node number.
reserved3	reserved for future use. The value of each octet is 0x00.

Table 43 – command (dataType: 0x07, dataSubType: 0x0002)

Value	Description
0x00	Not applicable
0x01	Deliver node information(nodeInfoDist)
0x02	Reserved for future use
0x03	Get statistical information(statistics)
0x04	Acquires detailed node information(nodeInfoDetail)
0x05..0xFF	Reserved for future use

Table 44 – subCommand type for each command type

Value of Command	Value	Description
0x01	0x00	Request
0x03	0x00	Request
	0x80	Response
0x04	0x00	Request
	0x80	Response

5.3.13.3.2.2 fSTraData

The structure of this field differs according to the dataType and dataSubType. For the data type, refer to 5.3.1.2. For the data sub type, refer to 5.3.13.2.7.

The structure of the Deliver node information is shown in Table 45.

Table 45 – Structure of Deliver node information

Field	Description
seqNumber	contains the delivery sequential number.
masterNetNumber	contains the master station network number.
masterDeviceType	contains the master station device type.
masterModelCode	contains the master station model code. The model code is vendor specific.
masterVendorCode	contains the master station vendor code. For the vendor code, refer to the Device Profile edition.
masterNodeType	contains the master station node type.
reserved1	reserved for future use.
masterMacAddress	contains the master station MAC address.
reserved2	reserved for future use.
dataNum	contains the number of node information deliveries.
message	contains the node information as specified in Table 46. The node information of the quantity indicated in dataNum is continuous.

Table 46 – Structure of Deliver node information – message

Field	Description
nodeNumber	contains the node number.
reserved1	reserved for future use. The value of each octet is 0x00.
availableFuncs	Refer to 5.3.6.13.
reserved2	reserved for future use. The value of each octet is 0x00.
netNumber	contains the network number.
deviceType	contains the device type.
modelCode	Refer to 5.3.2.5.
vendorCode	Refer to 5.3.2.4.
nodeType	contains the node type.
reserved3	reserved for future use.
macAddress	contains the MAC address.
reserved4	reserved for future use. The value of each octet is 0x00.

There are no fields for the Get statistical information request. The structure of the Get statistical information is shown in Table 47.

Table 47 – Structure of Get statistical information response

Field	Description
port1Mib1	indicates the number of HEC error frames in port 1. The number is the accumulated number since previously acquired.
port1Mib2	indicates the number of DCS/FCS error frames in port 1. The number is the accumulated number since previously acquired.
port1Mib3	indicates the number of undersize (28 octet) error frames in port 1. The number is the accumulated number since previously acquired.
port1Mib4	indicates the number of forward frames in port 1. The number is the accumulated number since previously acquired.
port1Mib5	indicates the number of frames delivered from port 1 to the upper layer. The number is the accumulated number since previously acquired.
port1Mib6	indicates the number of frames discarded due to a full forward buffer in port 1. The number is the accumulated number since previously acquired.
port1Mib7	indicates the number of frames delivered to the upper layer and discarded as a result of a full buffer in port 1. The number is the accumulated number since previously acquired.
reserved	reserved for future use. The value of each octet is 0x00.
port2Mib1	indicates the number of HEC error frames in port 2. The number is the accumulated number since previously acquired.
port2Mib2	indicates the number of DCS/FCS error frames in port 2. The number is the accumulated number since previously acquired.
port2Mib3	indicates the number of undersize (28 octet) error frames in port 2. The number is the accumulated number since previously acquired.
port2Mib4	indicates the number of forward frames in port 2. The number is the accumulated number since previously acquired.
port2Mib5	indicates the number of frames delivered from port 2 to the upper layer. The number is the accumulated number since previously acquired.
port2Mib6	indicates the number of frames discarded due to a full forward buffer in port 2. The number is the accumulated number since previously acquired.
port2Mib7	indicates the number of frames delivered to the upper layer and discarded as a result of a full buffer in port 2. The number is the accumulated number since previously acquired.
healthStatusNum	indicates the number of health status data. The value range is 0..128.
healthStatus	indicates the health status. The health status information of the number indicated in healthStatusNum is continuous. This value is vendor specific.

There are no field for the Acquisition of node details request. The fields for the Acquistion of node details response are specified in Table 48.

Table 48 – Structure of Acquisition of node details response

Field	Description
rySize	Refer to 5.3.6.8.
rwwSize	Refer to 5.3.6.9.
rxSize	Refer to 5.3.6.10.
rwrSize	Refer to 5.3.6.11.
reserved1	reserved for future use. The value of each octet is 0x00.
ports	indicates the number of ports.
tokenKeepTime	Refer to 5.3.4.7.

Field	Description
netBehaviour	Refer to 5.3.5.7.
nodeInfo	Refer to 5.3.6.2.
fwVersion	indicates the firmware version of the network.
deviceType	contains the device type.
modelCode	contains the model code.
vendorCode	contains the vendor code.
reserved2	reserved for future use. The value of each octet is 0x00.
modelName	contains the model name.
vendorName	contains the vendor name.
contInfo	contains the information flag. 1 indicates that related information is contained in the fields that follow. 0 indicates that no related information follows.
contFwVersion	contains the controller firmware version.
contDeviceType	contains the controller device type.
contModelCode	contains the controller model code.
contVendorCode	contains the controller vendor code.
reserved3	reserved for future use. The value of each octet is 0x00.
contModelName	contains the controller model name.
contVendorName	contains the controller vendor name.
contVendorSpecificInfo	contains vendor specific device information.

5.3.14 TransientAck-PDU

5.3.14.1 falArHeader

Refer to 5.3.1.

5.3.14.2 acks

This field specifies the number of ackData. The value is 1.

5.3.14.3 ackData

5.3.14.3.1 nodeNumber

This field contains the node number.

5.3.14.3.2 reserved1

This field is reserved for future use. The value of each octet is 0x00.

5.3.14.3.3 connectionInfo

This field contains the connection information of the received f-transientData-PDU. The value range is 0x01..0xFF.

5.3.14.3.4 dataSubType

When frame type is 0x22, data subtype received is set. Refer to 5.3.13.2.7. When frame type is 0x25, 0x0000 is fixed.

5.3.14.3.5 ret

This field indicates the result in response to the request of the received f-transientData-PDU. 0 indicates normal, and a value other than 0 indicates the error code when abnormal.

5.3.14.4 dcs

Refer to 5.3.2.7.

5.3.15 Transient2-PDU

5.3.15.1 falArHeader

Refer to 5.3.1.

5.3.15.2 l

This field specifies the data length from fno to data (in octets).

5.3.15.3 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.15.4 tp

Each bit of this field has the following meaning:

Bits 7..4:	Indicates the type. The value is 0.
Bits 3..0:	Indicates the sequence number.

5.3.15.5 fno

Each bit of this field has the following meaning:

Bit 7:	Indicates first frame identification. A value of 0 indicates the frame is a non-head frame. A value of 1 indicates the frame is a head frame.
Bits 6..0:	Indicates the divided frame number. A value of 0 indicates no division. A value of 1 to 7 indicates the divided frame number. The divided frame number decreases sequentially starting from the same number as the number of divisions. Example When a frame is divided into three, the divided frame numbers are sent in the order: 3, 2, 1.

5.3.15.6 dt

Each bit of this field has the following meaning:

Bit 7:	Indicates the priority. 0 indicates low, and 1 indicates high.
Bit 6:	Indicates the presence of a response frame. When the value is 0, a response frame is required. When the value is 1, it is not needed.
Bits 5-0:	Reserved for future use.

5.3.15.7 da

This field specifies the node number of the node that performs forwarding within the local network when the destination node is in a different network.

5.3.15.8 sa

This field indicates the node number of the source node.

5.3.15.9 dat

This field specifies the destination application type. The value is fixed to 0x22.

5.3.15.10 sat

This field indicates the source application type. The value is fixed to 0x22.

5.3.15.11 dmf

This field specifies the execution module destination. The values used are in accordance with Table 49.

Table 49 – Execution module specification

Value	Description
0x00	Within module
0x01	Within controller
0x02	Reserved for future use
0x03..0xFF	Within other module

5.3.15.12 smf

This field specifies the execution module source. The values used are in accordance with Table 49.

5.3.15.13 dna

This field specifies the network number of the destination node. The range of values is 0x00..0xEF and 0xFE. 0x00 indicates no specified network, and 0xFE indicates the default network.

5.3.15.14 ds

This field specifies the node number of the target destination node of the other network during forwarding.

5.3.15.15 did

Each bit of this field has the following meaning:

Bits 15..10: System specifications area. Stores the target destination node number.

Bits 9..0: Indicates the target destination identification number.

5.3.15.16 sna

This field indicates the network number of the startup source node. The range of values is 0x00..0xEF and 0xFE. 0x00 indicates no specified network, and 0xFE indicates the default network specification.

5.3.15.17 ss

This field indicates the node number of the transmission source node of the other network during forwarding.

5.3.15.18 sid

Each of the bits of this field has the following meaning:

Bits 15..10: System specifications area. Stores the startup source node number.

Bits 9..0: Indicates the startup source identification number.

5.3.15.19 l1

This field specifies the data length (in octet units) from ct to data.

5.3.15.20 ct

This field specifies the command type. The values used are in accordance with Table 50.

Table 50 – Command type

Value	Description
0x00	Not applicable
0x01..0x03	Reserved for future use
0x04	Get Memory Access Info
0x05..0x07	Reserved for future use
0x08	RUN
0x09	STOP
0x0A..0x0F	Reserved for future use
0x10	Read memory
0x11	Reserved for future use
0x12	Write memory
0x13..0x5F	Reserved for future use
0x60..0x7F	Vendor specific

5.3.15.21 rsv

Reserved for future use.

5.3.15.22 aps

Each bit of this field has the following meaning:

Bits 15..11: Indicates the task number assigned to the task. The value range is 0..255.

Bits 8..0: Indicates the identification number of the startup source application. The value range is 0..255.

5.3.15.23 rsts

This field is used for responses only. Each bit has the following meaning:

Bits15..12: Vendor definition

Bits 11..8:	Error occurrence location
Bit 7:	Error criticality. 0 Indicates warning error, and 1 indicates Major error.
Bits 6..0:	Error code

5.3.15.24 data

Refer to 5.2.12.23.

5.3.15.25 dcs

Refer to 5.3.2.7.

5.3.16 ParamCheck-PDU

5.3.16.1 falArHeader

Refer to 5.3.1.

5.3.16.2 reserved1

This field is reserved for future use. The value of the first, second, and fourth octets is 0x00. The value of bits 0..3 of the third octet is 0.

5.3.16.3 paramId

5.3.16.3.1 Overview

This field contains the parameter identification ID. A value of 0 indicates a discard parameter instruction.

5.3.16.3.2 date

Each bit of the elements of this field has the following meaning:

Bits 31..28	Day (ten's place)
Bits 27..24	Day (one's place)
Bits 23..20	Month (ten's place)
Bits 19..16	Month (one's place)
Bits 15..12	Year (ten's place)
Bits 11..8	Year (one's place)
Bits 7..4	Year (thousand's place)
Bits 3..0	Year (hundred's place)

5.3.16.3.3 timeNodeId

Each bit of the elements of this field has the following meaning:

Bits 31..24	Node number of setting source
Bits 23..20	Seconds (ten's place)
Bits 19..16	Seconds (one's place)
Bits 15..12	Minutes (ten's place)
Bits 11..8	Minutes (one's place)
Bits 7..4	Hour (ten's place)
Bits 3..0	Hour (one's place)

5.3.16.3.4 checksum

This field contains the sum-check value of the common parameters held by the master station.

5.3.16.4 reserved3

This field is reserved for future use. The value of each octet is 0x00.

5.3.16.5 dcs

Refer to 5.3.2.7.

5.3.17 Parameter-PDU

5.3.17.1 falArHeader

Refer to 5.3.1.

5.3.17.2 paramSetFlag

This field contains the parameter specification. Each bit has the following meaning:

Bits 7..3:	Reserved for future use. The value is 0.
Bit 2:	Common parameter setting. 0 indicates that the parameter is not valid, and 1 indicates that the parameter is valid.
Bit 1:	Operation command setting. 0 indicates that the parameter is not valid, and 1 indicates that the parameter is valid.
Bit 0:	Node number and network number setting. 0 indicates that the parameter is not valid, and 1 indicates that the parameter is valid.

5.3.17.3 addressOrder

5.3.17.3.1 Overview

This field is used when bit 0 of the paramSetFlag indicates a valid parameter. The field is not used and the value is 0 when the paramSetFlag indicates an invalid parameter.

5.3.17.3.2 assignedNetNumber

This field specifies the setting value of the network number. The range is 1..239.

5.3.17.3.3 assignedNodeNumber

This field specifies the setting value of the node number.

5.3.17.4 cmdOrder

5.3.17.4.1 Overview

This field is used when bit 1 of the paramSetFlag indicates a valid parameter. The field is not used and the value is 0 when the paramSetFlag indicates an invalid parameter.

5.3.17.4.2 cmd

Each bit of the elements of this field has the following meaning:

Bits 23..20:	Reserved for future use. The value is 0.
Bit 19:	Cyclic transmission stop instruction caused by master station assessment. 0 indicates enabled, and 1 indicates disabled. The master station provides this instruction to slave stations. The slave station reflects the instruction status in

bit 15 of cyclicStatus of the myStatus-PDU.

Bit 18: Cyclic transmission stop instruction caused by invalid node number. 0 indicates enabled, and 1 indicates disabled. The master station provides this instruction to slave stations. The slave station reflects the instruction status in bit 3 of cyclicStatus of the myStatus-PDU.

Bit 17: Cyclic transmission stop instruction caused by duplicate node number. 0 indicates enabled, and 1 indicates disabled. The master station provides this instruction to slave stations. The slave station reflects the instruction status in bit 13 of cyclicStatus of the myStatus-PDU.

Bit 16: Reserved node setting. 0 indicates that the node is a reserved node, and 1 indicates that the node is not a reserved node. The master station uses this setting when specifying a slave station as a reserved node. A slave station specified as a reserved node does not perform cyclic transmission. The slave station reflects the status of this setting in bit 2 of cyclicStatus of the myStatus-PDU.

Bits 15..5: Reserved for future use. The value is 0.

Bit 4: Indicates if the node type is valid as determined by node type field comparison between the nodeType field and the station's own station type. 0 indicates the node type is invalid and cycle transmission is not performed.

Bits 3..0: Reserved for future use. The value is 0.

5.3.17.4.3 nodeType

Refer to 5.3.1.3.3.

5.3.17.5 cyclicParameter

5.3.17.5.1 Overview

This field is used when bit 2 of the paramSetFlag indicates set. The field is not used and is 0 when the paramSetFlag indicates not set.

5.3.17.5.2 paramId

Refer to 5.3.16.3.

5.3.17.5.3 reserved1

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.4 masterStatus

Each bit of the elements of this field has the following meaning:

Bits 15..1: Reserved for future use. The value is 0.

Bit 0: Node unit guarantee function. 0 indicates that the node unit is not guaranteed, and 1 indicates that the node unit is guaranteed. This is used by the master station to notify the slave station as to whether or not the master station has a node unit guarantee function.

5.3.17.5.5 rySeqNumber

Each bit of the elements of this field has the following meaning:

Bit 7: Reserved for future use. The value is 0.

Bits 6..0: Sequential number of cyclicDataRY-PDU to be received. The range of values is 1..127.

5.3.17.5.6 ryBothEndsValidity

This field indicates whether or not to reflect into shared memory the first four octets and the last four octets of the area specified by ryDataSize and ryOffset from among the cycData of the cyclicDataRY-PDU having the sequential number specified by rySeqNumber. When ryDataSize and ryOffset are specified so that the data cross over multiple cyclicDataRY-PDUs, the last four octets are included in the last cyclicDataRY-PDU.

Each bit has the following meaning:

Bit 7:	Enabled information of the last octet. 1 indicates enabled, and 0 indicates disabled.
Bit 6:	Enabled information of the second octet from the end. 1 indicates enabled, and 0 indicates disabled.
Bit 5:	Enabled information of the third octet from the end. 1 indicates enabled, and 0 indicates disabled.
Bit 4:	Enabled information of the fourth octet from the end. 1 indicates enabled, and 0 indicates disabled.
Bit 3:	Enabled information of the fourth octet from the start. 1 indicates enabled, and 0 indicates disabled.
Bit 2:	Enabled information of the third octet from the start. 1 indicates enabled, and 0 indicates disabled.
Bit 1:	Enabled information of the second octet from the start. 1 indicates enabled, and 0 indicates disabled.
Bit 0:	Enabled information of the initial octet. 1 indicates enabled, and 0 indicates disabled.

5.3.17.5.7 ryDataSize

This field specifies the size of the data to be reflected in shared memory from within the cycData of the cyclicDataRY-PDU specified in rySeqNumber, from the address specified in ryOffset. The value is specified in units of four octets. The initial four octets and last four octets specified in ryBothEndsValidity are also included in the size.

Depending on the values of ryDataSize and ryOffset, the value reflected in shared memory using the cyclicDataRY-PDU specified in rySeqNumber as the start may cross over multiple cyclicDataRY-PDUs.

5.3.17.5.8 ryOffset

This field specifies the start of the area within the cycData of the cyclicDataRY-PDU to be reflected in shared memory, in octets from the start of cycData. The value is specified in units of four octets.

Depending on the values of ryDataSize and ryOffset, the value reflected in shared memory using the cyclicDataRY-PDU specified in rySeqNumber as the start may cross over multiple cyclicDataRY-PDUs.

5.3.17.5.9 reserved2

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.10 rwwSeqNumber

This field is defined as follows:

Bit 7:	Reserved for future use. The value is 0.
Bits 6..0:	Sequential number of the cyclicDataRWw-PDU to be received. The range of values is 1..127.

5.3.17.5.11 reserved3

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.12 rwwDataSize

This field specifies the size within the cycData of the cyclicDataRY-PDU having the sequential number specified in rwwSeqNumber to be reflected in shared memory from the address specified by rwwOffset. The value is specified in units of four octets.

Depending on the values of rwwDataSize and rwwOffset, the value reflected in shared memory using the cyclicDataRWw-PDU specified in rwwSeqNumber as the start may cross over multiple cyclicDataRWw-PDUs.

5.3.17.5.13 rwwOffset

This field specifies the start of the area to be reflected in shared memory from within the cycData of the cyclicDataRWw-PDU having the sequential number specified in rwwSeqNumber, based on an offset from the start of cycData. The value is specified in units of four octets.

Depending on the values of rwwDataSize and rwwOffset, the value reflected in shared memory using the cyclicDataRWw-PDU specified in rwwSeqNumber as the start may cross over multiple cyclicDataRWw-PDUs.

5.3.17.5.14 reserved4

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.15 rxBothEndsValidity

This field specifies whether or not the first four octets and the last four octets of the area specified by rxDataSize and rxOffset from among the cycData of the cyclicDataRX-PDU constitute the area reflected from shared memory.

Each bit has the following meaning:

Bit 7:	Enabled information of the last octet. 1 indicates enabled, and 0 indicates disabled.
Bit 6:	Enabled information of the second octet from the end. 1 indicates enabled, and 0 indicates disabled.
Bit 5:	Enabled information of the third octet from the end. 1 indicates enabled, and 0 indicates disabled.
Bit 4:	Enabled information of the fourth octet from the end. 1 indicates enabled, and 0 indicates disabled.
Bit 3:	Enabled information of the fourth octet from the start. 1 indicates enabled, and 0 indicates disabled.
Bit 2:	Enabled information of the third octet from the start. 1 indicates enabled, and 0 indicates disabled.
Bit 1:	Enabled information of the second octet from the start. 1 indicates enabled, and 0 indicates disabled.
Bit 0:	Enabled information of the initial octet. 1 indicates enabled, and 0 indicates disabled.

5.3.17.5.16 rxDataSize

This field specifies the size of the data of the cycData of the cyclicDataRY-PDU that was reflected from shared memory using the address specified in rxOffset as the start. The first four octets and last four octets specified in rxBothEndsValidity are also included in the size.

5.3.17.5.17 rxOffset

This field specifies the start of the area within the cycData of the cyclicDataRX-PDU that was reflected from shared memory, in octets from the start of cycData. The value is specified in units of four octets. The value of bits 1..0 is 0.

5.3.17.5.18 reserved5

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.19 rwrDataSize

This field indicates the size within the cycData of the cyclicDataRY-PDU that was reflected from shared memory using the address specified by rwrOffset as the start. The value is specified in units of four octets.

5.3.17.5.20 rwrOffset

This field specifies the start of the area that was reflected from shared memory from within the cycData of the cyclicDataRWr-PDU, based on an offset from the start of cycData. The value is specified in units of four octets. The value of bits 1..0 is 0.

5.3.17.5.21 reserved6

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.22 masterWatchTimer

This field specifies the setting value of the MasterWatchTimer.

Bit 15:	Reserved for future use. 0 is used.
Bits 14..0:	Setting value of MasterWatchTimer. The range of values is 1..32767. The unit is 400µs.

5.3.17.5.23 reserved7

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.24 cmRyBothEndsValidity

This field specifies whether or not to reflect into shared memory the first four octets and the last four octets of the area specified by cmRyDataSize and cmRyOffset within RY transmitted by the master station. The field is used only with the local station. When cmRyDataSize and cmRyOffset are specified so that the data cross over multiple cyclicDataRY-PDUs, the last four octets are included in the last cyclicDataRY-PDU.

Each bit has the following meaning:

Bit 7:	Enabled information of the last octet. 1 indicates enabled, and 0 indicates disabled.
Bit 6:	Enabled information of the second octet from the end. 1 indicates enabled, and 0 indicates disabled.
Bit 5:	Enabled information of the third octet from the end. 1 indicates enabled, and 0 indicates disabled.
Bit 4:	Enabled information of the fourth octet from the end. 1 indicates enabled, and 0 indicates disabled.
Bit 3:	Enabled information of the fourth octet from the start. 1 indicates enabled, and 0 indicates disabled.
Bit 2:	Enabled information of the third octet from the start. 1 indicates enabled, and 0

indicates disabled.

Bit 1: Enabled information of the second octet from the start. 1 indicates enabled, and 0 indicates disabled.

Bit 0: Enabled information of the initial octet. 1 indicates enabled, and 0 indicates disabled.

5.3.17.5.25 cmRyDataSize

This field specifies the size of the data to be reflected in shared memory from within RY transmitted by the master station, from the address specified in cmRyOffset. The first four octets and last four octets specified in cmRyBothEndsValidity are also included in the size. This field is used by the local station only.

5.3.17.5.26 cmRyOffset

This field specifies the start of the area within RY transmitted by the master station to be reflected in shared memory, in octets from the start of RY. The value is specified in units of four octets. This field is used by the local station only.

5.3.17.5.27 reserved8

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.28 cmRwwDataSize

This field specifies the size within RWw transmitted by the master station to be reflected in shared memory from the address specified by cmRwwOffset. The value is specified in units of four octets. The field is used by the local station only.

5.3.17.5.29 cmRwwOffset

This field specifies the start of the area to be reflected in shared memory from within RWw transmitted by the master station, based on an offset from the start of RWw. The value is specified in units of four octets. The field is used by the local station only.

5.3.17.5.30 reserved9

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.31 cmRxBothEndsValidity

This field specifies whether or not the first four octets and the last four octets of the area specified by cmRxDataSize and cmRxOffset within RX to be received by the master station from a slave station and constructed on the master station is to be reflected in shared memory. The field is used by the local station only. When cmRyDataSize and cmRyOffset are specified so that the data cross over multiple cyclicDataRX-PDUs, the last four octets are included in the last cyclicDataRX-PDU.

Each bit has the following meaning:

Bit 7: Enabled information of the last octet. 1 indicates enabled, and 0 indicates disabled.

Bit 6: Enabled information of the second octet from the end. 1 indicates enabled, and 0 indicates disabled.

Bit 5: Enabled information of the third octet from the end. 1 indicates enabled, and 0 indicates disabled.

Bit 4: Enabled information of the fourth octet from the end. 1 indicates enabled, and 0 indicates disabled.

Bit 3: Enabled information of the fourth octet from the start. 1 indicates enabled, and

0 indicates disabled.

Bit 2: Enabled information of the third octet from the start. 1 indicates enabled, and 0 indicates disabled.

Bit 1: Enabled information of the second octet from the start. 1 indicates enabled, and 0 indicates disabled.

Bit 0: Enabled information of the initial octet. 1 indicates enabled, and 0 indicates disabled.

5.3.17.5.32 cmRxDataSize

This field specifies the size of the data to be reflected in shared memory from within RX to be received by the master station from a slave station and constructed on the master station, from the address specified in cmRxyOffset. The first four octets and last four octets specified in cmRxBothEndsValidity are also included in the size. This field is used by the local station only.

5.3.17.5.33 cmRxOffset

This field specifies the start of the area to be reflected in shared memory from within RX received by the master station from a slave station and to be constructed on the master station, in octets from the start of RX. The field is used by the local station only. The value is specified in units of four octets. The value of bits 1..0 is 0.

5.3.17.5.34 reserved10

This field is reserved for future use. The value of each octet is 0x00.

5.3.17.5.35 cmRwrDataSize

This field specifies the size of the data to be reflected in shared memory from within RWr to be received by the master station from a slave station and constructed on the master station, from the address specified by cmRwrOffset. The field is used by the local station only.

5.3.17.5.36 cmRwrOffset

This field specifies the start of the area to be reflected in shared memory from within RWr to be received by the master station from a slave station and constructed on the master station, based on an offset from the start of RWr. This field is used by the local station only. The value is specified in units of four octets. The value of bits 1..0 is 0.

5.3.17.6 dcs

Refer to 5.3.2.7.

5.3.18 Timer-PDU

5.3.18.1 falArHeader

Refer to 5.3.1.

5.3.18.2 time

This field contains the timer value in units of 15,258 789 062 5 µs, using January 1, 2000, 00:00:00 as the reference point.

5.3.18.3 reserved

This field is reserved for future use. The value of each octet is 0x00.

5.3.18.4 dcs

Refer to 5.3.2.7.

6 Structure of the FAL protocol state machine

The FAL protocol state machine consists of three protocol state machines as shown in Figure 17. A protocol machine consists of, in the order from the data link layer side, data link layer mapping protocol machine (DMPM), application relationship protocol machine (ARPM), and FAL service protocol machine (FSPM).

The role of FSPM is to receive service primitives from FAL users and convert the primitives to internal primitives, select an ARPM state machine, and receive the internal primitives from ARPM and convert the primitives to FA service primitives ARPM and transmit them to FAL users.

The role of ARPM is to convert services primitives between ARPM and DMPM.

The role of DMPM is the mapping into the data link layer.

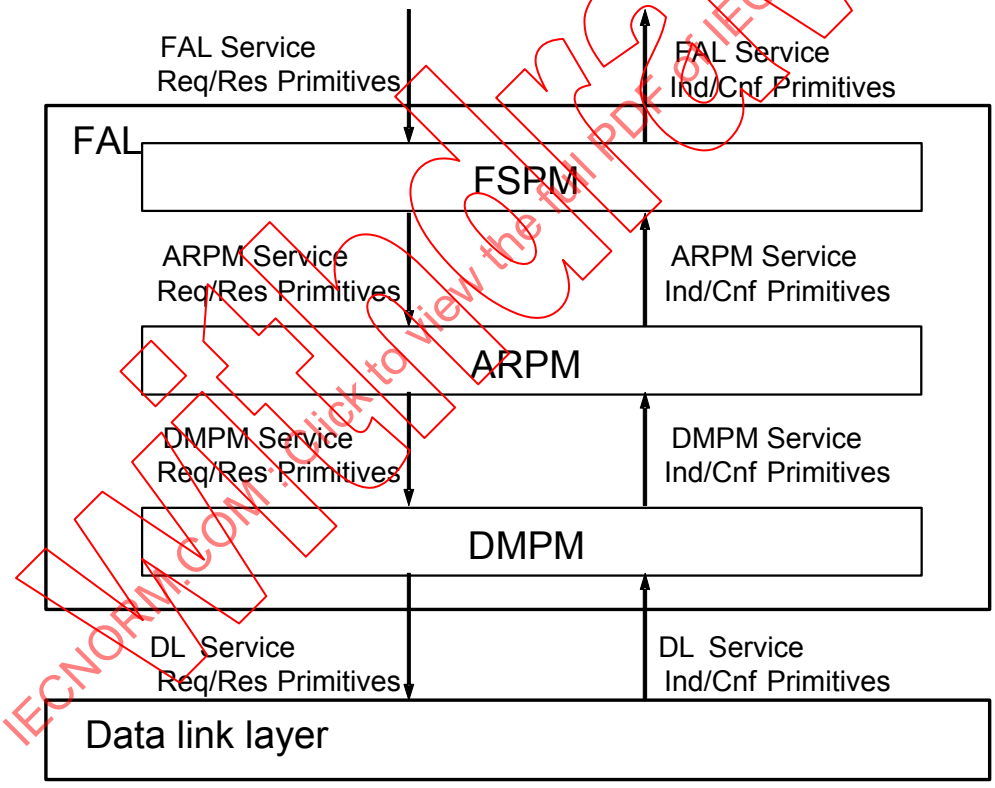


Figure 17 – Relationships between protocol machines

7 FAL service protocol machine (FSPM)

7.1 Overview

The FSPM provides an interface to FAL users. It performs the mapping between FAL user services and FAL internal services.

7.2 FSPM type C

7.2.1 Overview

The FSPM consists of three protocol machines: Cyclic data, Acyclic data and Management. The relationship between protocol machines is shown in Figure 18.

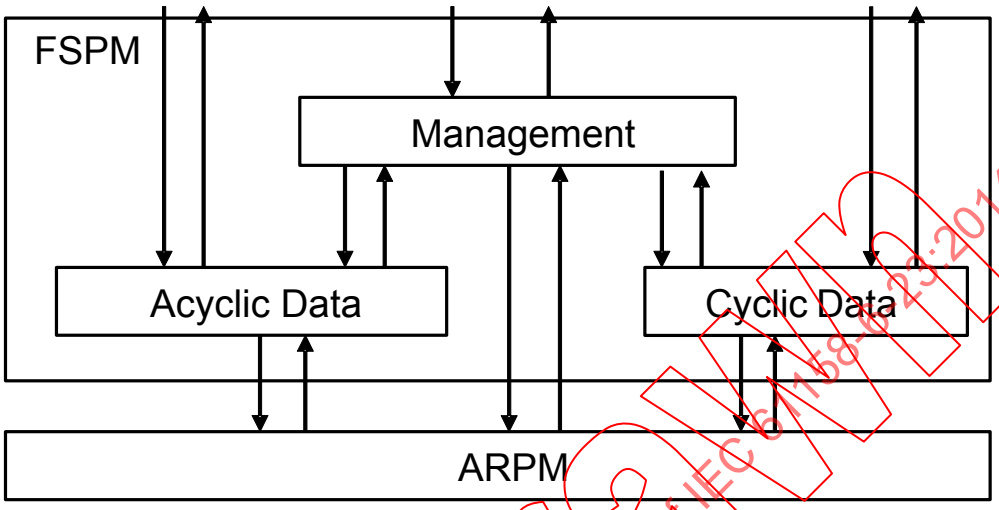


Figure 18 – Structure of FSPM C

The following primitives are issued from the FAL user to the FSPM.

Write Cyclic Data.req
Read Cyclic Data.req
Send Acyclic Data.req
Request Acyclic Data.req
Request Acyclic Data.rsp
Get Attribute.req
Get Attribute.rsp
Set Attribute.req
Set Attribute.rsp

The following primitives are issued from the FSPM to the FAL user.

Read Cyclic Data.cnf
Get Attribute.ind
Get Attribute.cnf
Set Attribute.ind
Set Attribute.cnf
Send Acyclic Data.ind
Request Acyclic Data.ind
Request Acyclic Data.cnf

7.2.2 FSPM

7.2.2.1 Cyclic data

Details of Cyclic data state machine are shown in Table 51.

Table 51 – Cyclic data state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	Write Cyclic Data.req => Update BitCM Data and WordCM Data; CT Update.req	ACTIVE
2	ACTIVE	CT Update.ind => Update BitCM Data or WordCM Data.	ACTIVE
3	ACTIVE	Read Cyclic Data.req => Read Cyclic Data.cnf(BitCM Data, WordCM Data)	ACTIVE

7.2.2.2 Acyclic data

Details of Acyclic Data state machine are shown in Table 52.

Table 52 – Acyclic data state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	Send Parameter 1.req => AC Send.req	ACTIVE
2	ACTIVE	AC Send.ind(Data) / Command == Send Parameter 1 => Send Parameter 1.ind	ACTIVE
3	ACTIVE	Send Parameter 2.req => AC Send.req	ACTIVE
4	ACTIVE	AC Send.ind(Data) / Command == Send Parameter 2 => Send Parameter 2.ind	ACTIVE
5	ACTIVE	Get System Info.req => AC Send.req	ACTIVE
6	ACTIVE	AC Send.ind(Data) / Command == Get System Info => Get System Info.ind	ACTIVE
7	ACTIVE	Get System Info.rsp => AC Send.rsp	ACTIVE
8	ACTIVE	AC Send.cnf(Data) / Command == Get System Info => Get System Info.cnf	ACTIVE
9	ACTIVE	Get Memory Access Info.req => AC Send.req	ACTIVE
10	ACTIVE	AC Send.ind(Data) / Command == Get Memory Access Info => Get Memory Access Info.ind;	ACTIVE
11	ACTIVE	Get Memory Access Info.rsp => AC Send.rsp	ACTIVE
12	ACTIVE	AC Send.cnf(Data) / Command == Get Memory Access Info =>	ACTIVE

#	Current state	Event/condition => action	Next state
		Get Memory Access Info.cnf	
13	ACTIVE	Run.req => AC Send.req	ACTIVE
14	ACTIVE	AC Send.ind(Data) / Command == Run => Run.ind	ACTIVE
15	ACTIVE	Run.rsp => AC Send.rsp	ACTIVE
16	ACTIVE	AC Send.cnf(Data) / Command == Run => Run.cnf	ACTIVE
17	ACTIVE	Stop.req => AC Send.req	ACTIVE
18	ACTIVE	AC Send.ind(Data) / Command == Stop => Stop.ind	ACTIVE
19	ACTIVE	Stop.rsp => AC Send.rsp	ACTIVE
20	ACTIVE	AC Send.cnf(Data) / Command == Stop => Stop.cnf	ACTIVE
21	ACTIVE	Line Test.req => AC Send.req	ACTIVE
22	ACTIVE	AC Send.ind(Data) / Command == Line Test => Line Test.ind	ACTIVE
23	ACTIVE	Line Test.rsp => AC Send.rsp	ACTIVE
24	ACTIVE	AC Send.cnf(Data) / Command == Line Test => Line Test.cnf	ACTIVE
25	ACTIVE	Read Memory.req => AC Send.req	ACTIVE
26	ACTIVE	AC Send.ind(Data) / Command == Read Memory => Read Memory.ind	ACTIVE
27	ACTIVE	Read Memory.rsp => AC Send.rsp	ACTIVE
28	ACTIVE	AC Send.cnf(Data) / Command == Read Memory => Read Memory.cnf	ACTIVE
29	ACTIVE	Write Memory.req => AC Send.req	ACTIVE
30	ACTIVE	AC Send.ind(Data) / Command == Write Memory =>	ACTIVE

#	Current state	Event/condition => action	Next state
		Write Memory.ind	
31	ACTIVE	Write Memory.rsp => AC Send.rsp	ACTIVE
32	ACTIVE	AC Send.cnf(Data) / Command == Write Memory => Write Memory.cnf	ACTIVE

7.2.2.3 Management

Details of Management state machine are shown in Table 53.

Table 53 – Management state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	Get Attribute.req => Get Attribute.ind	ACTIVE
2	ACTIVE	Get Attribute.rsp => Get Attribute.cnf	ACTIVE
3	ACTIVE	Set Attribute.req => Set Attribute.ind	ACTIVE
4	ACTIVE	Set Attribute.rsp => Set Attribute.cnf	ACTIVE

7.3 FSPM type F

7.3.1 Overview

The FSPM consists of five protocol machines: Cyclic data, Acyclic data, Management, Synchronization and Measurement. The relationship between protocol machines is shown in Figure 19. The continuous line represents a service issue, and the dashed line represents a linkage between protocol machines using parameters and others.

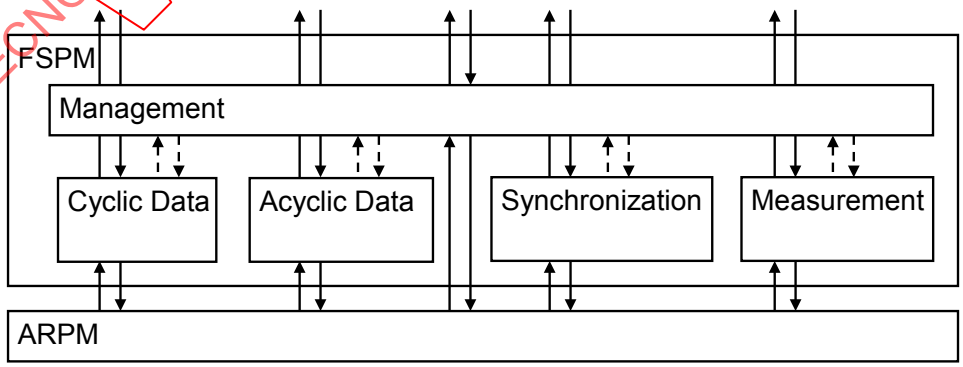


Figure 19 – Structure of FSPM F

The following primitives are issued from the FAL user to the FSPM.

RX_Ld.req
RX_Set.req
RX_Reset.req
RX_Read.req
RX_Write.req
RY_Ld.req
RY_Set.req
RY_Reset.req
RY_Read.req
RY_Write.req
RWr_Ld.req
RWr_Set.req
RWr_Reset.req
RWr_Read.req
RWr_Write.req
RWw_Ld.req
RWw_Set.req
RWw_Reset.req
RWw_Read.req
RWw_Write.req
Get Attribute.req
Set Attribute.req
Get Memory Access Info.req
Get Memory Access Info.rsp
Run.req
Run.rsp
Stop.req
Stop.rsp
Read Memory.req
Read Memory.rsp
Write Memory.req
Write Memory.rsp
Vendor Command.req
Vendor Command.rsp
Distribute Node Info.req
Get Statistics.req
Get Statistics.rsp
Get Node Info Detail.req
Get Node Info Detail.rsp
AC Data.req
AC Data.rsp
AC Data ND.req
AC Data ND.rsp

Start Measure.req
Get Offset.req

The following primitives are issued from the FSPM to the FAL user.

RX_Ld.cnf
RX_Read.cnf
RY_Ld.cnf
RY_Read.cnf
RWr_Ld.cnf
RWr_Read.cnf
RWw_Ld.cnf
RWw_Read.cnf
Get Attribute.cnf
Set Attribute.cnf
Get Memory Access Info.ind
Get Memory Access Info.cnf
Run.ind
Run.cnf
Stop.ind
Stop.cnf
Read Memory.ind
Read Memory.cnf
Write Memory.ind
Write Memory.cnf
Vendor Command.ind
Vendor Command.cnf
Distribute Node Info.ind
Get Statistics.ind
Get Statistics.cnf
Get Node Info Detail.ind
Get Node Info Detail.cnf
AC Data.ind
AC Data.cnf
AC Data ND.ind
AC Data ND.cnf
Synchroous Triger.ind
Start Measure.cnf
Get Offset.cnf

7.3.2 FSPM

7.3.2.1 Cyclic data

Details of Cyclic data state machine are shown in Table 54.

Table 54 – Cyclic data state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	Ld.req(Type, Address) => Ld.cnf(Data)	ACTIVE
2	ACTIVE	Set.req(Type, Address) => Updates RX, RY, RY, RWr, or RWw specified in Type. CT Update.req(Type, Address, 1, 1)	ACTIVE
3	ACTIVE	Reset.req(Type, Address) => Updates RX, RY, RY, RWr, or RWw specified in Type. CT Update.req(Type, Address, 1, 1)	ACTIVE
4	ACTIVE	Read.req(Type, Address, Size) => Read.cnf(Data)	ACTIVE
5	ACTIVE	Write.req(Type, Address, Size, Data) => Updates RX, RY, RY, RWr, or RWw specified in Type. CT Update.req(Type, Address, Size, Data)	ACTIVE
11	ACTIVE	CT Update.ind(Type, Offset, Size, Data) => Updates RX, RY, RY, RWr, or RWw specified in Type.	ACTIVE

7.3.2.2 Acyclic data

Details of Acyclic Data state machine are shown in Table 55.

Table 55 – Acyclic data state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	Get Memory Access Info.req => AC Send.req	ACTIVE
2	ACTIVE	AC Send.ind(Data) / Command == Get Memory Access Info => Get Memory Access Info.ind;	ACTIVE
3	ACTIVE	Get Memory Access Info.rsp => AC Send.rsp	ACTIVE
4	ACTIVE	AC Send.cnf(Data) / Command == Get Memory Access Info => Get Memory Access Info.cnf	ACTIVE
5	ACTIVE	Run.req => AC Send.req	ACTIVE
6	ACTIVE	AC Send.ind(Data) / Command == Run => Run.ind	ACTIVE
7	ACTIVE	Run.rsp => AC Send.rsp	ACTIVE
8	ACTIVE	AC Send.ind(Data) / Command == Run && Request type == Server Response	ACTIVE

#	Current state	Event/condition => action	Next state
		=> Run.cnf	
9	ACTIVE	Stop.req => AC Send.req	ACTIVE
10	ACTIVE	AC Send.ind(Data) / Command == Stop && Request type == Client Response => Stop.ind	ACTIVE
11	ACTIVE	Stop.rsp => AC Send.rsp	ACTIVE
12	ACTIVE	AC Send.cnf(Data) / Command == Stop && Request type == Server Response => Stop.cnf	ACTIVE
13	ACTIVE	Read Memory.req => AC Send.req	ACTIVE
14	ACTIVE	AC Send.ind(Data) / Command == Read Memory && Request type == Client Request => Read Memory.ind	ACTIVE
15	ACTIVE	Read Memory.rsp => AC Send.rsp	ACTIVE
16	ACTIVE	AC Send.ind(Data) / Command == Read Memory && Request type == Server Response => Read Memory.cnf	ACTIVE
17	ACTIVE	Wirte Memory.req => AC Send.req	ACTIVE
18	ACTIVE	AC Send.ind(Data) / Command == Write Memory && Request type == Client Request => Write Memory.ind	ACTIVE
19	ACTIVE	Write Memory.rsp => AC Send.rsp	ACTIVE
20	ACTIVE	AC Send.cnf(Data) / Command == Write Memory && Request type == Server Response => Write Memory.cnf	ACTIVE
21	ACTIVE	Vendor Command.req => AC Send.req	ACTIVE
22	ACTIVE	AC Send.ind(Data) / Command == Vendor Command && Request type == Client Request => Vendor Command.ind	ACTIVE

#	Current state	Event/condition => action	Next state
23	ACTIVE	Vendor Command.rsp => AC Send.rsp	ACTIVE
24	ACTIVE	AC Send.cnf(Data) / Command == Vendor Command && Request type == Server Response => Vendor Command.cnf	ACTIVE
25	ACTIVE	Distribute Node Info.req => AC Send.req	ACTIVE
26	ACTIVE	AC Send.ind(Data) / Command == Distribute Node Info => Distribute Node Info.ind	ACTIVE
27	ACTIVE	Get Statistics.req => AC Send.req	ACTIVE
28	ACTIVE	AC Send.ind(Data) / Command == Get Statistics && Request type == Client Request => Get Statistics.ind	ACTIVE
29	ACTIVE	Get Statistics.rsp => AC Send.rsp	ACTIVE
30	ACTIVE	AC Send.ind(Data) / Command == Get Statistics && Request type == Server Response => Get Statistics.cnf	ACTIVE
31	ACTIVE	Get Node Info Detail.req => AC Send.req	ACTIVE
32	ACTIVE	AC Send.ind(Data) / Command == Get Node Info Detail && Request type == Client Request => Get Node Info Detail.ind	ACTIVE
33	ACTIVE	Get Node Info Detail.rsp => AC Send.rsp	ACTIVE
34	ACTIVE	AC Send.ind(Data) / Command == Get Node Info Detail && Request type == Server Response => Get Node Info Detail.cnf	ACTIVE
35	ACTIVE	AC Data.req => AC Send.req	ACTIVE
36	ACTIVE	AC Data ND.req => AC Send ND.req	ACTIVE
37	ACTIVE	AC Send.ind(Data) / Command == AC Data && Request type == Client Request => AC Data.ind	ACTIVE

#	Current state	Event/condition => action	Next state
38	ACTIVE	AC Send.ind(Data) / Command == AC Send && Request type == Server Response => AC Data.cnf	ACTIVE
39	ACTIVE	AC Data.rsp => AC Send.req	ACTIVE
40	ACTIVE	AC Data ND.rsp => AC Send ND.req	ACTIVE

7.3.2.3 Management

Details of Management state machine are shown in Table 56.

Table 56 – Management state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	Get Attribute.req => Get Attribute.cnf	ACTIVE
2	ACTIVE	Set Attribute.req => Set Attribute.cnf	ACTIVE

7.3.2.4 Synchronization

Details of Synchronization state machine are shown in Table 57.

Table 57 – Synchronization state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	Syncronous Trigger Internal.ind => Syncronous Trigger.ind	ACTIVE

7.3.2.5 Measurement

Details of Measurement state machine are shown in Table 58.

Table 58 – Measurement state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	Start Measure.req => Start Measure Internal.req	ACTIVE
2	ACTIVE	Start Measure Internal.cnf(DATA) => Start Measure.cnf(DATA)	ACTIVE
3	ACTIVE	Get Offset.req => Get Offset Internal.req	ACTIVE
4	ACTIVE	Get Offset Internal.cnf(offset) => Get Offset.cnf(offset)	ACTIVE

8 Application relationship protocol machine (ARPM)

8.1 ARPM type C

8.1.1 Overview

The ARPM consists of four sub-protocols. The structure of ARPM is shown in Figure 20. The continuous line represents a service issue, and the dashed line represents a linkage between protocol machines using parameters and others.

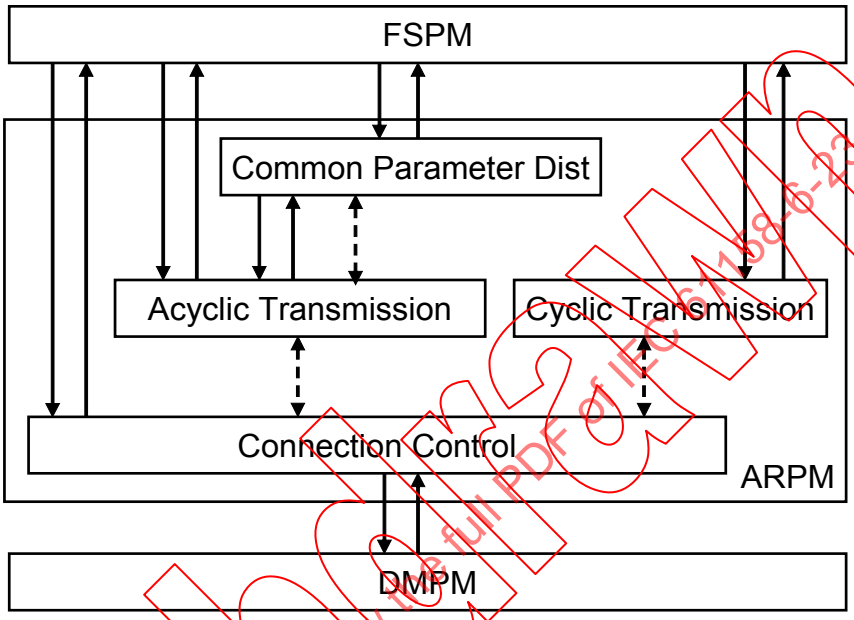


Figure 20 – Structure of ARPM C

8.1.2 Acyclic transmission

8.1.2.1 Primitive definition

The FSPM issues an AC Send.req service to Acyclic transmission. Common parameter dist issues an ACParamSend.req service to Acyclic transmission. Acyclic transmission issues an ACSend.ind service to FSPM. Acyclic transmission issues an ACParamSend.ind service to Common parameter dist.

8.1.2.2 Acyclic transmission state machine

Details of Acyclic transmission state machine are shown in Table 59.

Table 59 – Acyclic transmission state table

#	Current state	Event/condition => action	Next state
1	IDLE	/ ACTicket == TRUE => SendCounter = MaxSend	SENDER
2	SENDER	/ SendCounter != 0 && length(RemainingData) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => Data = CreateTransient1-PDU(RemainingData); OutPort.req(Transient1-PDU(Data)); SendCounter = SendCounter-1	SENDER
3	SENDER	/ SendCounter != 0 &&	SENDER

#	Current state	Event/condition => action	Next state
		length(RemainingData) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateTransient1-PDU(RemainingData); InPort.req(Transient1-PDU) ; SendCounter = SendCounter-1	
4	SENDER	AC Send.req(Data) / SendCounter != 0 && (OutLoopState == Through || OutLoopState == Loopback) => Create Transient1-PDU(Data); OutPort.req(Transient1-PDU); SendCounter = SendCounter-1	SENDER
5	SENDER	AC Send.req(Data) / SendCounter != 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateTransient1-PDU(Data); InPort.req(Transient1-PDU) ; SendCounter = SendCounter-1	SENDER
6	SENDER	/ SendCounter == 0 => ACTicket = FALSE	IDLE
7	IDLE	ACReceived(Transient1-PDU) => ReassembleData(Transient1-PDU)	IDLE
8	SENDER	ACReceived(Transient1-PDU) => ReassembleData(Transient1-PDU)	SENDER
9	IDLE	ACReceived(Transient2-PDU) => AC Param Send.ind(Data)	IDLE
10	SENDER	ACReceived(Transient2-PDU) => AC Param Send.ind(Data)	SENDER
11	IDLE	/ ReceivedDataAvailable == TRUE => AC Send.ind(Data); ReceivedDataAvailable = FALSE	IDLE
12	SENDER	/ ReceivedDataAvailable == TRUE => AC Send.ind(Data); ReceivedDataAvailable = FALSE	SENDER

8.1.2.3 Functions

Functions enabled in acyclic transmission are shown in Table 60.

Table 60 – Acyclic transmission functions

Name	Description
Length	Request the argument size
CreateTransient1-PDU	Generate Transient1-PDU. If argument data exceeds 1464 octets, Transeint1-PDU is generated using the first 1464 octets. The remaining data are considered RemainingData
ReassembleData	Reassemble the divided data that have been received. dataId is used for data identification. If the size of the received data is equivalent to wholeDataSize, it is considered that reassembly is completed and that ReceivedDataAvailable=TRUE.

8.1.3 Cyclic transmission

8.1.3.1 Primitive definition

The FSPM issues a CT Update.req service to Cyclic transmission. Cyclic transmission issues a CT Update.ind service to the FSPM.

8.1.3.2 Cyclic transmission state machine

Details of Cyclic transmission state machine are shown in Table 61.

NOTE The sending order is not specified for BitCM, WordCM, OutCM1, OutCM2, InCM1, and InCM2.

Table 61 – Cyclic transmission state table

#	Current state	Event/condition => action	Next state

1	IDLE	/ CTicket == TRUE
		=>
		seqno = 0;
		UpdateWaiting = FALSE;
		if length(BitCM) = 0 then BitCMSent = TRUE;
		if length(WordCM) = 0 then WordCMSent = TRUE;
		if length(OutCM1) = 0 then OutCM1Sent = TRUE;
		if length(OutCM2) = 0 then OutCM2Sent = TRUE;
		if length(InCM1) = 0 then InCM1Sent = TRUE;
		if length(InCM2) = 0 then InCM2Sent = TRUE;
			SENDER

2	Any (Any state)	CT Update.req(Data Type, Offset Address, Size, Data) => Update(Data Type, Offset Address, Size, Data)	Any (no change)
3	Any (Any state)	CT Update.req(Data Type, Offset Address, Size, Data) => Update(Data Type, Offset Address, Size, Data)	Any (no change)
4	SENDER	/ UpdateWaiting == TRUE => Update(Data Type, Offset Address, Size, Data); UpdateWaiting = FALSE	SENDER
5	SENDER	/ BitCMSent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER
6	SENDER	/ BitCMSent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER
7	BitCMSENDER	/ length(RemainingBitCM) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER
8	BitCMSENDER	/ length(RemainingBitCM) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER
9	BitCMSENDER	/ length(RemainingBitCM) == 0 => BitCMSent = TRUE	SENDER

#	Current state	Event/condition => action	Next state
10	BitCMSENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	BitCMSENDER
11	SENDER	/ WordCMSent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataW-PDU(seqno, WordCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	WordCMSENDER
12	SENDER	/ WordCMSent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataW-PDU(seqno, WordCM); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	WordCMSENDER
13	WordCMSENDER	/ length(RemainingWordCM) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataW-PDU(seqno, WordCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	WordCMSENDER
14	WordCMSENDER	/ length(RemainingWordCM) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataW-PDU(seqno, WordCM); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	WordCMSENDER
15	WordCMSENDER	/ length(RemainingWordCM) == 0 => WordCMSent = TRUE	SENDER
16	WordCMSENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	WordCMSENDER
17	SENDER	/ OutCM1Sent != TRUE&& Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER
18	SENDER	/ OutCM1Sent != TRUE&& Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER
19	OutCM1SENDER	/ length(RemainingOutCM1) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER
20	OutCM1SENDER	/ length(RemainingOutCM1) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER
21	OutCM1SENDER	/ length(RemainingOutCM1) == 0 =>	SENDER

#	Current state	Event/condition => action	Next state
		OutCM1Sent = TRUE	
22	OutCM1SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	OutCM1SENDER
23	SENDER	/ OutCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataOut2-PDU(seqno, OutCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM2SENDER
24	SENDER	/ OutCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataOut2-PDU(seqno, OutCM2); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM2SENDER
25	OutCM2SENDER	/ length(RemainingOutCM2) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataOut2-PDU(seqno, OutCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM2SENDER
26	OutCM2SENDER	/ length(RemainingOutCM2) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataOut2-PDU(seqno, OutCM2); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM2SENDER
27	OutCM2SENDER	/ length(RemainingOutCM2) == 0 => OutCM2Sent = TRUE	SENDER
28	OutCM2SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	OutCM2SENDER
29	SENDER	/ InCM1Sent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER
30	SENDER	/ InCM1Sent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER
31	InCM1SENDER	/ length(RemainingInCM1) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER
32	InCM1SENDER	/ length(RemainingInCM1) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER
33	InCM1SENDER	/ length(RemainingInCM1) == 0	SENDER

#	Current state	Event/condition => action	Next state
		=> InCM1Sent = TRUE	
34	InCM1SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	InCM1SENDER
35	SENDER	/ InCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataIn2-PDU(seqno, InCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM2SENDER
36	SENDER	/ InCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataIn2-PDU(seqno, InCM2); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM2SENDER
37	InCM2SENDER	/ length(RemainingInCM2) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataIn2-PDU(seqno, InCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM2SENDER
38	InCM2SENDER	/ length(RemainingInCM2) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataIn2-PDU(seqno, InCM2); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM2SENDER
39	InCM2SENDER	/ length(RemainingInCM2) == 0 => InCM2Sent = TRUE	SENDER
40	InCM2SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	InCM2SENDER
41	Any (Any state)	CReceived(CyclicDataB-PDU) => When in the order of the sequential number ReceivedBitCM = RetrieveBitCM(CyclicDataB-PDU); CT Update.ind(BitCM, Offset Address, Size,ReceivedBitCM) When not in the order of the sequential number, discarded	Any (no change)
42	Any (Any state)	CReceived(CyclicDataW-PDU) => When in the order of the sequential number ReceivedWordCM = RetrieveWordCM(CyclicDataW-PDU); CT Update.ind(WordCM, Offset Address, Size, ReceivedWordCM) When not in the order of the sequential number, discarded	Any (no change)
43	Any (Any state)	CReceived(CyclicDataOut1-PDU) => When in the order of the sequential number ReceivedOutCM1 = RetrieveOutCM1(CyclicDataOut1-PDU); CT Update.ind(OutCM1, Offset Address, Size, ReceivedOutCM1) When not in the order of the sequential number, discarded	Any (no change)
44	Any (Any state)	CReceived(CyclicDataOut2-PDU) => When in the order of the sequential number	Any (no change)

#	Current state	Event/condition => action	Next state
		ReceivedOutCM2 = RetrieveOutCM2(CyclicDataOut2-PDU); CT Update.ind(OutCM2, Offset Address, Size, ReceivedOutCM2) When not in the order of the sequential number, discarded	
45	Any (Any state)	CReceived(CyclicDataIn1-PDU) => When in the order of the sequential number ReceivedInCM1 = RetrieveInCM1(CyclicDataIn1-PDU); CT Update.ind(InCM1, Offset Address, Size, ReceivedInCM1) When not in the order of the sequential number, discarded	Any (no change)
46	Any (Any state)	CReceived(CyclicDataIn2-PDU) => When in the order of the sequential number ReceivedInCM2 = RetrieveInCM2(CyclicDataIn2-PDU); CT Update.ind(InCM2, Offset Address, Size, ReceivedInCM2) When not in the order of the sequential number, discarded	Any (no change)
47	SENDER	/ BitCMSent == TRUE && WordCMSent == TRUE && OutCM1Sent == TRUE && OutCM2Sent == TRUE && InCM1Sent == TRUE && InCM2Sent == TRUE => CTicket = FALSE	IDLE

8.1.3.3 Functions

Functions enabled in Cyclic transmission are shown in Table 62.

Table 62 – Cyclic transmission functions

Name	Description
length	Request the size given by argument.
Update	Update the data of BitCM, WordCM, OutCM1, OutCM2, InCM1, and InCM2 which Cyclic Transmission holds to send.
CreateCyclicDataB-PDU	Generate CyclicDataB-PDU. If argument data exceeds 1468 octets, CyclicDataB-PDU is generated using the first 1468 octets. The remaining data are considered RemainingBitCM. When the last PDU is generated, the value of Bit7 of seqNumber is 1.
CreateCyclicDataW-PDU	Generate CyclicDataW-PDU. If argument data exceeds 1468 octets, CyclicDataW-PDU is generated using the first 1468 octets. The remaining data are considered RemainingWordCM. When the last PDU is generated, the value of Bit7 of seqNumber is 1.
CreateCyclicDataOut1-PDU	Generate CyclicDataOut1-PDU. If argument data exceeds 1468 octets, CyclicDataOut1-PDU is generated using the first 1468 octets. The remaining data are considered RemainingOutCM1. When the last PDU is generated, the value of Bit7 of seqNumber is 1.
CreateCyclicDataOut2-PDU	Generate CyclicDataOu2-PDU. If argument data exceeds 1468 octets, CyclicDataOut2-PDU is generated using the first 1468 octets. The remaining data are considered RemainingOutCM2. When the last PDU is generated, the value of Bit7 of seqNumber is 1.
CreateCyclicDataIn1-PDU	Generate CyclicDataIn1-PDU. If argument data exceeds 1468 octets, CyclicDataIn1-PDU is generated using the first 1468 octets. The remaining data are considered RemainingInCM1. When the last PDU is generated, the value of Bit7 of seqNumber is set to 1.
CreateCyclicDataIn2-PDU	Generate CyclicDataIn2-PDU. If argument data exceeds 1468 octets, CyclicDataIn2-PDU is generated using the first 1468 octets. The remaining data are considered RemainingInCM2. When the last PDU is generated, the value of Bit7 of seqNumber is 1.
RetrieveBitCM(PDU)	Retrieve Offset Address, Size, and BitCM data from PDU.

Name	Description
RetrieveWordCM(PDU)	Retrieve Offset Address, Size, and WordCM data from PDU.
RetrieveOutCM1(PDU)	Retrieve Offset Address, Size, and OutCM1 data from PDU.
RetrieveOutCM2(PDU)	Retrieve Offset Address, Size, and OutCM2 data from PDU.
RetrieveInCM1(PDU)	Retrieve Offset Address, Size, and InCM1 data from PDU.
RetrieveInCM2(PDU)	Retrieve Offset Address, Size, and InCM2 data from PDU.

8.1.4 Connection control

8.1.4.1 Connection control state machine

Connection control state machine is described below in Table 63 through Table 76.

Table 63 – Connection control state machine – Initial

#	Current state	Event/condition => action	Next state
1	Initial	InPort.ind(Port_State) / Port_State == LinkUp => Start ConnectTimer InPortState = Checking	Connect
2	Initial	OutPort.ind(Port_State) / Port_State == LinkUp => Start ConnectTimer OutPortState = Checking	Connect
3	Initial	InPort.ind(Port_State) / Port_State == LinkDown =>	Initial
4	Initial	OutPort.ind(Port_State) / Port_State == LinkDown =>	Initial
5	Initial	/ NTNTest == TRUE =>	NTNTestMaster

Table 64 – Connection control state machine – Connect

#	Current state	Event/condition => action	Next state
1	Connect	InPort.ind(Port_State) / Port_State == LinkUp && InPortState == LinkDown && OutPortState == Checking => InPortState = Checking	Connect
2	Connect	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == Checking && OutPortState == LinkDown => OutPortState = Checking	Connect
3	Connect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == NG && OutPortState == Checking => InPortState = LinkDown	Connect
4	Connect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == Checking && OutPortState != OK => InPortState = LinkDown; Stop ConnectTimer.	Initial

#	Current state	Event/condition => action	Next state
5	Connect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == Checking && OutPortState == NG => OutPortState = LinkDown	Connect
6	Connect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == Checking => OutPortState = LinkDown; Stop ConnectTimer	Initial
7	Connect	ConnectTimer times out. / InPortState == Checking && (OutPortState != OK && OutPortState != Checking) => InPort.req(Connect-PDU(PortChoice=In))	Connect
8	Connect	ConnectTimer times out. / (InPortState != OK && InPortState != Checking) && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	Connect
9	Connect	ConnectTimer times out. / InPortState == Checking && OutPortState == Checking => InPort.req(Connect-PDU(PortChoice=In)) OutPort.req(Connect-PDU(PortChoice=Out))	Connect
10	Connect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	Connect
11	Connect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice != Out => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	Connect
12	Connect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Connect
13	Connect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice != In => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	Connect
14	Connect	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && OutPortState == Checking => InPortState = NG	Connect
15	Connect	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == Checking => OutPortState = NG	Connect
16	Connect	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK => InPortState = OK; Start ScanTimer.	Scan
17	Connect	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == Checking && OutPortState != Checking => InPortState = NG;	Initial

#	Current state	Event/condition => action	Next state
		Stop ConnectTimer.	
18	Connect	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK => OutPortState = OK; Start SendScanTimer.	Scan
19	Connect	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState != Checking && OutPortState == Checking => OutPortState = NG; Stop ConnectTimer.	Initial
20	Connect	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState != OK => InPortState = OK; Start ScanTimer.	Scan
21	Connect	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == Checking => OutPortState = OK; Start ScanTimer.	Scan
22	Connect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
23	Connect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 65 – Connection control state machine – Scan

#	Current state	Event/condition => action	Next state
1	Scan	InPort.ind(Port_State) / Port_State == LinkUp && InPortState == LinkDown && OutPortState == OK && StopConnectFlag != ON => Start ConnectTimer; InPortState = Checking	Scan
2	Scan	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState == LinkDown && StopConnectFlag != ON => Start ConnectTimer; OutPortState = Checking	Scan
3	Scan	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	Scan	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	Scan	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK =>	Initial

#	Current state	Event/condition => action	Next state
		OutPortStart = LinkDown	
6	Scan	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	Scan	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	Scan
8	Scan	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	Scan
9	Scan	ScanTimer times out. / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU(scanState = Through))	Scan
10	Scan	ScanTimer times out. / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU(scanState = InLoopback))	Scan
11	Scan	ScanTimer times out. / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU(scanState = OutLoopback))	Scan
12	Scan	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	Scan
13	Scan	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	Scan
14	Scan	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Scan
15	Scan	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	Scan
16	Scan	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
17	Scan	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	Scan
18	Scan	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK &&	Scan

#	Current state	Event/condition => action	Next state
		InPortState == OK && OutPortState == Checking => OutPortState = OK	
19	Scan	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	Scan
20	Scan	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState != OK && Latest(Scan-PDU) == TRUE => InPort.req(Scan-PDU)	Scan
21	Scan	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => OutPort.req(Scan-PDU)	Scan
22	Scan	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	Scan
23	Scan	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState != OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => OutPort.req(Scan-PDU)	Scan
24	Scan	InPort.ind(source_address, Scan-PDU(scanState)) / source_address == my address && InPortState == OK => Start DetectScanTimer.	ScanWait
25	Scan	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
26	Scan	OutPort.ind(source_address, Scan-PDU(scanState)) / source_address == my address && InPortState != OK && OutPortState == OK => Start DetectScanTimer.	ScanWait
27	Scan	OutPort.ind(source_address, Scan-PDU) / source_address == my address && InPortState == OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => InPort.req(Scan-PDU)	Scan
28	Scan	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
29	Scan	InPort.ind(PDU) / PDU != Connect-PDU && PDU != ConnectAck-PDU && PDU != Scan-PDU =>	Scan
30	Scan	OutPort.ind(PDU) / PDU != Connect-PDU && PDU != ConnectAck-PDU && PDU != Scan-PDU =>	Scan
31	Scan	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest =>	NTNTestSlave

#	Current state	Event/condition => action	Next state
		OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	
32	Scan	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 66 – Connection control state machine – ScanWait

#	Current state	Event/condition => action	Next state
1	ScanWait	InPort.ind(Port_State) / Port_State == LinkUp && InPortState == LinkDown && OutPortState == OK && StopConnectFlag != ON => Start ConnectTimer; InPortState = Checking	ScanWait
2	ScanWait	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState == LinkDown && StopConnectFlag != ON => Start ConnectTimer; OutPortState = Checking	ScanWait
3	ScanWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	ScanWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	ScanWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	ScanWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState !=LinkDown => OutPortState = LinkDown	Scan
7	ScanWait	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	ScanWait
8	ScanWait	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	ScanWait
9	ScanWait	DetectScanTimer times out. / InPortState == OK || OutPortState == OK => Start CollectTimer.	Collect
10	ScanWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	ScanWait

#	Current state	Event/condition => action	Next state
11	ScanWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	ScanWait
12	ScanWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	ScanWait
13	ScanWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	ScanWait
14	ScanWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
15	ScanWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	ScanWait
16	ScanWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
17	ScanWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	ScanWait
18	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address == my address && (InPortState == OK || InPortState == Checking) => Restart DetectScanTimer.	ScanWait
19	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => Restart DetectScanTimer; OutPort.req(Scan-PDU)	ScanWait
20	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState != OK && Latest(Scan-PDU) == TRUE => Restart DetectScanTimer; InPort.req(Scan-PDU)	ScanWait
21	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && Latest(Scan-PDU) != TRUE => Restart DetectScanTimer;	ScanWait
22	ScanWait	OutPort.ind(source_address, Scan-PDU)	ScanWait

#	Current state	Event/condition => action	Next state
		/ source_address == my address && (InPortState == OK || OutPortState == OK) => Restart DetectScanTimer.	
23	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState != OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => Restart DetectScanTimer; OutPort.req(Scan-PDU)	ScanWait
24	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState != OK && OutPortState == OK && Latest(Scan-PDU) != TRUE => Restart DetectScanTimer.	ScanWait
25	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address == my address && InPortState == OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => InPort.req(Scan-PDU)	ScanWait
26	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	ScanWait
27	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
28	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
29	ScanWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
30	ScanWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 67 – Connection control state machine – Collect

#	Current state	Event/condition => action	Next state
1	Collect	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	Collect
2	Collect	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	Collect

#	Current state	Event/condition => action	Next state
3	Collect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	Collect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	Collect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	Collect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	Collect	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	Collect
8	Collect	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	Collect
9	Collect	CollectTimer times out. / InPortState == OK && OutPortState == OK => OutPort.req(Collect-PDU(InLoopState = Through, OutLoopState = Through))	Collect
10	Collect	CollectTimer times out. / InPortState == OK && OutPortState != OK => InPort.req(Collect-PDU(InLoopState = Loopback, OutLoopState =OutPortState))	Collect
11	Collect	CollectTimer times out. / InPortState != OK && OutPortState == OK => OutPort.req(Collect-PDU(InLoopState = InPortState, OutLoopState = Loopback))	Collect
12	Collect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	Collect
13	Collect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	Collect
14	Collect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Collect
15	Collect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) &&	Collect

#	Current state	Event/condition => action	Next state
		OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	
16	Collect	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
17	Collect	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	Collect
18	Collect	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
19	Collect	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	Collect
20	Collect	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
21	Collect	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	Collect	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
23	Collect	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
24	Collect	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
25	Collect	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	Collect
26	Collect	InPort.ind(source_address, Collect-PDU) / source_address != my address && InPortState == OK && OutPortState == OK => CPDReceived(Collect-PDU); OutPort.req(Collect-PDU)	Collect
27	Collect	InPort.ind(source_address, Collect-PDU) / source_address != my address && InPortState == OK && OutPortState != OK => CPDReceived(Collect-PDU); InPort.req(Collect-PDU)	Collect
28	Collect	OutPort.ind(source_address, Collect-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Collect-PDU)	Collect

#	Current state	Event/condition => action	Next state
29	Collect	OutPort.ind(source_address, Collect-PDU) / source_address != my address && InPortState != OK && OutPortState == OK => CPDReceived(Collect-PDU); OutPort.req(Collect-PDU)	Collect
30	Collect	InPort.ind(source_address, Collect-PDU) / source_address == my address && InPortState == OK => Start DetectCollectTimer.	CollectWait
31	Collect	OutPort.ind(source_address, Collect-PDU) / source_address == my address && InPortState != OK && OutPortState == OK => Start DetectCollectTimer.	CollectWait
32	Collect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
33	Collect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 68 – Connection control state machine – CollectWait

#	Current state	Event/condition => action	Next state
1	CollectWait	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	CollectWait
2	CollectWait	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	CollectWait
3	CollectWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	CollectWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	CollectWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	CollectWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	CollectWait	ConnectTimer times out. / InPortState == Checking && OutPortState == OK	CollectWait

#	Current state	Event/condition => action	Next state
		=> InPort.req(Connect-PDU(PortChoice=In));	
8	CollectWait	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	CollectWait
9	CollectWait	DetectCollectTimer times out. / InPortState == OK || OutPortState == OK => CollectEnd == TRUE; Start SelectTimer.	Select
10	CollectWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	CollectWait
11	CollectWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	CollectWait
12	CollectWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	CollectWait
13	CollectWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	CollectWait
14	CollectWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
15	CollectWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	CollectWait
16	CollectWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
17	CollectWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	CollectWait
18	CollectWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
19	CollectWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan

#	Current state	Event/condition => action	Next state
20	CollectWait	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
21	CollectWait	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	CollectWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
23	CollectWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	CollectWait
24	CollectWait	InPort.ind(source_address, Collect-PDU) / source_address == my address && InPortState == OK => Restart DetectCollectTimer.	CollectWait
25	CollectWait	OutPort.ind(source_address, Collect-PDU) / source_address == my address && InPortState != OK && OutPortState == OK => Restart DetectCollectTimer.	CollectWait
26	CollectWait	InPort.ind(source_address, Collect-PDU) / source_address != my address && InPortState == OK && OutPortState == OK => Restart DetectCollectTimer; CPDReceived(Collect-PDU); OutPort.req(Collect-PDU)	CollectWait
27	CollectWait	InPort.ind(source_address, Collect-PDU) / source_address != my address && InPortState == OK && OutPortState != OK => Restart DetectCollectTimer; CPDReceived(Collect-PDU); InPort.req(Collect-PDU)	CollectWait
28	CollectWait	OutPort.ind(source_address, Collect-PDU) / source_address != my address && InPortState != OK && OutPortState == OK => Restart DetectCollectTimer. CPDReceived(Collect-PDU); OutPort.req(Collect-PDU)	CollectWait
29	CollectWait	OutPort.ind(Collect-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Collect-PDU)	CollectWait
30	CollectWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
31	CollectWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 69 – Connection control state machine – Select

#	Current state	Event/condition => action	Next state
1	Select	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	Select
2	Select	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	Select
3	Select	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	Select	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	Select	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	Select	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	Select	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	Select
8	Select	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	Select
9	Select	SelectTimer times out. / InPortState == OK && OutPortState == OK => OutPort.req(Select-PDU)	Select
10	Select	SelectTimer times out. / InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	Select
11	Select	SelectTimer times out. / InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	Select
12	Select	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	Select
13	Select	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown =>	Select

#	Current state	Event/condition => action	Next state
		InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	
14	Select	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Select
15	Select	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	Select
16	Select	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
17	Select	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	Select
18	Select	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
19	Select	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	Select
20	Select	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
21	Select	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	Select	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
23	Select	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
24	Select	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
25	Select	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	Select
26	Select	InPort.ind(source_address, Select-PDU) / source_address == my address && InPortState == OK =>	TokenStartWait

#	Current state	Event/condition => action	Next state
		Start LaunchTimer.	
27	Select	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	LaunchWait
28	Select	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
29	Select	InPort.ind(source_address, Select-PDU) / source_address > my address =>	Select
30	Select	OutPort.ind(source_address, Select-PDU) / source_address == my address && InPortState != OK && OutPortState == OK => Start LaunchTimer.	TokenStartWait
31	Select	OutPort.ind(source_address, Select-PDU) / source_address < my address && InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
32	Select	OutPort.ind(source_address, Select-PDU) / source_address > my address && InPortState != OK && OutPortState == OK =>	Select
33	Select	OutPort.ind(Select-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Select-PDU)	Select
34	Select	InPort.ind(PDU) / (PDU == Token-PDU || PDU == MyStatus-PDU || PDU == Transient1-PDU || PDU == Dummy-PDU || PDU == Transient2-PDU || PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState == OK =>	Select
35	Select	OutPort.ind(PDU) / (PDU == Token-PDU || PDU == MyStatus-PDU || PDU == Transient1-PDU || PDU == Dummy-PDU || PDU == Transient2-PDU || PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && OutPortState == OK =>	Select
36	Select	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
37	Select	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest =>	NTNTestSlave

#	Current state	Event/condition => action	Next state
		InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	

Table 70 – Connection control state machine – TokenStartWait

#	Current state	Event/condition => action	Next state
1	TokenStartWait	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	TokenStartWait
2	TokenStartWait	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	TokenStartWait
3	TokenStartWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	TokenStartWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	TokenStartWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	TokenStartWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	TokenStartWait	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	TokenStartWait
8	TokenStartWait	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	TokenStartWait
9	TokenStartWait	LaunchTimer times out. / InPortState == OK && OutPortState == OK => OutPort.req(Select-PDU)	TokenStartWait
10	TokenStartWait	LaunchTimer times out. / InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	TokenStartWait
11	TokenStartWait	LaunchTimer times out. / InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	TokenStartWait
12	TokenStartWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown =>	TokenStartWait

#	Current state	Event/condition => action	Next state
		InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	
13	TokenStartWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	TokenStartWait
14	TokenStartWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	TokenStartWait
15	TokenStartWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	TokenStartWait
16	TokenStartWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
17	TokenStartWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	TokenStartWait
18	TokenStartWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
19	TokenStartWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	TokenStartWait
20	TokenStartWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
21	TokenStartWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	TokenStartWait	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
23	TokenStartWait	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
24	TokenStartWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
25	TokenStartWait	OutPort.ind(Scan-PDU)	TokenStartWait

#	Current state	Event/condition => action	Next state
		/ InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	
26	TokenStartWait	InPort.ind(Launch-PDU) / InPortStatus == OK && OutPortState != OK => CTicket = TRUE	TokenReleaseWait
27	TokenStartWait	InPort.ind(Launch-PDU) / InPortStatus == OK && OutPortState == OK => CTicket = TRUE	TokenReleaseWait
28	TokenStartWait	OutPort.ind(Launch-PDU) / InPortState != OK && OutPortState == OK => CTicket = TRUE	TokenReleaseWait
29	TokenStartWait	OutPort.ind(Launch-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Launch-PDU)	TokenStartWait
30	TokenStartWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
31	TokenStartWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 71 – Connection control state machine – LaunchWait

#	Current state	Event/condition => action	Next state
1	LaunchWait	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	LaunchWait
2	LaunchWait	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	LaunchWait
3	LaunchWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	LaunchWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	LaunchWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	LaunchWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown	Scan

#	Current state	Event/condition => action	Next state
		=> OutPortState = LinkDown	
7	LaunchWait	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	LaunchWait
8	LaunchWait	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	LaunchWait
9	LaunchWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	LaunchWait
10	LaunchWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	LaunchWait
11	LaunchWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	LaunchWait
12	LaunchWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	LaunchWait
13	LaunchWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
14	LaunchWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	LaunchWait
15	LaunchWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
16	LaunchWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	LaunchWait
17	LaunchWait	OutPort.ind(Select-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Select-PDU)	LaunchWait
18	LaunchWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan

#	Current state	Event/condition => action	Next state
19	LaunchWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
20	LaunchWait	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
21	LaunchWait	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	LaunchWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
23	LaunchWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	LaunchWait
24	LaunchWait	InPort.ind(source_address, Select-PDU) / source_address == my address && =>	LaunchWait
25	LaunchWait	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	LaunchWait
26	LaunchWait	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
27	LaunchWait	InPort.ind(source_address, Select-PDU) / source_address > my address =>	LaunchWait
28	LaunchWait	OutPort.ind(source_address, Select-PDU) / source_address == my address && InPortState != OK && OutPortState == OK =>	LaunchWait
29	LaunchWait	OutPort.ind(source_address, Select-PDU) / source_address < my address && InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
30	LaunchWait	OutPort.ind(source_address, Select-PDU) / source_address > my address && InPortState != OK && OutPortState == OK =>	LaunchWait
31	LaunchWait	InPort.ind(Launch-PDU) / InPortStatus == OK && OutPortStatus != OK => InPort.req(Launch-PDU)	TokenWait
32	LaunchWait	InPort.ind(Launch-PDU) / InPortStatus == OK && OutPortStatus == OK => OutPort.req(Launch-PDU)	TokenWait
33	LaunchWait	OutPort.ind(Launch-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Launch-PDU)	TokenWait
34	LaunchWait	OutPort.ind(Launch-PDU)	LaunchWait

#	Current state	Event/condition => action	Next state
		/ InPortState == OK && OutPortState == OK => InPort.req(Launch-PDU)	
35	LaunchWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
36	LaunchWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 72 – Connection control state machine – TokenReleaseWait

#	Current state	Event/condition => action	Next state
1	TokenReleaseWait	CTicket == FALSE => ACTicket = TRUE	TokenReleaseWait
2	TokenReleaseWait	ACTicket == FALSE / InPortState == OK && OutPortState != OK => InPort.ind(Token-PDU)	TokenReleased
3	TokenReleaseWait	ACTicket == FALSE / OutPortState == OK => OutPort.ind(Token-PDU) Start NetworkWatchTimer.	TokenReleased
4	TokenReleaseWait	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	TokenReleaseWait
5	TokenReleaseWait	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	TokenReleaseWait
6	TokenReleaseWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
7	TokenReleaseWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
8	TokenReleaseWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
9	TokenReleaseWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
10	TokenReleaseWait	ConnectTimer times out. / InPortState == Checking && OutPortState == OK	TokenReleaseWait

#	Current state	Event/condition => action	Next state
		=> InPort.req(Connect-PDU(PortChoice=In));	
11	TokenReleaseWait	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	TokenReleaseWait
12	TokenReleaseWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	TokenReleaseWait
13	TokenReleaseWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	TokenReleaseWait
14	TokenReleaseWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	TokenReleaseWait
15	TokenReleaseWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	TokenReleaseWait
16	TokenReleaseWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
17	TokenReleaseWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	TokenReleaseWait
18	TokenReleaseWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
19	TokenReleaseWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	TokenReleaseWait
20	TokenReleaseWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
21	TokenReleaseWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	TokenReleaseWait	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan

#	Current state	Event/condition => action	Next state
23	TokenReleaseWait	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
24	TokenReleaseWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
25	TokenReleaseWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	TokenReleaseWait
26	TokenReleaseWait	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	LaunchWait
27	TokenReleaseWait	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
28	TokenReleaseWait	InPort.ind(source_address, Select-PDU) / source_address > my address =>	Select
29	TokenReleaseWait	OutPort.ind(source_address, Select-PDU) / source_address < my address && InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
30	TokenReleaseWait	OutPort.ind(source_address, Select-PDU) / source_address > my address && InPortState != OK && OutPortState == OK =>	Select
31	TokenReleaseWait	OutPort.ind(Select-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Select-PDU)	TokenReleaseWait
32	TokenReleaseWait	InPort.ind(Launch-PDU) =>	TokenReleaseWait
33	TokenReleaseWait	OutPort.ind(Launch-PDU) =>	TokenReleaseWait
34	TokenReleaseWait	InPort.ind(PDU) / (PDU == MyStatus-PDU || PDU == Transient1-PDU || PDU == Dummy-PDU || PDU == Transient2-PDU || PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState == OK =>	TokenReleaseWait
35	TokenReleaseWait	OutPort.ind(PDU) / (PDU == MyStatus-PDU || PDU == Transient1-PDU || PDU == Dummy-PDU || PDU == Transient2-PDU || PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) &&	TokenReleaseWait

#	Current state	Event/condition => action	Next state
		OutPortState == OK =>	
36	TokenReleaseWait	Control Cyclic.req(Control_Type) / Control_Type == Restart => CyclicControl = Running; Control Cyclic.cnf(State = Running)	TokenReleaseWait
37	TokenReleaseWait	Control Cyclic.req(Control_Type) / Control_Type == Stop => CyclicControl = Stop; Control Cyclic.cnf(State = Stop)	TokenReleaseWait
38	TokenReleaseWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
39	TokenReleaseWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 73 – Connection control state machine – TokenReleased

#	Current state	Event/condition => action	Next state
1	TokenReleased	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	TokenReleased
2	TokenReleased	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	TokenReleased
3	TokenReleased	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	TokenReleased	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	TokenReleased	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	TokenReleased	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	TokenReleased	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	TokenReleased
8	TokenReleased	ConnectTimer times out.	TokenReleased

#	Current state	Event/condition => action	Next state
		/ InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	
9	TokenReleased	NetworkWatchTime times out.=>	Select
10	TokenReleased	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	TokenReleased
11	TokenReleased	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	TokenReleased
12	TokenReleased	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	TokenReleased
13	TokenReleased	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	TokenReleased
14	TokenReleased	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
15	TokenReleased	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	TokenReleased
16	TokenReleased	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
17	TokenReleased	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	TokenReleased
18	TokenReleased	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
19	TokenReleased	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
20	TokenReleased	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
21	TokenReleased	OutPort.ind(Scan-PDU)	Scan

#	Current state	Event/condition => action	Next state
		/ InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	
22	TokenReleased	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
23	TokenReleased	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	TokenReleased
24	TokenReleased	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	LaunchWait
25	TokenReleased	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
26	TokenReleased	InPort.ind(source_address, Select-PDU) / source_address > my address =>	Select
27	TokenReleased	OutPort.ind(source_address, Select-PDU) / source_address < my address && InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
28	TokenReleased	OutPort.ind(source_address, Select-PDU) / source_address > my address && InPortState != OK && OutPortState == OK =>	Select
29	TokenReleased	OutPort.ind(Select-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Select-PDU)	TokenReleased
30	TokenReleased	InPort.ind(Launch-PDU) =>	TokenReleased
31	TokenReleased	OutPort.ind(Launch-PDU) =>	TokenReleased
32	TokenReleased	InPort.ind(Token-PDU) / InPortState == OK =>	TokenReleaseWait
33	TokenReleased	OutPort.ind(Token-PDU) / InPortState != OK && OutPortState == OK =>	TokenReleaseWait
34	TokenReleased	OutPort.ind(Token-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Token-PDU)	TokenReleased
41	TokenReleased	InPort.ind(source_address, PDU) / source_address == my address && (PDU == MyStatus-PDU || PDU == Transient1-PDU || PDU == Dummy-PDU || PDU == Transient2-PDU || PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState == OK	TokenReleased

#	Current state	Event/condition => action	Next state
		=> Restart NetworkWatchTimer.	
42	TokenReleased	InPort.ind(source_address, PDU) / source_address != my address && (PDU == MyStatus-PDU || PDU == Dummy-PDU) && InPortState == OK && OutPortState != OK => InPort.req(PDU): Restart NetworkWatchTimer.	TokenWait
43	TokenReleased	InPort.ind(source_address, PDU) / source_address != my address && (PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState == OK && OutPortState != OK => ACReceived(PDU) InPort.req(PDU): Restart NetworkWatchTimer.	TokenWait
44	TokenReleased	InPort.ind(source_address, PDU) / source_address != my address && (PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState == OK && OutPortState != OK => CReceived(PDU) InPort.req(PDU): Restart NetworkWatchTimer.	TokenWait
45	TokenReleased	InPort.ind(source_address, PDU) / source_address != my address && (PDU == MyStatus-PDU || PDU == Dummy-PDU) && InPortState == OK && OutPortState == OK => OutPort.req(PDU): Restart NetworkWatchTimer.	TokenWait
46	TokenReleased	InPort.ind(source_address, PDU) / source_address != my address && (PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState == OK && OutPortState == OK => ACReceived(PDU) OutPort.req(PDU): Restart NetworkWatchTimer.	TokenWait
47	TokenReleased	InPort.ind(source_address, PDU) / source_address != my address && (PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState == OK && OutPortState == OK => CReceived(PDU) OutPort.req(PDU): Restart NetworkWatchTimer.	TokenWait
48	TokenReleased	OutPort.ind(source_address, PDU) / source_address == my address && (PDU == MyStatus-PDU || PDU == Transient1-PDU || PDU == Dummy-PDU || PDU == Transient2-PDU || PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU ||	TokenReleased

#	Current state	Event/condition => action	Next state
		PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && OutPortState == OK => Restart NetworkWatchTimer.	
49	TokenReleased	OutPort.ind(source_address, PDU) / source_address != my address && (PDU == MyStatus-PDU || PDU == Dummy-PDU) && InPortState != OK && OutPortState == OK => OutPort.req(PDU): Restart NetworkWatchTimer.	TokenWait
50	TokenReleased	OutPort.ind(source_address, PDU) / source_address != my address && (PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState != OK && OutPortState == OK => ACReceived(PDU) OutPort.req(PDU): Restart NetworkWatchTimer.	TokenWait
51	TokenReleased	OutPort.ind(source_address, PDU) / source_address != my address && (PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState != OK && OutPortState == OK => CReceived(PDU) OutPort.req(PDU): Restart NetworkWatchTimer.	TokenWait
52	TokenReleased	OutPort.ind(source_address, PDU) / source_address != my address && (PDU == MyStatus-PDU || PDU == Dummy-PDU && InPortState == OK && OutPortState == OK => InPort.req(PDU)	TokenReleased
53	TokenReleased	OutPort.ind(source_address, PDU) / source_address != my address && (PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState == OK && OutPortState == OK => ACReceived(PDU) InPort.req(PDU)	TokenReleased
54	TokenReleased	OutPort.ind(source_address, PDU) / source_address != my address && (PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState == OK && OutPortState == OK => CReceived(PDU) InPort.req(PDU)	TokenReleased
55	TokenReleased	Control Cyclic.req(Control_Type) / Control_Type == Restart => CyclicControl = Running; Control Cyclic.cnf(State = Running)	TokenReleased
56	TokenReleased	Control Cyclic.req(Control_Type) / Control_Type == Stop => CyclicControl = Stop;	TokenReleased

#	Current state	Event/condition => action	Next state
		Control Cyclic.cnf(State = Stop)	
57	TokenReleased	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
58	TokenReleased	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 74 – Connection control state machine – TokenWait

#	Current state	Event/condition => action	Next state
1	TokenWait	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	TokenWait
2	TokenWait	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	TokenWait
3	TokenWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	TokenWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	TokenWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	TokenWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	TokenWait	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	TokenWait
8	TokenWait	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	TokenWait
9	TokenWait	NetworkWatchTime times out. =>	Select
10	TokenWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK));	TokenWait

#	Current state	Event/condition => action	Next state
		InPortState = OK	
11	TokenWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	TokenWait
12	TokenWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	TokenWait
13	TokenWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	TokenWait
14	TokenWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
15	TokenWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	TokenWait
16	TokenWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
17	TokenWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	TokenWait
18	TokenWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
19	TokenWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
20	TokenWait	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
21	TokenWait	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	TokenWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
23	TokenWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK	TokenWait

#	Current state	Event/condition => action	Next state
		=> InPort.req(Scan-PDU)	
24	TokenWait	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	LaunchWait
25	TokenWait	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
26	TokenWait	InPort.ind(source_address, Select-PDU) / source_address > my address && InPortState == OK =>	Select
27	TokenWait	OutPort.ind(source_address, Select-PDU) / source_address < my address && InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
28	TokenWait	OutPort.ind(source_address, Select-PDU) / source_address > my address && InPortState != OK && OutPortState == OK =>	Select
29	TokenWait	OutPort.ind(source_address, Select-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Select-PDU)	TokenWait
30	TokenWait	InPort.ind(Launch-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Launch -PDU)	TokenWait
31	TokenWait	InPort.ind(Launch-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Launch -PDU)	TokenWait
32	TokenWait	OutPort.ind(Launch-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Launch -PDU)	TokenWait
33	TokenWait	OutPort.ind(Launch-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Launch -PDU)	TokenWait
34	TokenWait	InPort.ind(Token-PDU) / InPortState == OK =>	TokenRelease Wait
35	TokenWait	OutPort.ind(Token-PDU) / InPortState != OK && OutPortState == OK =>	TokenRelease Wait
36	TokenWait	OutPort.ind(Token-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Token-PDU)	TokenWait
40	TokenWait	InPort.ind(source_address, PDU) / (PDU == MyStatus-PDU || PDU == Dummy-PDU) && InPortState == OK && OutPortState != OK => InPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
41	TokenWait	InPort.ind(source_address, PDU) / (PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState == OK && OutPortState != OK	TokenWait

#	Current state	Event/condition => action	Next state
		=> ACReceived(PDU); InPort.req(PDU); Restart NetworkWatchTimer.	
42	TokenWait	InPort.ind(source_address, PDU) / (PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState == OK && OutPortState != OK => CReceived(PDU); InPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
43	TokenWait	InPort.ind(source_address, PDU) / (PDU == MyStatus-PDU || PDU == Dummy-PDU) && InPortState == OK && OutPortState == OK => OutPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
44	TokenWait	InPort.ind(source_address, PDU) / (PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState == OK && OutPortState == OK => ACReceived(PDU); OutPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
45	TokenWait	InPort.ind(source_address, PDU) / (PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState == OK && OutPortState == OK => CReceived(PDU); OutPort.req(PDU); Restart NetworkWatchTimer	TokenWait
46	TokenWait	OutPort.ind(source_address, PDU) / (PDU == MyStatus-PDU || PDU == Dummy-PDU) && InPortState != OK&& OutPortState == OK => OutPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
47	TokenWait	OutPort.ind(source_address, PDU) / (PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState != OK&& OutPortState == OK => ACReceived(PDU); OutPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
48	TokenWait	OutPort.ind(source_address, PDU) / (PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState != OK&& OutPortState == OK => CReceived(PDU); OutPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
49	TokenWait	OutPort.ind(source_address, PDU) / (PDU == MyStatus-PDU ||	TokenWait

#	Current state	Event/condition => action	Next state
		PDU == Dummy-PDU) && InPortState == OK && OutPortState == OK => InPort.req(PDU)	
50	TokenWait	OutPort.ind(source_address, PDU) / PDU ==(PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState == OK && OutPortState == OK => ACReceived(PDU); InPort.req(PDU)	TokenWait
51	TokenWait	OutPort.ind(source_address, PDU) / (PDU == CyclicDataW-PDU || PDU == CyclicDataB-PDU || PDU == CyclicDataOut1-PDU || PDU == CyclicDataOut2-PDU || PDU == CyclicDataIn1-PDU || PDU == CyclicDataIn2-PDU) && InPortState == OK && OutPortState == OK => CReceived(PDU); InPort.req(PDU)	TokenWait
52	TokenWait	Control Cyclic.req(Control_Type) / Control_Type == Restart => CyclicControl = Running; Control Cyclic.cnf(State = Running)	TokenWait
53	TokenWait	Control Cyclic.req(Control_Type) / Control_Type == Stop => CyclicControl = Stop; Control Cyclic.cnf(State = Stop)	TokenWait
54	TokenWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
55	TokenWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 75 – Connection control state machine – NTNTestMaster

#	Current state	Event/condition => action	Next state
1	NTNTestMaster	OutPort.ind(Port_State) / Port_State == LinkUp => OutPort.req(Connect-PDU(PortChoice=NTNTest)); NTNTestTimer is started.	NTNTestMaster
2	NTNTestMaster	OutPort.ind(Port_State) / Port_State == LinkDown =>	NTNTestMaster
3	NTNTestMaster	OutPort.ind(ConnectAck-PDU(PortChoice)) / PortCheckResut != NTNTestNG && PortCheckResut != NTNTestOK => NTN Test Result = NTN Test NG	NTNTestMaster
4	NTNTestMaster	OutPort.ind(ConnectAck-PDU(PortChoice)) / PortCheckResut == NTNTestNG => NTN Test Result = NTN Test NG	NTNTestMaster
5	NTNTestMaster	OutPort.ind(ConnectAck-PDU(PortChoice))	NTNTestMaster

#	Current state	Event/condition => action	Next state
		/ (PortCheckResut == NTNTestNG || PortCheckResut == NTNTestOK) && Tries != 0 => OutPort.req(NTNTest-PDU); Tries = Tries – 1; NTNTestTimer is restarted.	
6	NTNTestMaster	OutPort.ind(ConnectAck-PDU(PortChoice)) / (PortCheckResut == NTNTestNG || PortCheckResut == NTNTestOK) && Tries == 0 => NTN Test Result = NTN Test OK	NTNTestMaster
7	NTNTestMaster	NTNTestTimer times out. => NTN Test Result = NTN Test NG	NTNTestMaster
8	NTNTestMaster	OutPort.ind(Connect-PDU) =>	NTNTestMaster

Table 76 – Connection control state machine – NTNTestSlave

#	Current state	Event/condition => action	Next state
1	NTNTestSlave	OutPort.ind(Connect-PDU) => OutPortReq(Connect-PDU)	NTNTestSlave
2	NTNTestSlave	OutPort.ind(Port_State) / Port_State == LinkDown =>	Initial

8.1.4.2 Functions

Functions used in connection control are shown in Table 77.

Table 77 – Function list of connection control

Name	Description
Latest(Scan-PDU)	Decides whether the data of Scan-PDU is the latest. If yes, return TRUE, if not, return FALSE.
CPDReceived(Collect-PDU)	Return Collect-PDU to Common Parameter Dist state machine.
CReceived(PDU)	Return PDU to Cyclic Transmission state machine.
ACReceived(PDU)	Return PDU to Acyclic Transmission state machine.

8.1.5 Common parameter dist

8.1.5.1 Common parameter dist state machine

Details of the common parameter dist state machine are shown in Table 78.

Table 78 – Common parameter dist state table

#	Current state	Event/condition => action	Next state
1	SetUp	CPD Set.req(NodeID, Param) => CPUpdeate(NodeId, Param); CPID = CreateCPID(NodeId); CPIDUpdate(CPID); CPD Set.ind	SetUp

#	Current state	Event/condition => action	Next state
2	SetUp	CPDReceived(Collect-PDU(NodeId, CPID)) => CPIDUpdate(NodeId, CPID); CollectEnd == FALSE	Initial
3	Initial	CPDReceived(Collect-PDU(NodeId, CPID)) => CPIDUpdate(NodeId, CPID); CollectEnd == FALSE	Initial
4	Initial	CollectEnd == TRUE => MasterNode = CPDMaster(); ReceivingNodes = DestNodeNum()	CPNodeTypeSelect
5	Initial	CPD Set.req(NodeID, Param) => CPUpdeate(NodeId, Param); CPID = CreateCPID(NodeId); CPIDUpdate(CPID); CPD Set.ind	Initial
6	CPNodeTypeSelect	/ MasterNode == Master && ReceivingNodes > 0 => CreateDistResultList()	MasterInit
7	CPNodeTypeSelect	/ MasterNode == Master && ReceivingNodes == 0	MasterEnd
8	CPNodeTypeSelect	/ MasterNode == Slave && HasValidCP() == TRUE	SlaveEnd
9	CPNodeTypeSelect	/ MasterNode == Slave && HasValidCP() != TRUE	SlaveInit
10	MasterInit	CollectEnd == TRUE => MasterNode = CPDMaster(); ReceivingNodes = DestNodeNum()	CPNodeTypeSelect
11	MasterInit	TokenPassingStart==TRUE => CPW.req	CPWSend
12	MasterInit	CPD Set.req(NodeID, Param) => CPUpdeate(NodeId, Param); CPID = CreateCPID(NodeId); CPIDUpdate(CPID); CPD Set.ind	Initial
13	CPWSend	=> CPWC.req, TPCWTimer is started.	CPWCRReceiveWaiting
14	CPWCRReceiveWaiting	CollectEnd == TRUE => MasterNode = CPDMaster(); ReceivingNodes = DestNodeNum()	CPNodeTypeSelect
15	CPWCRReceiveWaiting	TPCWTimer times out. => ReceivingNodes = DestNodeNum(); ResultCheckedNodes = DistResultListNum();	TPCWExpired
16	CPWCRReceiveWaiting	CPWCR.ind(source_node, CheckResult) / CheckResult == CPNotReceived => AddDestNode(NodeId)	CPWCRReceiveWaiting
17	CPWCRReceiveWaiting	CPWCR.ind(source_node, CheckResult) / (CheckResult == OK || CheckResult == NG) && RemoveDistResult(source_node) && DistResultListNum() == 0 => TPCWTimer stops	MasterEnd
18	CPWCRReceiveWaiting	CPD Set.req(NodeID, Param) => CPUpdeate(NodeId, Param); CPID = CreateCPID(NodeId); CPIDUpdate(CPID); CPD Set.ind	Initial

#	Current state	Event/condition => action	Next state
19	TPCWExpired	ReceivingNodes > 0 => TPCW stops, CPW.req	CPWSend
20	TPCWExpired	ReceivingNodes == 0 && ResultCheckedNodes > 0 => TPCW stops, CPWC.req	CPWCRReceiveWaiting
21	TPCWExpired	ReceivingNodes == 0 && ResultCheckedNodes == 0 => TPCW stops.	MasterEnd
22	MasterEnd	CollectEnd == TRUE => MasterNode = CPDMaster(); ReceivingNodes = DestNodeNum()	CPNodeTypeSelect
23	MasterEnd	CPD Set.req(NodeID, Param) => CPUpdeate(NodeId, Param); CPID = CreateCPID(NodeId); CPIDUpdate(CPID); CPD Set.ind	Initial
24	SlaveInit	CollectEnd == TRUE => MasterNode = CPDMaster(); ReceivingNodes = DestNodeNum()	CPNodeTypeSelect
25	SlaveInit	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) == 0 => CPIDUpdate(CPID)	SlaveEnd
26	SlaveInit	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) != 0 => CPIDUpdate(CPID); CPCheck(CP); CPCheckResult = CPChecking	ReceivedCPChecking
27	SlaveInit	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) == 0 => CPWCR.req(CheckResult = CPNotReceived)	SlaveInit
28	SlaveInit	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) != 0 => CPWCR.req(CheckResult = CPCheckResult)	SlaveEnd
29	SlaveInit	CPD Set.req(NodeID, Param) => CPUpdeate(NodeId, Param); CPID = CreateCPID(NodeId); CPIDUpdate(CPID); CPD Set.ind	Initial
30	ReceivedCPChecking	CollectEnd == TRUE => MasterNode = CPDMaster(); ReceivingNodes = DestNodeNum(); CollectEndReceived = TRUE	ReceivedCPChecking
31	ReceivedCPChecking	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) == 0 => CPIDUpdate(CPID)	SlaveEnd
32	ReceivedCPChecking	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) != 0 => CPIDUpdate(CPID);	ReceivedCPChecking

#	Current state	Event/condition => action	Next state
		CPChanged = TRUE	
33	ReceivedCPChecking	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) == 0 => CPWCR.req(CheckResult = CPNotReceived); CPChanged = TRUE	ReceivedCPChecking
34	ReceivedCPChecking	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) != 0 => CPWCR.req(CheckResult = CPChecking)	ReceivedCPChecking
35	ReceivedCPChecking	CPCheckFinished / CollectEndReceived == TRUE => CollectEndReceived = FALSE	CPNodeTypeSelect
36	ReceivedCPChecking	CPCheckFinished / CollectEndReceived != TRUE && CPChanged == TRUE	SlaveInit
37	ReceivedCPChecking	CPCheckFinished / CollectEndReceived != TRUE && CPChanged != TRUE	CPWCReceiveWaiting
38	CPWCReceiveWaiting	CollectEnd == TRUE => MasterNode = CPDMaster(); ReceivingNodes = DestNodeNum()	CPNodeTypeSelect
39	CPWCReceiveWaiting	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) == 0 => CPIDUpdate(CPID)	SlaveEnd
40	CPWCReceiveWaiting	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) != 0 => CPIDUpdate(CPID); CPCheck(CP);	ReceivedCPChecking
41	CPWCReceiveWaiting	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) == 0 => CPWCR.req(CheckResult = CPNotReceived)	SlaveInit
42	CPWCReceiveWaiting	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) != 0 => CPWCR.req(CheckResult = CPCheckResult)	SlaveEnd
43	SlaveEnd	CollectEnd == TRUE => MasterNode = CPDMaster(); ReceivingNodes = DestNodeNum()	CPNodeTypeSelect
44	SlaveEnd	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) == 0 => CPIDUpdate(CPID)	SlaveEnd
45	SlaveEnd	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) != 0 => CPIDUpdate(CPID); CPCheck(CP);	ReceivedCPChecking
46	SlaveEnd	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) == 0	SlaveInit

#	Current state	Event/condition => action	Next state
		=> CPWCR.req(CheckResult = CPNotReceived)	
47	SlaveEnd	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecevNodeList) == TRUE && CPComp(CP) != 0 => CPWCR.req(CheckResult = CPCheckResult)	SlaveEnd

8.1.5.2 Functions

Functions enabled in Connection parameter dist are shown in Table 79.

Table 79 – Function list of connection control

Name	Description
CPUpdate(NodeId, Param)	Updates common parameters.
CreateCPID(NodeId)	Generates common parameter ID by using NodeId and time.
CPIDUpdate(CPID)	Updates common parameter ID (CPID).
CPDMaster()	Determines whether the node performs parameter delivery. If yes, return TRUE, if not, return FALSE. Whether a node performs parameter delivery is determined as follows. 1. If it is a management node, the node performs parameter delivery. 2. If there is no management node and the cyclic transmission is being performed, and if the node number of the node is the least one among the nodes under the cyclic transmission, the node performs parameter delivery. 3. If there is no management node, nor any node that is performing the cyclic transmission, and if the node retains common parameters and the node number of the node is the least one among the nodes that retain common parameters, the node performs parameter delivery.
HasValidCP()	Compares the common parameter ID of parameter delivered node with the common parameter ID that is retained. If they are the same, return TRUE, if not, return FALSE.
DestNodeNum()	Returns the number of nodes to which parameters need to be sent. Whether parameters need to be sent or not is determined as follows. The common parameter ID of a node is 0. The common parameter ID of a node is different from the common parameter ID of the common parameter which is to be sent.
CreateDestNodeList()	Generates the list of nodes to which parameters need to be sent. Whether parameters need to be sent or not is determined in the same way as DestNodeNum().
CreateResultCheckList()	Generates the list of nodes to which the result of the parameter sending need to be checked. Whether to send parameters or not is determined in the same way as DestNodeNum().
AddDestNode(NodeId)	Adds the nodes, to which parameter need to be sent, to the node list generated in CreateDestNodeList().
RemoveDestNode(NodeId)	Deletes a specified node from the node list generated in CreateDestNodeList().
IncludeThisNode(RecvNodeList)	Determines whether the node is included in RecvNodeList. If yes, return TRUE, if not, return FALSE.
CPComp(CP)	Compares the common parameter given as CP with the common parameter that is retained. If they are same, return 0, if not, return any number other than 0.
CPCheck(CP)	Checks if the common parameter given as CP has been properly received. After checking, the result (CPCheckOK or CPCheckNG) is entered into CPCheckResult, and CPCheckFinished event occurs.

8.1.5.3 Mapping of internal service

The internal service that common parameter dist issues or receives is mapped into the service for acyclic transmission. Internal service and service mapping to acyclic transmission are shown in Table 80.

Table 80 – Mapping of internal service and acyclic transmission service

Internal service	Acyclic Transmission service	Parameter
CPW.req	AC Param Send.req	Request Type:Push Request Data Type:Parameer Distribution Data: Refer to Table 17.
CPW.ind	AC Param Send.ind	Request Type:Push Request Data Type:Parameer Distribution Data: Refer to Table 17.
CPWC.req	AC Param Send.req	Request Type:Push Request Data Type:Parameer Distribution Data:Refer to Table 18.
CPWC.ind	AC Param Send.ind	Request Type:Push Request Data Type:Parameer Distribution Data: Refer to Table 18.
CPWCR.req	AC Param Send.req	Request Type:Push Request Data Type:Parameer Distribution Data:Refer to Table 19.
CPWCR.ind	AC Param Send.ind	Request Type:Push Request Data Type:Parameer Distribution Data:Refer to Table 19.

8.2 ARPM type F

8.2.1 Overview

The structure of ARPM is shown in Figure 21. The continuous line represents a service issue, and the dashed line represents a linkage between protocol machines using parameters and others.

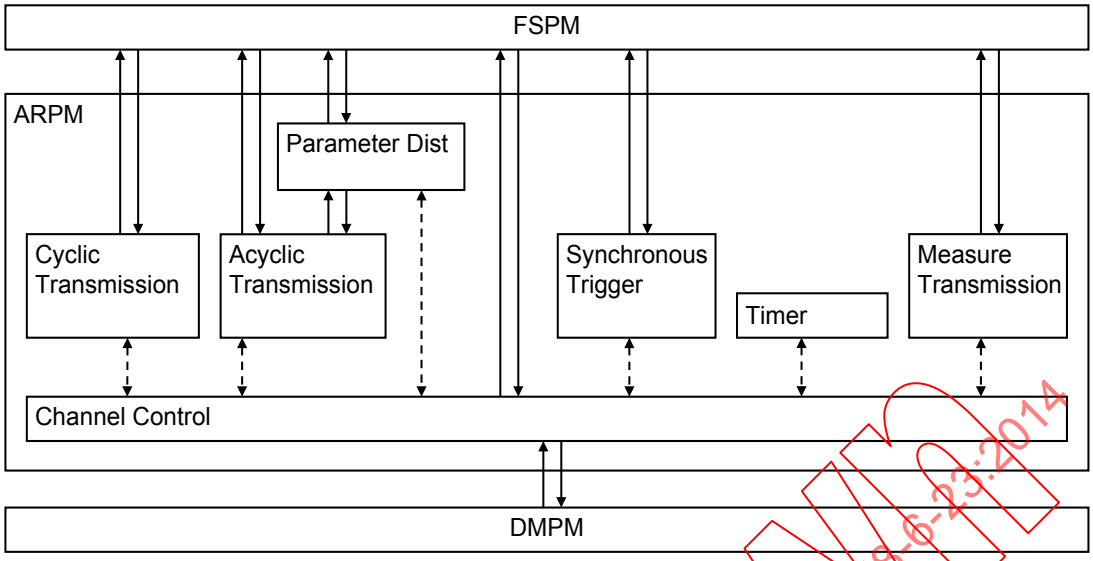


Figure 21 – Structure of ARPM F

8.2.2 Acyclic transmission

8.2.2.1 Primitive definition

FSPM issues an AC Send.req service or AC Send ND.req service to Acyclic transmission. Parameter dist issues an AC Send.req service or AC Send ND.req service to Acyclic transmission. Acyclic transmission issues an AC Send.ind service to FSPM. Acyclic transmission issues an AC Param Send.ind service to Parameter dist.

8.2.2.2 Acyclic transmission state machine

The states of the Acyclic transmission state machine are shown in Table 81 and the details are specified in Table 82.

Table 81 – Acyclic transmission states

Name	Description
IDLE	Waiting for the start of acyclic transmission
ACSENDER	Acyclic transmission in progress
ACSENDER_ND	Acyclic transmission according to the node scan limiting method is in progress.

Table 82 – Acyclic transmission state table

#	Current state	Event/condition => action	Next state
1	IDLE	/ ACTicket == TRUE && TokenHopCounter ≥ TraAvailHopCounter && TraAllows > 0 =>	ACSENDER
2	IDLE	/ ACTicketND == TRUE && TraAllowsND > 0 =>	ACSENDER_ND
3	IDLE	/ ACTicket == TRUE && (TokenHopCounter < TraAvailHopCounter || TraAllows == 0) =>	IDLE

#	Current state	Event/condition => action	Next state
		ACTicket = FALSE	
4	IDLE	/ ACTicketND == TRUE && TraAllowsND == 0 => ACTicketND = FALSE	IDLE
5	ACSENDER	/ QueLen() > 0 && TraAllows > 0 => pdu = Deque(); SendACyclic (pdu); TraAllows = TraAllows - 1	ACSENDER
6	ACSENDER_ND	/ QueLenND() > 0 && TraAllowsND > 0 => pdu = DequeND(); SendACyclic (pdu); TraAllowsND = TraAllowsND - 1	ACSENDER
7	ACSENDER	/ TraAllows == 0 => TraLastHopCounter = TokenHopCounter; ACTicket = FALSE	IDLE
8	ACSENDER_ND	/ TraAllowsND == 0 => ACTicketND = FALSE	IDLE
9	ACSENDER	/ QueLen() < 0 => ACTicket = FALSE	IDLE
10	ACSENDER_ND	/ QueLenND() < 0 => ACTicketND = FALSE	IDLE
11	IDLE	AC Send.req(netno, nodeno, actype, data) => pdus = CreateTransient-PDU(netno, nodeno, actype, data); Enque(pdus)	IDLE
12	IDLE	AC Send NDreq(netno, nodeno, actype, data) => pdus = CreateTransient-PDU(netno, nodeno, actype, data); EnqueND(pdus)	IDLE
13	ACSENDER	AC Send.req(netno, nodeno, actype, data) => pdus = CreateTransient-PDU(netno, nodeno, actype, data); Enque(pdus)	ACSENDER
14	ACSENDER_ND	AC Send NDreq(netno, nodeno, actype, data) => pdus = CreateTransient-PDU(netno, nodeno, actype, data); EnqueND(pdus)	ACSENDER_ND
15	IDLE	ACReceived(pdu) => AC Send.ind(pdu)	IDLE
16	ACSENDER	ACReceived(pdu) => AC Send.ind(pdu)	ACSENDER
17	ACSENDER_ND	ACReceived(pdu) => AC Send.ind(pdu)	ACSENDER_ND

#	Current state	Event/condition => action	Next state
18	IDLE	ACStopSending() => ACTicket = FALSE; ACTicketND = FALSE;	IDLE
19	ACSENDER	ACStopSending() => ACTicket = FALSE; ACTicketND = FALSE;	IDLE
20	ACSENDER_ND	ACStopSending() => ACTicket = FALSE; ACTicketND = FALSE;	IDLE

8.2.2.3 Functions

Functions enabled in acyclic transmission are shown in Table 83.

Table 83 – Acyclic transmission functions

Name	Description
CreateTransient-PDU (netno, nodeno, actype, data)	Generates the transient1-PDU when actype==Transient1, transientAck-PDU when actype==TransientAck, transient2-PDU when actype==Transient2, parameter-PDU when actype==Parameter, paramCheck-PDU when actype==ParamCheck, and timer-PDU when actype==Timer. Generates multiple transientAck-PDUs in accordance with the size of data given as an argument when actype==TransientAck.
Eeque(pdus)	Queues pdus given as arguments. Queues in the order of the pdu seqNumber when multiple pdus are given.
EequeND(pdus)	Queues pdu given as arguments (for limiting the number of node unit transmissions). When more than one pdu is specified, queuing is performed in the order of the seqNumber of the pdus.
Deque()	Retrieves the start of the queue.
DequeND()	Retrieves the start of the queue (for limiting the number of node unit transmissions).
QueLen()	Returns the size of the queue.
QueLenND()	Returns the size of the queue (for limiting the number of node unit transmissions).

8.2.2.4 Variables

Functions enabled in acyclic transmission are shown in Table 84.

Table 84 – Acyclic transmission variables

Name	Description
ACTicket	Flag indicating that Acyclic transmission is allowed
ACTicketND	Flag indicating that Acyclic transmission for limiting the number of node unit transmissions is allowed
TokenHopCounter	Token passing counter
TraAllows	Number of times transient transmission is allowed
TraAllowsND	Number of times transient transmission is allowed (node unit restriction on number of transmissions)
TraAvailHopCounter	Minimum value of token passing counter
TraLastHopCounter	Last transmission token passing counter

8.2.3 Cyclic transmission

8.2.3.1 Primitive definition

FSPM issues a CT Update.req service to Cyclic transmission. Cyclic transmission issues a CT Update.ind service to FSPM.

8.2.3.2 Cyclic transmission state machine

The states of the Cyclic transmission state machine are shown in Table 85 and the details are specified in Table 86.

Table 85 – Cyclic transmission states

Name	Description
IDLE	Waiting for the start of Cyclic Transmission
RWrRWwSENDER	RWr or RWw transmission in progress
RXRYSENDER	RX or RY transmission in progress

Table 86 – Cyclic transmission state table

#	Current state	Event/condition => action	Next state

1	IDLE	/
		CTicket == TRUE
		=>
		seqno = 1;
		sending = FALSE;
		UpdateWaiting = FALSE;
		if Length(RXRY) == 0
		then RXRYSent = TRUE
		else RXRYSent = FALSE;
		if Length(RWrRWw) == 0
		then RWrRWwSent = TRUE
		else RWrRWwSent = FALSE
			RWrRWwSENDER

2	RWrRWwSENDER	/
		RWrRWwSent != TRUE && sending == FALSE
		=>
		pdu = CreateCyclicDataRWrRWw-PDU(seqno, type, RWrRWw);
		SendCyclic(pdu);
		seqno = seqno + 1;
		sending == TRUE
			RWrRWwSENDER

3	RWrRWwSENDER	/ RWrRWwSent != TRUE && sending == TRUE && Length(RemainingRWrRWw) > 0 => pdu = CreateCyclicDataRWrRWw -PDU(seqno, type, RemainingRWrRWw); SendCyclic(pdu); seqno = seqno + 1	RWrRWwSENDER
4	RWrRWwSENDER	/ RWrRWwSent != TRUE sending == TRUE && Length(RemainingRWrRWw) == 0 => RWrRWwSent = TRUE; sending = FALSE	RXRYSENDER

5	RXRYSENDER	/
		RXRYSent != TRUE && sending == FALSE
		=>
		pdu = CreateCyclicDataRXRY-PDU(seqno, type, RXRY);
		SendCyclic(pdu);
		seqno = seqno + 1;
		sending == TRUE;
			RXRYSENDER

#	Current state	Event/condition => action	Next state
6	RXRYSENDER	/ RXRYSent != TRUE && sending == TRUE && Length(RemainingRXRY) > 0 => pdu = CreateCyclicDataRXRY-PDU(seqno, type, RemainingRXRY); SendCyclic(pdu); seqno = seqno + 1	RXRYSENDER
7	RXRYSENDER	/ RXRYSent != TRUE && sending == TRUE && Length(RemainingRXRY) == 0 => RXRYSent = TRUE; sending = FALSE; CTicket = FALSE	IDLE
8	IDLE	CT Update.req(Data Type, Offset Address, Size, Data) => Update(Data Type, Offset Address, Size, Data)	IDLE
9	IDLE	CT Update.req(Data Type, Offset Address, Size, Data) => Update(Data Type, Offset Address, Size, Data)	IDLE
10	IDLE	/ UpdateWaiting == TRUE => Update(Data Type, Offset Address, Size, Data); UpdateWaiting = FALSE	IDLE
11	RXRYSENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	RXRYSENDER
12	RWrRWwSEN DER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	RWrRWwSEN DER
13	Any (any state)	Creceived(cyclicDataRX-PDU) => When in sequential number order: ReceivedRXRY = RetrieveRXRY(cyclicDataRX-PDU); CT Update.ind(RXRY, Offset Address, Size, ReceivedRXRY) When not in sequential number order: Discarded.	Any (no state change)
14	Any (any state)	CReceived(cyclicDataRY-PDU) => When in sequential number order: ReceivedRXRY = RetrieveRXRY(cyclicDataRY-PDU); CT Update.ind(RXRY, Offset Address, Size, ReceivedRXRY) When not in sequential number order: Discarded.	Any (no state change)
15	Any (any state)	CReceived(CyclicDataRWr-PDU) => When in sequential number order: ReceivedRWrRWw = RetrieveRWrRWw(CyclicDataRWr- PDU); CT Update.ind(RWrRWw, Offset Address, Size, ReceivedRWrRWw) When not in sequential number order: Discarded.	Any (no state change)
16	Any (any state)	CReceived(CyclicDataRWw-PDU) => When in sequential number order: ReceivedRWrRWw = RetrieveRWrRWw(CyclicDataRWw- PDU); CT Update.ind(RWrRWw, Offset Address, Size, ReceivedRWrRWw) When not in sequential number order: Discarded	Any (no state change)
17	Any (any state)	CStopSending() => CTicket = FALSE	IDLE

8.2.3.3 Functions

Functions enabled in Cyclic transmission are shown in Table 87.

Table 87 – Cyclic transmission functions

Name	Description
Length	Finds the size of the data given by the argument.
Update	Updates the data of the RXRY and RWrRWw held by Cyclic Transmission for transmission.
CreateCyclicDataRXRY-PDU (seqno, type, RXRY)	Generates the cyclicDataRX-PDU when type==RX, and cyclicDataRY-PDU when type==RY. Sets the seqno in bits 6..1 of seqNumber of the cyclicDataRX-PDU or cyclicDataRY-PDU. Uses 1468 octets from the start when RXRY given as the argument exceeds 1468 octets. The remaining data are set as RemainingRXRY. Sets 1 in bit 7 of seqNumber of the cyclicDataRX-PDU or cyclicDataRY-PDU when Length (RemainingRXRY) == 0, indicating that the PDU is the last PDU.
CreateCyclicDataRWrRWw-PDU (seqno, type, RWrRWw)	Generates the cyclicDataRWr-PDU when type==RWr, and cyclicDataRWw-PDU when type==RWw. Sets the seqno in bits 6..1 of seqNumber of the cyclicDataRWr-PDU or cyclicDataRWw-PDU. Uses 1468 octets from the start when RXRY given as the argument exceeds 1468 octets. The remaining data are set as RemainingRWrRWw. Sets 1 in bit 7 of seqNumber of the cyclicDataRWr-PDU or cyclicDataRWw-PDU when Length (RemainingRWrRWw) == 0, indicating that the PDU is the last PDU.
RetrieveRXRY(PDU)	Retrieves the Offset Address, Size, and RXRY data from PDU.
RetrieveRWrRWw(PDU)	Retrieves the Offset Address, Size, and RWrRWr data from PDU.

8.2.3.4 Variables

Functions enabled in Cyclic transmission are shown in Table 88.

Table 88 – Cyclic transmission variables

Name	Description
CTicket	Flag indicating that cyclic transmission is allowed.

8.2.4 Channel control

8.2.4.1 Channel control state machine

The states of the Channel control state machine for the Mater station are described in Table 89 with the details specified in Table 91 through Table 105.

The states of the channel control state machine for the Slave station are described in Table 90 with the details specified in Table 106 through Table 111.

Table 89 – Master station channel control states

Name	Description
MasterDown	Not started
Listen	After power ON
MasterArbitration	Selects transmission control manager
PrimaryMasterScatterTD	Detects connected node as transmission control manager
PrimaryMasterSettingUp	Sets up token passing path and sets the path in slaves as transmission control manager.
PrimaryMasterHoldToken	Holding token as transmission control manager

Name	Description
PrimaryMasterSolicitToken	Waiting for token as transmission control manager
PrimaryMasterInviting	Detects node at return to system
MasterMeasurement	Measures transmission path delays.
MasterWaitTD	Waiting for return to system as a node that is not the transmission control manager
MasterWaitSetup	Waiting for token passing path setup as a node that is not the transmission control manager
MasterSolicitToken	Waiting for token as a node that is not the transmission control manager
MasterHoldToken	Holding the token as a node that is not the transmission control manager

Table 90 – Slave station channel control states

Name	Description
SlaveDown	Not started
SlaveWaitTD	Disconnected
SlaveWaitSetup	Waiting for transmission path information
SlaveSolicitToken	Waiting for token

Table 91 – Master station state table – MasterDown

#	Current state	Event/condition => action	Next state
1	MasterDown	Power ON => ListenTimer startup; ChannelGroup = NULL	Listen

Table 92 – Master station state table – Listen

#	Current state	Event/condition => action	Next state
1	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 => pdu = CreatePersuasion(); Send.req (sport, DA, pdu) to all ports other than rport; ListenTimer stop; TR8 = TRUE;	MasterWaitTD

2	Listen	Receive.ind(rport, DA, SA, ftype, pdu)
		/
		DA==BROADCAST && SA !=MyAddr && ftype == 0x11
		=>
		ListenTimer stop;
		pdu = CreateTestDataAck();
		Send.req(rport, SA, pdu);
		pdu = CreateTestData-PDU();
		Send.req (sport, DA, pdu) to all ports other than rport;
		TR9 = TRUE
			MasterWaitSetup

3	Listen	Receive.ind (rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 => ListenTimer restart	Listen
4	Listen	Receive.ind(rport, DA, SA, ftype, pdu) /	Listen

#	Current state	Event/condition => action	Next state
		DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	
5	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr &&ftype == 0x13 => ListenTimer restart	Listen
6	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	Listen
7	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x14 => ListenTimer restart	Listen
8	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	Listen
9	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport; ListenTimer restart	Listen
10	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) || ftype == 0x20 || ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ListenTimer restart	Listen
11	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ListenTimer restart	Listen
12	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	Listen
13	Listen	ListenTimer time-out => ListenTimer stop; TR3 = TRUE	PrimaryMaster ScatterTD

Table 93 – Master station state table – MasterArbitration

#	Current state	Event/condition => action	Next state
1	MasterArbitration	/ TR2 == TRUE => SynchronizationMaster=FALSE TR2 = FALSE; RvLastArbTimer startup; ChannelGroup = NULL; SendArbTimer startup; pdu = CreatePersuasion(); Send.req (port, BROADCAST, pdu) to all ports	MasterArbitration
2	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; RvLastArbTimer restart	MasterArbitration
3	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than report; SendArbTimer stop; RvLastArbTimer stop; TR8 = TRUE	MasterWaitTD
4	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitration
5	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitration
6	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitration
7	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / SA != MyAddrftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitration
8	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) ==TRUE && SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) || ftype == 0x20 || ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitration
9	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => Deliver received frame to upper layer	MasterArbitration
10	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitration

#	Current state	Event/condition => action	Next state
11	MasterArbitration	SendArbTimer time-out => pdu = CreatePersuasion(); Send.req (port, BROADCAST, pdu) to all ports; SendArbTimer restart	MasterArbitration
12	MasterArbitration	RvLastArbTimer timeout => SendArbTimer stop; RvLastArbTimer stop; TR3 = TRUE	PrimaryMaster ScatterTD

Table 94 – Master station state table – PrimaryMasterScatterTD

#	Current state	Event/condition => action	Next state
1	PrimaryMaster ScatterTD	/ TR3 == TRUE && RingCheck == FALSE => TR3 =FALSE; Nodes=0; ANodes=0; pdu = CreateTestData(); Send.req (port, BROADCAST, pdu) to all ports; ChannelGroup = MCAST(MyAddr); TDAckTimer startup SynchronizationMaster=TRUE;	PrimaryMaster ScatterTD
2	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 &&CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; TDAckTimer stop; TR2 = TRUE	MasterArbitration
3	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr &&ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; TDAckTimer stop; TR8 = TRUE	MasterWaitTD
4	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => TDAckTimer stop; TR2 = TRUE	MasterArbitration
5	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) == Low => Nodes++	PrimaryMaster ScatterTD
6	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) != Low => TDAckTimer stop; TR2 = TRUE	MasterArbitration
7	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster ScatterTD
8	PrimaryMaster	Receive.ind(rport, DA, SA, ftype, pdu)	PrimaryMaster

#	Current state	Event/condition => action	Next state
	ScatterTD	/ DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	ScatterTD
9	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster ScatterTD
10	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster ScatterTD
11	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	PrimaryMaster ScatterTD
12	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport MyStatusReceived(pdu)	PrimaryMaster ScatterTD
13	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	PrimaryMaster ScatterTD
15	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu)	PrimaryMaster ScatterTD
16	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster ScatterTD
15	PrimaryMaster ScatterTD	TDAckTimer time-out / Nodes > 0 => TDAckTimer stop; TR4 = TRUE	PrimaryMaster SettingUp
16	PrimaryMaster ScatterTD	TDAckTimer time-out / Nodes == 0 => TDAckTimer stop; TR3 = TRUE	PrimaryMaster ScatterTD
17	PrimaryMaster ScatterTD	/ TR3 == TRUE && RingCheck == TRUE => TR3 =FALSE; Nodes=0;ANodes=0; Invalidate port 2; pdu = CreateTestData(); Send.req (port, BROADCAST, pdu) to port 1; ChannelGroup = MCAST(MyAddr); TDAckTimer startup	PrimaryMaster ScatterTD

#	Current state	Event/condition => action	Next state
18	PrimaryMaster ScatterTD	TDAckTimer time-out / Nodes > 0 && RingCheck == TRUE && IsValidPort(port 2) != TRUE => Invalidate port 1; Validate port 2; pdu = CreateTestData(); Send.req (port, BROADCAST, pdu) to port 2; ChannelGroup = MCAST(MyAddr); TDAckTimer startup	PrimaryMaster ScatterTD
19	PrimaryMaster ScatterTD	TDAckTimer time-out / Nodes > 0 && RingCheck == TRUE && IsValidPort(port 1) != TRUE => TDAckTimer stop; TR4 = TRUE	PrimaryMaster SettingUp

Table 95 – Master station state table – PrimaryMasterSettingUp

#	Current state	Event/condition => action	Next state
1	PrimaryMaster SettingUp	/ TR4 == TRUE && RingCheck == FALSE => TR4 = FALSE; SARcvd=0; Nodes+=ANodes; CTRProgress = NotComplete; CreateTokenRoute();pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) to all ports; SAckTimer startup	PrimaryMaster SettingUp
2	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; SAckTimer stop; TR2 = TRUE	MasterArbitration
3	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; SAckTimer stop; TR8 = TRUE	MasterWaitTD
4	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => SAckTimer stop; TR2 = TRUE	MasterArbitration
5	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SettingUp
6	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SettingUp
7	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x14 &&	PrimaryMaster SettingUp

#	Current state	Event/condition => action	Next state
		SARcvd+1 < SetupNodes => SARcvd++;pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) to all ports	
8	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / RingChecked == FALSE && DA == MyAddr && SA != MyAddr && ftype == 0x14 && SARcvd+1== SetupNodes => SAckTimer stop; TR5 = TRUE	PrimaryMaster HoldToken
9	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SettingUp
10	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SettingUp
11	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	PrimaryMaster SettingUp
12	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu)	PrimaryMaster SettingUp
13	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	PrimaryMaster SettingUp
14	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu)	PrimaryMaster SettingUp
15	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SettingUp
16	PrimaryMaster SettingUp	SAckTimer time-out => SAckTimer stop; pdu = CreatePersuasion(); Send.req (port, BROADCAST, pdu) to all ports TR3 = TRUE	PrimaryMaster ScatterTD
17	PrimaryMaster SettingUp	/ TR4 == TRUE && RingCheck == TRUE && IsValidPort(port 1) == TRUE && IsValidPort(port 2) == TRUE && IsSlavePortInvalidated() == FALSE => TR4 = FALSE; SARcvd=0;	PrimaryMaster SettingUp

#	Current state	Event/condition => action	Next state
		Nodes+=ANodes; CTRProgress = NotComplete; if CompareRoutes() == TRUEthen Invalidate port 2; CreateTokenRoute(); pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) to all ports; SAckTimer startup	
18	PrimaryMaster SettingUp	/ TR4 == TRUE && RingCheck == TRUE && (IsValidPort(port 1) == FALSE || IsValidPort(port 2) == FALSE) && IsSlavePortInvalidated() == FALSE => TR4 = FALSE;SARcvd=0; Nodes+=ANodes;CTRProgress = NotComplete; CreateTokenRoute(); pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) to all ports; SAckTimer startup	PrimaryMaster SettingUp
19	PrimaryMaster SettingUp	/ RingCheck == TRUE && IsValidPort(port 1) == TRUE && IsValidPort(port 2) == TRUE && IsSlavePortInvalidated() == TRUE => TR4 = FALSE;SARcvd=0; Nodes+=ANodes; CTRProgress = NotComplete; CreateTokenRoute(); pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) to all ports; SAckTimer startup	PrimaryMaster SettingUp
20	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / RingChecked == TRUE && DA == MyAddr && SA != MyAddr && ftype == 0x14 && SARcvd+1 == SetupNodes => SAckTimer stop; Invalidate port connected to invalid port of slave; pdu = CreateSetup(); Send.req (port, MAC address of slave with invalid port, pdu); SlavePortValidating=TRUE; SAckTimer startup	PrimaryMaster SettingUp
21	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / RingChecked == TRUE && DA == MyAddr && SA != MyAddr && ftype == 0x14 && SlavePortValidating == TRUE => SlavePortValidating = FALSE; SAckTimer stop; TR5 = TRUE	PrimaryMaster HoldToken

Table 96 – Master station state table – PrimaryMasterHoldToken

#	Current state	Event/condition => action	Next state
1	PrimaryMaster HoldToken	/ TR5 == TRUE => TR5 = FALSE;SetInvitationFlag(); SetMeasureFlag(); CTicket = FALSE;sents = 1; if token-PDU.traAllows > 0 then TraAllows = token-PDU.traAllows;	PrimaryMaster HoldToken

#	Current state	Event/condition => action	Next state
		TraAvailHopCounter = 0; else TraAvailHopCounter = token-PDU.traLastHopCounter; TraLastHopCounter = 0; TraAllows = Max Transients per round; TokenHopCounter = 0; TraAllowsND = GetTraAllowsND(); SetNextPhase(PrimaryMasterHoldToken,0);	
2	PrimaryMaster HoldToken	/ PMChangeFlag == ON => TR2 = TRUE	MasterArbitration
3	PrimaryMaster HoldToken	/ PH == 1 && InvitationFlag == OFF && MyStatusSendTimingFlag == ON => InvitationFlag == NULL; MyStatusSendTimingFlag = OFF pdu = CreateMyStatus(); Send.req (port, ChannelGroup, pdu) to all ports; MyStatusSend(pdu); Enque(pdu); CTicket = TRUE; SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMaster HoldToken
4	PrimaryMaster HoldToken	/ PH == 1 && InvitationFlag == ON => InvitationFlag == NULL; TR7 = TRUE	PrimaryMasterInviting
5	PrimaryMaster HoldToken	/ PH == 10 && CTicket == FALSE => SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMaster HoldToken
6	PrimaryMaster HoldToken	/ PH == 11 => ACTicketND = TRUE; SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMaster HoldToken
7	PrimaryMaster HoldToken	/ PH == 12 => ACTicket = TRUE; SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMaster HoldToken
8	PrimaryMaster HoldToken	/ PH == 20 && ((ACTicketND == FALSE) && (ACTIcket == FALSE)) => SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMaster Inviting

9	PrimaryMaster HoldToken	/
		PH == 3 &&
		sents < Multiple Transmit
		=>
		sents = sents + 1;
		PduNum = QueLen();
		while (PduNum > 0)
		pdu = Deque();
		Send.req (port, ChannelGroup, pdu) to all ports;
		Enque(pdu);
		PduNum = PduNum -1
			PrimaryMaster HoldToken

10	PrimaryMaster HoldToken	/ PH == 3 && sents == Multiple Transmit => QueDelete(); pdu = CreateToken(); Send.req (port, ChannelGroup, pdu) to all ports;	PrimaryMaster HoldToken

#	Current state	Event/condition => action	Next state
		Enque(pdu); sents = 1; PH = 4	
11	PrimaryMaster HoldToken	/ PH == 4 && sents < Multiple Token => pdu = Deque(); Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu); sents = sents + 1	PrimaryMaster HoldToken
12	PrimaryMaster HoldToken	/ PH == 4 && sents == Multiple Token => QueDelete(); PH = 0; TR6 = TRUE	PrimaryMaster SolicitToken
13	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; TR2 = TRUE	MasterArbitration
14	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; TR8 = TRUE	MasterWaitTD
15	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => TR2 = TRUE	MasterArbitration
16	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster HoldToken
17	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster HoldToken
18	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster HoldToken
19	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster HoldToken
20	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	PrimaryMaster HoldToken
21	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport;	PrimaryMaster HoldToken

#	Current state	Event/condition => action	Next state
		MyStatusReceived(pdu)	
22	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	PrimaryMaster HoldToken
23	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu)	PrimaryMaster HoldToken
24	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster HoldToken
25	PrimaryMaster HoldToken	SendCyclic(pdu) => Send.req (sport, ChannelGroup, pdu) to all ports; Enque(pdu)	PrimaryMaster HoldToken
26	PrimaryMaster HoldToken	/ MeasureFlag == ON => MeasureFlag = NULL TR12 = TRUE	MasterMeasurement

Table 97 – Master station state table – PrimaryMasterSolicitToken

#	Current state	Event/condition => action	Next state
1	PrimaryMaster SolicitToken	/ TR6 == TRUE => TR6 = FALSE; NetWatchTimer startup; SetNewTokenFlag()	PrimaryMaster SolicitToken
2	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; NetWatchTimer stop; TR2 = TRUE	MasterArbitration
3	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; NetWatchTimer stop; TR8 = TRUE	MasterWaitTD
4	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => NetWatchTimer stop; TR2 = TRUE	MasterArbitration
5	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SolicitToken

#	Current state	Event/condition => action	Next state
6	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SolicitToken
7	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SolicitToken
8	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) to all ports other than rport; NetWatchTimer stop; AddReceivedToken(pdu.tseq_no); TR5 = TRUE	PrimaryMaster HoldToken
9	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== TRUE => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SolicitToken
10	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport; NetWatchTimer restart	PrimaryMaster SolicitToken
11	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
12	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
13	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
14	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
15	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr =>	PrimaryMaster SolicitToken

#	Current state	Event/condition => action	Next state
		Send.req (sport, DA, pdu) to all ports other than rport	
16	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() != MyAddr && NewTokenFlag == ON && RingCheck == FALSE => NetWatchTimer stop; pdu = CreateToken(); Send.req (sport, LostTokenDst(), pdu) to all ports; NetWatchTimer restart	PrimaryMaster SolicitToken
17	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() != MyAddr && NewTokenFlag == OFF && RingCheck == FALSE => NetWatchTimer stop; pdu = CreatePersuasion(); Send.req (sport, BROADCAST, pdu) to all ports; TR3 = TRUE	PrimaryMaster ScatterTD
18	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() == MyAddr => NetWatchTimer stop; pdu = CreateToken(); Send.req (sport, MyAddr, pdu) to all ports; TR5 = TRUE	PrimaryMaster HoldToken
19	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() != MyAddr && RingCheck == TRUE => NetWatchTimer stop; Failure location assessment; pdu = CreateSetup (Invalidate failure location port); Send.req (port, failure upstream node, pdu) to valid port; UpperSlavePortInvalidating = TRUE; NetWatchTimer restart	PrimaryMaster SolicitToken
20	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x14 && RingCheck == TRUE && UpperSlavePortInvalidating == TRUE => NetWatchTimer stop; UpperSlavePortInvalidating == FALSE Validate invalid port; pdu = CreateSetup (Invalidate failure location port); Send.req (port, failure location downstream node, pdu) to validated port; LowerSlavePortInvalidating == TRUE NetWatchTimer restart	PrimaryMaster SolicitToken
21	PrimaryMaster SolicitToken	eceive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x14 && RingCheck == TRUE && LowerSlavePortInvalidating == TRUE => NetWatchTimer stop; LowerSlavePortInvalidating = FALSEpdu = CreatePersuasion(); Send.req (sport, BROADCAST, pdu) to all ports; TR3 = TRUE	PrimaryMaster ScatterTD

Table 98 – Master station state table – PrimaryMasterInviting

#	Current state	Event/condition => action	Next state
1	PrimaryMasterInviting	/ TR7 == TRUE && RingCheck == FALSE => TR7 = FALSE; TDAckTimer startup; ANodes = 0; pdu = CreateTestData(); Send.req (sport, BROADCAST, pdu) to all ports	PrimaryMasterInviting
2	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; TDAckTimer stop; TR2 = TRUE	MasterArbitration
3	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; TDAckTimer stop; TR8 = TRUE	MasterWaitTD
4	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => TDAckTimer stop; TR2 = TRUE	MasterArbitration
5	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) == Low => ANodes++	PrimaryMasterInviting
6	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) != Low => TDAckTimer stop; TR2 = TRUE	MasterArbitration
7	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterInviting
8	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterInviting
9	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterInviting
10	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterInviting
11	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr &&	PrimaryMasterInviting

#	Current state	Event/condition => action	Next state
		(ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu);	
12	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu);	PrimaryMasterInviting
13	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu);	PrimaryMasterInviting
14	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu);	PrimaryMasterInviting
15	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterInviting
16	PrimaryMasterInviting	TDAckTimer time-out / ANodes > 0 => TDAckTimer stop; TR4 = TRUE	PrimaryMaster SettingUp
17	PrimaryMasterInviting	TDAckTimer time-out / ANodes == 0 => TDAckTimer stop; TR5 = TRUE	PrimaryMaster HoldToken
18	PrimaryMasterInviting	/ TR7 == TRUE && RingCheck == TRUE && SlavePortStateChanged() == TRUE => TR7 = FALSE; pdu = CreateSetup (Validate port with port status change); Send.req (sport, port with port status change, pdu) to all ports; SlavePortValidating = TRUE	PrimaryMasterInviting
19	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x14 && RingCheck == TRUE && SlavePortValidating = TRUE => TDAckTimer startup; ANodes = 0; pdu = CreateTestData(); Send.req (sport, BROADCAST, pdu) to all ports	PrimaryMasterInviting

Table 99 – Master station state table – MasterWaitTD

#	Current state	Event/condition => action	Next state
1	MasterWaitTD	/ TR8 == TRUE =>	MasterWaitTD

#	Current state	Event/condition => action	Next state
		TR8 = FALSE; TDWaitTimer startup; ChannelGroup = NULL	
2	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) to all ports other than rport; TDWaitTimer restart	MasterWaitTD

3	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu)
		/
		DA==BROADCAST && SA !=MyAddr && ftype == 0x11
		=>
		TDAckTimer stop;
		pdu = CreateTestDataAck();Send.req(rport, SA, pdu);
		pdu = CreateTestData();
		Send.req (rport, DA, pdu) to all ports other than rport;
		TR9 = TRUE
			MasterWaitSet up

4	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD
5	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD
6	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD
7	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport; TDWaitTimer restart	MasterWaitTD
8	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && && SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) || ftype == 0x20 || ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD
9	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu);	MasterWaitTD
10	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD
11	MasterWaitTD	TDWaitTimer time-out => TDWaitTimer stop; TR3 = TRUE	PrimaryMaster ScatterTD

Table 100 – Master station state table – MasterWaitSetup

#	Current state	Event/condition => action	Next state
1	MasterWaitSetup	/ TR9 == TRUE => TR9 = FALSE; SWaitTimer startup	MasterWaitSetup
2	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; SWaitTimer stop; TR2 = TRUE	MasterArbitration
3	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; SWaitTimer stop; TR8 = TRUE	MasterWaitTD

4	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu)
		/
		DA==BROADCAST && SA !=MyAddr && ftype == 0x11
		=>
		pdu = CreateTestDataAck();
		Send.req(rport, SA, pdu);
		pdu = CreateTestData();
		Send.req (rport, DA, pdu) to all ports other than rport;
		SWaitTimer restart
			MasterWaitSetup

5	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitSetup
6	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => SWaitTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, pdu); TR10 = TRUE	MasterSolicitToken
7	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitSetup
8	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitSetup
9	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport; SWaitTimer restart	MasterWaitSetup
10	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && && SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) || ftype == 0x20 ||	MasterWaitSetup

#	Current state	Event/condition => action	Next state
		ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport	
11	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu);	MasterWaitSetup
12	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitSetup
13	MasterWaitSetup	SWaitTimer time-out => SWaitTimer stop; TR3 = TRUE	PrimaryMasterScatterTD

Table 101 – Master station state table – MasterSolictToken (without Transmission path delay measurement)

#	Current state	Event/condition => action	Next state
1	MasterSolicitToken	/ TR10 == TRUE => TR10 = FALSE; PrevTSeqNo = NULL; LeaveTimer startup	MasterSolicitToken
2	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop	MasterArbitration
3	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop	MasterWaitTD
4	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (rport, DA, pdu) to all ports other than rport	MasterSolicitToken
5	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
6	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, pdu); LeaveTimer startup	MasterSolicitToken

#	Current state	Event/condition => action	Next state
7	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
8	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
9	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no); TR11 = TRUE	MasterHoldToken
10	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== TRUE => Send.req (sport, DA, pdu) to all ports other than rport;	MasterSolicitToken
11	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
12	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	MasterSolicitToken
13	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu)	MasterSolicitToken
14	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	MasterSolicitToken
15	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu)	MasterSolicitToken
16	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
17	MasterSolicitToken	LeaveTimer time-out => LeaveTimer stop;	MasterWaitTD

#	Current state	Event/condition => action	Next state
		TR8 = TRUE	

Table 102 – Master station state table – MasterSolictToken (with Transmission path delay measurement)

#	Current state	Event/condition => action	Next state
1	MasterSolicitToken	/ TR10 == TRUE => TR10 = FALSE; PrevTSeqNo = NULL; LeaveTimer startup	MasterSolicitToken
2	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop	MasterArbitration
3	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop	MasterWaitTD
4	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (rport, DA, pdu) to all ports other than rport	MasterSolicitToken
5	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
6	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, pdu); LeaveTimer startup	MasterSolicitToken
7	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
8	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
9	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no);	MasterHoldToken

#	Current state	Event/condition => action	Next state
		TR11 = TRUE	
10	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== TRUE => Send.req (sport, DA, pdu) to all ports other than rport;	MasterSolicitToken
11	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
12	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	MasterSolicitToken
13	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu)	MasterSolicitToken
14	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	MasterSolicitToken
15	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu)	MasterSolicitToken
16	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
17	MasterSolicitToken	LeaveTimer time-out => LeaveTimer stop; TR8 = TRUE	MasterWaitTD
18	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x40 => MeasureRecieved(rport, DA, SA, pdu)	MasterSolicitToken
19	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x41 => MeasureAckRecieved(rport, DA, SA, pdu)	MasterSolicitToken
20	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x42 => OffsetRecieved(rport, DA, SA, pdu)	MasterSolicitToken
21	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) /	MasterSolicitToken

#	Current state	Event/condition => action	Next state
		ftype == 0x43 => UpdateRecieved(rport, DA, SA, pdu)	

Table 103 – Master station state table – MasterHoldToken

#	Current state	Event/condition => action	Next state
1	MasterHoldToken	/ TR11 == TRUE => TR11 = FALSE; SetNextPhase(MasterHoldToken,0);	MasterHoldToken
2	MasterHoldToken	/ PH == 1 => MyStatusSendTimingFlag = OFF; pdu = CreateMyStatus(); Send.req(port,ChannelGroup,pdu) to all ports; MyStatusSend(pdu); CTicket = TRUE; TokenHopCounter = token-PDU.tokenHopCounter; TraAvailHopCounter = token-PDU.traAvailHopCounter; TraLastHopCounter = token-PDU.traLastHopCounter; TraAllows = token-PDU.traAllows; TraAllowsND = GetTraAllowsND(); SetNextPhase(MasterHoldToken,0);	MasterHoldToken
3	MasterHoldToken	/ PH == 10 && CTicket == FALSE => SetNextPhase(MasterHoldToken,PH);	MasterHoldToken
4	MasterHoldToken	/ PH == 11 => ACTicketND = TRUE; SetNextPhase(MasterHoldToken,PH);	MasterHoldToken
5	MasterHoldToken	/ PH == 12 => ACTicket = TRUE; SetNextPhase(MasterHoldToken,PH);	MasterHoldToken
6	MasterHoldToken	/ PH == 20 && ((ACTicketND == FALSE) && (ACTicket == FALSE)) => SetNextPhase(MasterHoldToken,PH);	MasterHoldToken

7	MasterHoldToken	/
		PH == 3 &&
		sents < Multiple Transmit
		=>
		sents = sents + 1;
		PduNum = QueLen();
		while (PduNum > 0)
		pdu = Deque();
		Send.req (port, ChannelGroup, pdu) to all ports;
		Enque(pdu);
		PduNum = PduNum -1
			MasterHoldToken

8	MasterHoldToken	/
		PH == 3 &&
		sents == Multiple Transmit
		=>
		QueDelete();
		pdu = CreateToken();
		Send.req (port, ChannelGroup, pdu) to all ports;
		Enque(pdu);
		sents = 1;
		PH = 3
			MasterHoldToken

#	Current state	Event/condition => action	Next state
9	MasterHoldToken	/ PH == 4 && sents < Multiple Token => pdu = Deque(); Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu); sents = sents + 1	MasterHoldToken
10	MasterHoldToken	/ PH == 4 && sents == Multiple Token => QueDelete(); PH = 0; TR10 = TRUE	MasterSolicitToken
11	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) to all ports other than rport; TR2=TRUE	MasterArbitration
12	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) to all ports other than rport; TR8=TRUE	MasterWaitTD
13	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (sport, DA, pdu) to all ports other than rport	MasterHoldToken
14	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterHoldToken
15	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => CStopSending(); ACStopSending(); LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU); TR10 = TRUE	MasterSolicitToken
16	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterHoldToken
17	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterHoldToken
18	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 =>	MasterHoldToken

#	Current state	Event/condition => action	Next state
		Send.req (sport, DA, pdu) to all ports other than rport	
19	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => CStopSending(); ACStopSending() Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu); TR10 = TRUE	MasterSolicitToken
20	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && ftype == 0x20 => CStopSending(); ACStopSending() Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu); TR10 = TRUE	MasterSolicitToken
21	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => CStopSending(); ACStopSending() Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu); TR10 = TRUE	MasterSolicitToken
22	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => CStopSending(); ACStopSending() ACReceived(pdu); TR10 = TRUE	MasterSolicitToken
23	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterHoldToken
24	MasterHoldToken	SendCyclic(pdu) => Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu)	MasterHoldToken
25	MasterHoldToken	SendACyclic(pdu) => Send.req (sport, ChannelGroup, pdu) to all ports; Enque(pdu)	MasterHoldToken

Table 104 – Master station state table – MasterMeasurement (without Transmission path delay measurement function)

#	Current state	Event/condition => action	Next state
1	MasterMeasurement	/ TRn == TRUE => TRn = FALSE	PrimaryMasterHoldToken

Table 105 – Master station state table – MasterMeasurement (with Transmission path delay measurement function)

#	Current state	Event/condition => action	Next state
1	MasterMeasurement	/ TR12 == TRUE => TR12 = FALSE MeasureTransmissionFlag = TRUE	MasterMeasurement
2	MasterMeasurement	SendMeasure(port, DA, pdu) => Send.req(port, DA, pdu)	MasterMeasurement
3	MasterMeasurement	SendUpdate(port, DA, pdu) => Send.req(port, DA, pdu)	PrimaryMasterHoldToken
4	MasterMeasurement	SendOffset(port, DA, pdu) => Send.req(port, DA, pdu)	MasterMeasurement
5	MasterMeasurement	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x41 => MeasureAckRecieved()	MasterMeasurement

Table 106 – Slave station state table – SlaveDown

#	Current state	Event/condition => action	Next state
1	SlaveDown	Power ON => ChannelGroup = NULL	SlaveWaitTD

Table 107 – Slave station state table – SlaveWaitTD

#	Current state	Event/condition => action	Next state
1	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD

2	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu)
		/
		DA==BROADCAST && SA !=MyAddr && ftype == 0x11
		=>
		pdu = CreateTestDataAck();
		Send.req(rport, SA, pdu);
		pdu = CreateTestData();
		Send.req (sport, DA, pdu) to all ports other than rport;
			SlaveWaitSetup

3	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
4	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
5	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD

#	Current state	Event/condition => action	Next state
6	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
7	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && && SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) || ftype == 0x20 || ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
8	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu)	SlaveWaitTD
9	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD

Table 108 – Slave station state table – SlaveWaitSetup

#	Current state	Event/condition => action	Next state
1	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) to all ports other than rport; ChannelGroup = NULL	SlaveWaitTD

2	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu)
		/
		DA==BROADCAST && SA !=MyAddr && ftype == 0x11
		=>
		pdu = CreateTestDataAck();
		Send.req(rport, SA, pdu);
		pdu = CreateTestData();
		Send.req (sport, DA, pdu) to all ports other than rport
			SlaveWaitSetup

3	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup
4	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU); TR3 = TRUE	SlaveSolicitToken
5	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup
6	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 =>	SlaveWaitSetup

#	Current state	Event/condition => action	Next state
		Send.req (sport, DA, pdu) to all ports other than rport	
7	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup
8	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && && SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) || ftype == 0x20 || ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; CRecieved(pdu)	SlaveWaitSetup
9	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu)	SlaveWaitSetup
10	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup

Table 109 – Slave station state table – SlaveSolicitToken (without Transmission path delay measurement)

#	Current state	Event/condition => action	Next state
1	SlaveSolicitToken	/ TR3 == TRUE => TR3 = FALSE; LeaveTimer startup	SlaveSolicitToken
2	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop; ChannelGroup = NULL	SlaveWaitTD
3	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
4	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
5	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU)	SlaveSolicitToken
6	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13	SlaveSolicitToken

#	Current state	Event/condition => action	Next state
		=> Send.req (sport, DA, pdu) to all ports other than rport	
7	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
8	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no); TR4 = TRUE	SlaveHoldToken
9	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== TRUE => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
10	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
11	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	SlaveSolicitToken
12	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && ftype ≥ 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu)	SlaveSolicitToken
13	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	SlaveSolicitToken
14	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu)	SlaveSolicitToken
15	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
16	SlaveSolicitToken	LeaveTimer time-out => LeaveTimer stop; ChannelGroup = NULL	SlaveWaitTD

Table 110 – Slave station state table – SlaveSolicitToken (with Transmission path delay measurement)

#	Current state	Event/condition => action	Next state
1	SlaveSolicitToken	/ TR3 == TRUE => TR3 = FALSE; LeaveTimer startup	SlaveSolicitToken
2	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop; ChannelGroup = NULL	SlaveWaitTD
3	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
4	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
5	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU)	SlaveSolicitToken
6	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
7	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
8	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no); TR4 = TRUE	SlaveHoldToken
9	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== TRUE => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
10	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken

#	Current state	Event/condition => action	Next state
11	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	SlaveSolicitToken
12	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && ftype ≥ 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu)	SlaveSolicitToken
13	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	SlaveSolicitToken
14	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => ACReceived(pdu)	SlaveSolicitToken
15	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
16	SlaveSolicitToken	LeaveTimer time-out => LeaveTimer stop; ChannelGroup = NULL	SlaveWaitTD
17	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x40 => MeasureRecieved(rport, DA, SA, pdu)	SlaveSolicitToken
18	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x41 => MeasureAckRecieved(rport, DA, SA, pdu)	SlaveSolicitToken
19	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x42 => OffsetRecieved(rport, DA, SA, pdu)	SlaveSolicitToken
20	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x43 => UpdateRecieved(rport, DA, SA, pdu)	SlaveSolicitToken

Table 111 – Slave station state table – SlaveHoldToken

#	Current state	Event/condition => action	Next state
1	SlaveHoldToken	/ TR4 == TRUE => TR4 = FALSE; SetNextPhase(SlaveHoldToken,0);	SlaveHoldToken

#	Current state	Event/condition => action	Next state

2	SlaveHoldToken	/
		PH == 1
		=>
		pdu = CreateMyStatus();
		Send.req(port,ChannelGroup,pdu) to all ports;
		CTicket = TRUE;
		sents = 0;
		TokenHopCounter = token-PDU.tokenHopCounter;
		TraAvailHopCounter = token-PDU.traAvailHopCounter;
		TraLastHopCounter = token-PDU.traLastHopCounter;
		TraAllows = token-PDU.traAllows;
		TraAllowsND = GetTraAllowsND();
		SetNextPhase(SlaveHoldToken,0);
			SlaveHoldToken

3	SlaveHoldToken	/ PH == 10 && CTicket == FALSE => SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
4	SlaveHoldToken	/ PH == 11 => ACTicketND = TRUE; SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
5	SlaveHoldToken	/ PH == 12 => ACTicket = TRUE; SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
6	SlaveHoldToken	/ PH == 20 && ((ACTicketND == FALSE) && (ACTicket == FALSE)) => SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken

7	SlaveHoldToken	/
		PH == 3 &&
		sents < Multiple Transmit
		=>
		sents = sents + 1;
		PduNum = QueLen();
		while (PduNum > 0)
		pdu = Deque();
		Send.req (port, ChannelGroup, pdu) to all ports;
		Enque(pdu);
		PduNum = PduNum -1
			SlaveHoldToken

8	SlaveHoldToken	/
		PH == 3 &&
		sents == Multiple Transmit
		=>
		QueDelete();
		pdu = CreateToken();
		Send.req (port, ChannelGroup, pdu) to all ports;
		Enque(pdu);
		sents = 1;
		PH = 3
			SlaveHoldToken

9	SlaveHoldToken	/ PH == 4 && sents == Multiple Token => pdu = Deque(); Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu); sents = sents + 1	SlaveHoldToken
10	SlaveHoldToken	/ PH == 4 && sents == Multiple Token => QueDelete(); PH = 0; TR3 = TRUE	SlaveSolicitToken
11	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10	SlaveWaitTD

#	Current state	Event/condition => action	Next state
		=> Send.req (sport, DA, pdu) to all ports other than rport; CStopSending(); ACStopSending(); ChannelGroup = NULL	
12	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
13	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
14	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => CStopSending(); ACStopSending(); ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU); TR3 = TRUE	SlaveSolicitToken
15	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
16	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
17	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
18	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
19	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu); LeaveTimer startup	SlaveSolicitToken
20	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && ftype == 0x20 => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu); LeaveTimer startup	SlaveSolicitToken

#	Current state	Event/condition => action	Next state
21	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype == 0x28 || ftype == 0x1C) => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu); LeaveTimer startup	SlaveSolicitToken
22	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 || ftype == 0x25 || ftype == 0x29 || ftype == 0x2E) => CStopSending(); ACStopSending(); ACReceived(pdu); LeaveTimer startup	SlaveSolicitToken
23	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
24	SlaveHoldToken	SendCyclic(pdu) => Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu)	SlaveHoldToken
25	SlaveHoldToken	SendACyclic(pdu) => Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu)	SlaveHoldToken

8.2.4.2 Functions

The Channel control functions are shown in Table 112 and Table 113 for Master and Slave stations respectively.

Table 112 – Master station channel control functions

Name	Description
getNextPhase(current_state, current_phase)	Sets up the next internal phase (represented by the variable PH).
CmpPriority(mst_pri, SA)	Compares the priority levels and returns Low or High. Low : mst_pri < MyPriority || (mst_pri == MyPriority && SA < MyAddr) High : mst_pri > MyPriority || (mst_pri == MyPriority && SA > MyAddr)
CreateTokenRoute()	Generates the token passing path, and sets SetupNodes.
MCAST(mac_addr)	Sets the I/G bit of addr to 1, and returns the address converted to a multicast address.
SetInvitationFlag()	Sets the flag indicating whether or not the node is to return to the system. The available values are as follows: ON: Return to system OFF: Do not return to system NULL: Undefined
SetMeasureFlag()	Sets the flag indicating whether or not the transmission path delay is to be measured. The available values are as follows:

Name	Description
	ON: Measure transmission path delay OFF: Do not measure transmission path delay NULL: Undefined
SetNewTokenFlag()	Sets a flag indicating whether or not to regenerate a token. The available values are as follows: ON: Regenerate OFF: Do not regenerate
IsValidPort(port)	Assesses whether the port is valid, and returns TRUE if it is and FALSE if it is not.
IsSlavePortInvalidated()	Assesses whether or not a slave station with an invalid port exists, and returns TRUE if it does, and FALSE if it does not.
SlavePortStateChanged ()	Returns TRUE if the port status of the slave station changed from link disconnected to link up, and FALSE if it did not change from link disconnected to link up.
LostTokenDst()	Finds the destination of the lost token.
AddReceivedToken(tseq _no)	Holds the sequence number of the received token.
IsReceivedToken(tseq_ no)	Assesses whether tseq_no is included in the token sequence numbers held by AddReceivedToken, and returns TRUE if it is and FALSE if it is not.
IsMulticast(addr)	Returns TRUE if the address if a multicast address, and FALSE if the address is not a multicast address.
CreatePersuasion()	Creates and returns the persuasion-PDU.
CreateTestData()	Creates and returns the testData-PDU.
CreateTestDataAck()	Creates and returns the testDataAck-PDU.
CreateSetup()	Creates and returns the setup-PDU.
CreateSetupAck()	Creates and returns the setupAck-PDU.
CreateToken()	Creates and returns the token-PDU.
CReceived(pdu)	Delivers pdu to the Cyclic Transmission state machine.
ACReceived(pdu)	Delivers pdu to the Acyclic Transmission state machine.
Enque(pdus)	Queues pdus given in arguments. Queues in the seqNumber order of pdu when multiple pdus are given.
Deque()	Retrieves pdus queued by Enque from the start of the queue.
QueDelete()	Deletes all pdus from the queue.
QueLen()	Returns the size of the queue.
SetTrInfo(pdu)	Holds the information included in the setup-PDU received as pdu.
MyStatusSend(pdu)	Sends MyStatus frame transmission notification
GetTraAllowsND()	Returns a limit value for node unt transient transmissions.

Table 113 – Slave station channel control functions

Name	Description
SetTrInfo(pdu)	Holds the information included in the setup-PDU received as pdu.
AddReceivedToken(tseq _no)	Holds the sequence number of the received token.
IsReceivedToken(tseq_ no)	Assesses whether or not the tseq_no is included in the token sequence numbers held by AddReceivedToken, and returns TRUE if it is and FALSE if it is not.
CreatePersuasion()	Creates and returns the persuasion-PDU.
CreateTestDataAck()	Creates and returns the testDataAck-PDU.

Name	Description
CreateSetupAck()	Creates and returns the setupAck-PDU.
CreateToken()	Creates and returns the token-PDU.
CReceived(pdu)	Delivers pdu to the Cyclic Transmission state machine.
ACReceived(pdu)	Delivers pdu to the Acyclic Transmission state machine.
GetTraAllowsND()	Returns a limit value for node unt transient transmissions.

8.2.4.3 Variables

The variables used in Channel control are shown in Table 114 and Table 115 for Master and Slave stations respectively.

Table 114 – Master station channel control variables

Name	Description
ChannelGroup	Multicast address indicating affiliated network
Nodes	Indicates the number of nodes subject to transmission control. (Used as the counter for the number of responses of TestDataAck.)
ANodes	Indicates the number of responses of TestDataAck upon return to the system.
SetupNodes	Indicates the number of Setup transmission destinations.
SARcvd	Indicates the number of responses of SetupAck.
PMChangeFlag	Flag indicating whether or not the transmission control manager is to be changed
InvitationFlag	Flag indicating whether or not the node is to return to the system
NewTokenFlag	Flag indicating whether or not a new token is to be generated upon token loss
MyPriority	Priority level of own node
MyAddr	MAC address of own node
BROADCAST	Broadcast MAC address FF-FF-FF-FF-FF-FF
CTRProgress	Generation status of token passing path
TraAllows	Number of times transient transmission is allowed
TraAvailHopCounter	Minimum value of number of token-PDU transfers in which transient transmission is allowed
TraLastHopCounter	Number of node token-PDU transfers in which the last transient transmission was performed
RingCheck	Flag indicating whether or not a ring check is to be performed
SynchronizationMaster	Flag indicating the synchronization manager.
PH	Indicates the subphase.

Table 115 – Slave station channel control variables

Name	Description
ChannelGroup	Multicast address indicating affiliated network
MyAddr	MAC address of own node
BROADCAST	Broadcast MAC address FF-FF-FF-FF-FF-FF
PrevTSeqNo	Received token sequence number
PH	Indicates the subphase.

8.2.4.4 Timers

The timers used in Channel control are shown in Table 116 and Table 117 for Master and Slave stations respectively.

Table 116 – Master station channel control timers

Name	Description
ListenTimer	Determines the time for monitoring the presence of communication at startup.
TDAckTimer	Determines the TestDataAck response wait time.
SAckTimer	Determines the SetupAck response wait time.
RvLastArbTimer	Determines the Persuasion reception wait time.
SendArbTimer	Determines the Persuasion send interval.
NetWatchTimer	Determines the time for monitoring whether or not communication is normal.
TDWaitTimer	Determines the TestData reception wait time.
SWaitTimer	Determines the Setup reception wait time.
LeaveTimer	Determines the time of disconnection detection.

Table 117 – Slave station channel control timers

Name	Description
LeaveTimer	Used for detecting disconnection.
MasterWatchTimer	Used for monitoring the master station.

8.2.5 Parameter dist

8.2.5.1 Paramter dist state machine

The states of the Parameter dist state machine for the Mater station are described in Table 118 with the details specified in Table 120.

The states of the Parameter dist state machine for the Slave station are described Table 119 with the details specified in Table 121.

Table 118 – Master station parameter dist states

Name	Description
IDLE	Parameters not distributed
InitParamChecking	Initial check of parameter distribution status in progress
ParamSending	Parameter distribution in progress
ParamChecking	Parameter distribution status check in progress
ParamSent	Parameter distribution completed

Table 119 – Slave station parameter dist states

Name	Description
ParamNone	No parameters
ParamCheck	Parameter check in progress
ParamHold	Holding parameters; parameters not reflected
ParamOK	Holding parameters; parameters reflected

Name	Description
ParamErr	Holding parameters; parameter error

Table 120 – Master station parameter dist state table

#	Current state	Event/condition => action	Next state
1	IDLE	ParameterDistReady() => AC Send.req(ParameterCheck, data)	InitParamChecking
2	InitParamChecking	MyStatusReceived(pdu) / MyStatusFromAllNodes() == FALSE =>	InitParamChecking
3	InitParamChecking	MyStatusReceived(pdu) / MyStatusFromAllNodes() == TRUE && ParamAllNodesReceived() == FALSE => GetNonReceivedNodes(); AC Send.req(netno, nodeno, Parameter, data)	ParamSending
4	InitParamChecking	MyStatusReceived(pdu) / MyStatusFromAllNodes() == TRUE && ParamAllNodesReceived() == TRUE => AC Send.req(ParameterCheck, data)	ParamChecking
5	ParamSending	MyStatusReceived(pdu) / ParamAllNodesReceived() == FALSE =>	ParamSending
6	ParamSending	MyStatusReceived(pdu) / MyStatusFromAllNodes() == TRUE => AC Send.req(ParameterCheck, data)	ParamChecking
7	ParamChecking	MyStatusReceived(pdu) / ParamAllNodesChecked() == TRUE => AC Send.req(ParameterCheck, data)	ParamSent
8	ParamChecking	MyStatusReceived(pdu) / ParamAllNodesChecked() == FALSE =>	ParamChecking
9	ParamSent	MyStatusReceived(pdu) / ParamAllNodesChecked() == FALSE => GetNonCheckedNodes(); AC Send.req(netno, nodeno, Parameter, data)	ParamSending

Table 121 – Slave station parameter dist state table

#	Current state	Event/condition => action	Next state
1	ParamNone	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck =>	ParamNone
2	ParamNone	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter => DoParameterCheck(pdu)	ParamCheck

#	Current state	Event/condition => action	Next state
3	ParamCheck	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter =>	ParamCheck
4	ParamCheck	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE =>	ParamCheck
5	ParamCheck	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
6	ParamCheck	ParameterCheckComplete(Result) / Result == TRUE =>	ParamHold
7	ParamCheck	ParameterCheckComplete(Result) / Result == FALSE =>	ParamErr
8	ParamErr	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter => DoParameterCheck(pdu)	ParamCheck
9	ParamErr	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE =>	ParamErr
10	ParamErr	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
11	ParamHold	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter =>	ParamCheck
12	ParamHold	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE => UpdateParameter(pdu)	ParamOK
13	ParamHold	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
14	ParamOK	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter =>	ParamCheck
15	ParamOK	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE => UpdateParameter(pdu)	ParamOK
16	ParamOK	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
17	ParamOK	MasterWatchTimer time-out =>	ParamHold

8.2.5.2 Functions

The Parameter dist functions are shown in Table 122 and Table 123 for Master and Slave stations respectively.

Table 122 – Master station parameter dist functions

Name	Description
ParameterDistReady	Ready for parameter distribution
ParamAllNodesReeived()	Assesses whether or not all nodes have received parameters. Returns TRUE if so and FALSE if not.
ParamAllNodesChecked()	Assesses whether or not all nodes have checked parameters. Returns TRUE if so and FALSE if not.
MyStatusFromAllNodes()	Assesses whether or not MyStatus has been received from all nodes. Returns TRUE if so and FALSE if not.
GetNonReceivedNodes()	Gets the network numbers and node numbers of nodes that have not received parameters.
GetNonCheckedNodes()	Gets the network numbers and node numbers of the nodes that have not checked parameters.
GetNetNo(pdu)	Gets the network number from pdu.
GetNodeNo(pdu)	Gets the node number from pdu.

Table 123 – Slave station parameter dist functions

Name	Description
DoParameterCheck(pdu)	Checks the parameters given by pdu. Calls ParameterCheckComplete(result) once completed. Sets result to TRUE if parameters are normal and to FALSE if parameters are abnormal.
CmpParamID(pdu)	Compares the parameter ID of the parameter given by pdu with the parameter ID of the parameter given by pdu in DoParameterCheck. Returns TRUE if they match and FALSE if they do not match.
UpdateParameter(pdu)	Sets the parameters given in pdu as the new parameters.

8.2.6 Synchronous trigger

8.2.6.1 Primitive definition

Synchronous trigger issues Synchronous trigger internal Internal.ind service to FSPM.

8.2.6.2 Synchronous trigger state machine

The states of the Synchronous trigger state machine for the Mater station are described in Table 124 with the details specified in Table 126.

The states of the Synchronous trigger state machine for the Slave station are described Table 125 with the details specified in Table 127.

Table 124 – Master station synchronous trigger states

Name	Description
ACTIVE	Operating state

Table 125 – Slave station synchronous trigger states

Name	Description
ACTIVE	Operating state

Table 126 – Master station synchronous trigger state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	MyStatusSend (pdu) / (SynchronizationMaster == TRUE) && (getMyStatusSyncFlag (pdu) == TRUE) => Synchronous Trigger Internal.ind	ACTIVE
2	ACTIVE	MyStatusReceived(pdu) / (SynchronizationMaster == FALSE) && getMyStatusSyncFlag () == TRUE => Synchronous Trigger Internal.ind	ACTIVE

Table 127 – Slave station synchronous trigger state table

#	Current state	Event/condition => action	Next state
1	ACTIVE	MyStatusReceived(pdu) / getMyStatusSyncFlag () == TRUE => Synchronous Trigger Internal.ind	ACTIVE

8.2.6.3 Functions

Functions enabled in Synchronous trigger are shown in Table 128.

Table 128 – Synchronous trigger functions

Name	Description
getMyStatusSyncFlag	Acquires the value of SyncFlag of the MyStatus frame.

8.2.6.4 Variables

Synchronous trigger uses no variables.

8.2.7 Timer

8.2.7.1 Timer state machine

The Timer state machine states are described in Table 129 and Table 130 with the detail is specified in Table 131 and Table 132.

Table 129 – Timer states – Best effort type

Name	Description
ACTIVE	Operating state

Table 130 – Timer states – Fixed cycle type

Name	Description
ACTIVE	Operating state
WAIT	Waiting state

Table 131 – Timer state table – Best effort type

#	Current state	Event/condition => action	Next state
1	ACTIVE	/ => MyStatusSendTimingFlag = ON	ACTIVE

Table 132 – Timer state table – Fixed cycle type

#	Current state	Event/condition => action	Next state
1	WAIT	/ MyStatusSendTimingFlag == OFF => ScanTimer startup	ACTIVE
2	ACTIVE	ScanTimer time-out => MyStatusSendTimingFlag = ON	WAIT

8.2.7.2 Functions

The Timer uses no functions.

8.2.7.3 Variables

The variables used by the Timer are shown in Table 133.

Table 133 – Timer variables

Name	Description
MyStatusSendTimingFlag	Sets the communication start flag. ON; Start communication OFF: Wait

8.2.7.4 Timers

The timers used by the fixed cycle type timer are shown in Table 134.

Table 134 – Fixed cycle timer

Name	Description
ScanTimer	Determines the communication cycle time.

8.2.8 Measure transmission

8.2.8.1 Primitive definition

FSPM issues a StartMeasureInternal.req service to MeasureTransmission. MeasureTransmission issues a StartMeasureInternal.cnf service to FSPM.

8.2.8.2 Measure transmission state machine

The states of the Measure transmission state machine for the Mater station are described in Table 135 with the details specified in Table 137.

The states of the Measure transmission state machine for the Slave station are described Table 136 with the details specified in Table 138.

Table 135 – Master station measure transmission states

Name	Description
M_IDLE	Initial state
SM_MEASURE	Measuring as synchronization manager
SM_WAIT_MEASUREACK	Waiting for MeasureAck as synchronization manager
M_WAIT_MEASUREACK	Measuring
M_WAIT_UPDATE	Waiting for measurement result update

Table 136 – Slave station measure transmission states

Name	Description
S_IDLE	Waiting
S_WAIT_MEASUREACK	Measuring
S_WAIT_UPDATE	Waiting for measurement result update

Table 137 – Master station measure transmission state table

#	Current state	Event/condition => action	Next state
1	M_IDLE	Start Measure Internal.req / (SynchronizationMaster == TRUE) && (MeasureTransmission Flag == TRUE) => MeasureTransmission Flag == FALSE slaveCounterMeasureDone = 0 slaveCounterMeasureAll = getSlaveCounterMeasureAll()	SM_MEASURE

2	M_IDLE	MeasureRecieved(rport, DA, SA, pdu)
		/
		(SynchronizationMaster == FALSE) &&
		DA == MyAddr
		=>
		startRoundtripTimer();
		DA = SA;
		pdu = CreateMeasureAck();
		SendMeasureAck(rport, DA, pdu);
		roundtrip = stopRounttripTimer();
			M_WAIT_OFFSET

3	M_IDLE	MeasureRecieved(rport, DA, SA, pdu) / (SynchronizationMaster == FALSE) && (DA != MyAddr) => SendMeasureAck(sport, DA, pdu) to all ports other than rport	M_WAIT_MEASUREACK
4	M_IDLE	Get Offset Internal.req / SynchronizationMaster == FALSE => Get Offset Internal.cnf(offset)	M_IDLE
5	SM_MEASURE	/ slaveCounterMeasureDone < slaveCounterMeasureAll => startRoundTripTimer();	SM_WAIT_MEASURE_ACK

#	Current state	Event/condition => action	Next state

		DA = getTerminatedSlaveAddress(slaveCounterMeasureDone);
		port = getTernimatedSlaveDirection(slaveCounterMeasureDone);
		pdu = CreateMeasure();
		SendMeasure(port, DA, pdu)	

6	SM_MEASURE	/ slaveCounterMeasureDone == slaveCounterMeasureAll => DA = ChannelGroup; pdu = CreateUpdate(); SendUpdate(sport, DA, pdu) to all ports Start Measure Internal.cnf(TRUE)	M_IDLE

7	SM_WAIT_MEASUREACK	MeasureAckRecieved()
		=>
		roundtrop = stopRoundTripTimer();
		offset = getOffset(roundtrip);
		DA = getTerminatedSlaveAddress(SlaveCounterMeasureDone);
		sport = getTernimatedSlaveDirection(SlaveCounterMeasureDone);
		pdu = CreateOffset(offset);
		SendOffset(sport, DA, pdu);
		slaveCounterMeasureDone++
			SM_MEASURE

8	SM_WAIT_MEASUREACK	IsRoundtripTimerOver() => Start Measure Internal.cnf(FALSE)	M_IDLE
9	M_WAIT_MEASUREACK	MeasureAckRecieved(rport, DA, SA, pdu) => roudtrip = stopRoundtripTimer(); SendMeasureAck(sport, DA, pdu); to all ports other than rport startArrivalOffsetTimer()	M_WAIT_OFFST
10	M_WAIT_MEASUREACK	IsRoundtripTimerOver()	M_IDLE
11	M_WAIT_OFFSET	OffsetRecieved(rport, DA, SA, pdu) => roudtrip = stopRoundtripTimer(); SendOffset(sport, DA, pdu) to all ports other than rport startArrivalOffsetTimer()	M_WAIT_UPDATE
12	M_WAIT_OFFSET	IsArrivalOffsetTimerOver()	M_IDLE
13	M_WAIT_UPDATE	UpdateRecieved(rport, DA, SA, pdu) => SendUpdate(sport, DA, pdu) to all ports other than rport offset = generateOffset(roundtrip, distributeOffset)	M_IDLE
14	M_WAIT_UPDATE	IsArrivalUpdateTimerOver()	M_IDLE

Table 138 – Slave station measure transmission state table

#	Current state	Event/condition => action	Next state

1	S_IDLE	MeasureRecieved(rport, DA, SA, pdu)
		/
		DA == MyAddr
		=>
		startRoundtripTimer();
		DA = SA;
		pdu = CreateMeasureAck();
		SendMeasureAck(rport, DA, pdu);
		roundtrip = stopRounttripTimer();
			S_WAIT_OFFSET

2	S_IDLE	MeasureRecieved(rport, DA, SA, pdu) / DA != MyAddr =>	S_WAIT_MEASUREACK

#	Current state	Event/condition => action	Next state
		SendMeasure(sport, DA, pdu) to all ports other than rport startRoundtripTimer();	
3	S_IDLE	Get Offset Internal.req => Get Offset Internal.cnf(offset)	S_IDLE
4	S_WAIT_MEASUREACK	MeasureAckRecieved(rport, DA, SA, pdu) => roudtrip = stopRoundtripTimer(); SendMeasureAck(sport, DA, pdu) to all ports other than rport startArrivalOffsetTimer()	S_WAIT_OFFST
5	S_WAIT_MEASUREACK	IsRoundtripTimerOver()	S_IDLE
6	S_WAIT_OFFSET	OffsetRecieved(rport, DA, SA, pdu) => roudtrip = stopRoundtripTimer(); SendOffset(sport, DA, pdu) to all ports other than rport startArrivalOffsetTimer()	S_WAIT_UPDATE
7	S_WAIT_OFFSET	IsArrivalOffsetTimerOver()	S_IDLE
8	S_WAIT_UPDATE	UpdateRecieved(rport, DA, SA, pdu) => SendUpdate(sport, DA, pdu) to all ports other than rport offset = generateOffset(roundtrip, distributeOffset)	S_IDLE
9	S_WAIT_UPDATE	IsArrivalUpdateTimerOver() =>	S_IDLE

8.2.8.3 Functions

Functions used by the Measure transmission are shown in Table 139 and Table 140 for the Master and the Slave stations respectively.

Table 139 – Master station measure transmission functions

Name	Description
getSlaveCounterMeasureAll()	Acquires the number of slave stations that perform delay measurement.
getTerminatedSlaveAddress(s)	Acquires the MAC address of the terminating slave station.
getTerminatedSlaveDirection(s)	Acquires the port connected to the terminating slave station.
getOffset(roundtrip)	Acquires the offset value from roundtrip.
CreateMeasure()	Creates and returns the measure-PDU.
CreateMeasureAck()	Creates and returns the measureAck-PDU.
CreateOffset(offset)	Creates and returns the offset-PDU that reflects the offset value.
CreateUpdate()	Creates and returns the update-PDU.
SendMeasure(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
SendMeasureAck(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
SendOffset(port, DA, pdu)	Sends port, DA, SA, and pdu to the Channel Control state machine.
SendUpdate(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
startRoundtripTimer()	Starts the RoundTripTimer.
stopRoudtripTimer()	Stops the RoundTripTimer, and returns the RoundTripTimer value.
IsRoundtripTimerOver()	Returns TRUE when the RoundTripTimer times out and FALSE in all other cases.
startArrivalOffsetTimer()	Starts the ArrivalOffsetTimer.

Name	Description
stopArrivalOffsetTimer()	Stops the ArrivalOffsetTimer.
IsArrivalOffsetTimerOver()	Returns TRUE when theArrivalOffsetTimer times out and FALSE in all other cases.
startArrivalUpdateTimer()	Starts the ArrivalUpdateTimer.
stopArrivalUpdateTimer()	Stops the ArrivalUpdateTimer.
IsArrivalUpdateTimerOver()	Returns TRUE when the ArrivalUpdateTimer times out and FALSE in all other cases.

Table 140 – Slave station measure transmission functions

Name	Description
CreateMeasure()	Creates and returns the measure-PDU.
CreateMeasureAck()	Creates and returns the measureAck-PDU.
CreateUpdate()	Creates and returns the update-PDU.
SendMeasure(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
SendMeasureAck(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
SendUpdate(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
startRoundtripTimer()	Starts the RoundTripTimer.
stopRoudtripTimer()	Stops the RoundTripTimer, and returns the RoundTripTimer value.
IsRoundtripTimerOver()	Returns TRUE when the RoundTripTimer times out, and FALSE in all other cases.
startArrivalOffsetTimer()	Starts the ArrivalOffsetTimer.
stopArrivalOffsetTimer()	Stops the ArrivalOffsetTimer.
IsArrivalOffsetTimerOver()	Returns TRUE when the ArrivalOffsetTimer times out, and FALSE in all other cases.
startArrivalUpdateTimer()	Starts the ArrivalUpdateTimer.
stopArrivalUpdateTimer()	Stops the ArrivalUpdateTimer.
IsArrivalUpdateTimerOver()	Returns TRUE when the ArrivalUpdateTimer times out, and FALSE in all other cases.

8.2.8.4 Variables

Measure transmission variables are shown in Table 141.

Table 141 – Master station measure transmission valiables

Name	Description
slaveCounterMeasureDone	Indicates the number of slave stations for which delay measurement is completed.
SlaveCounterMeasureAll	Indicates the total number of slave stations for which delay measurement is performed.
roundtrip	Indicates the round-trip time.
offset	Indicates the offset value.
roundTrip	Indicates the round-trip time.
offset	Indicates the offset value.

9 DLL mapping protocol machine (DMPM)

9.1 DMPM type C

The structure of the DMPM is shown in Figure 22. The DMPM maps the DMPM service to the data link layer service (DL service). DMPM uses services provided by the MAC sub layer of the data link layer as the DL services. The mapping of DMPM and DLL is shown in Table 142 and the PDU MAC destination address mapping is shown in Table 143.

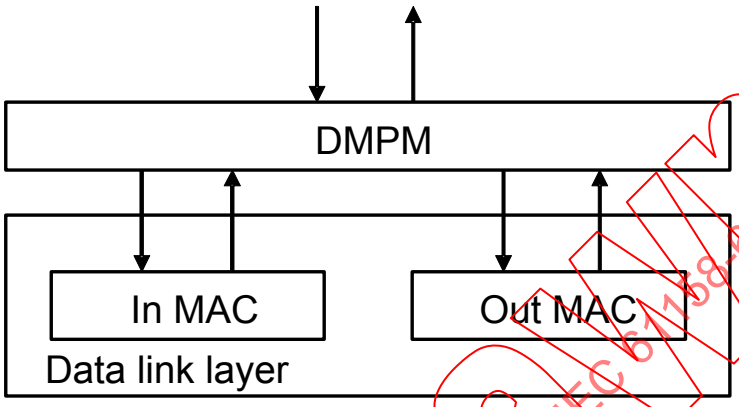


Figure 22 – Structure of type C DMPM

Table 142 – Mapping of type C DMPM service and DL service

DMPM service	DL service	Remarks
InPort.req(PDU)	MA_DATA.req(destination_address,source_address, frame_check_sequence) in In MAC.	Refer to Table 143 for destination_address. source_address is the MAC address of a node.
InPort.ind(PDU)	MA_DATA.ind(destination_address,source_address,PDU, frame_check_sequence) in In MAC.	Refer to Table 143 for destination_address. source_address is the MAC address of a source.
InPort.ind(Port_State)	Shows the link status of In MAC.	
OutPort.req(PDU)	MA_DATA.req(destination_address,source_address, frame_check_sequence) in Out MAC.	Refer to Table 143 for destination_address. source_address is the MAC address of a node.
OutPort.ind(PDU)	MA_DATA.ind(destination_address,source_address, PDU,frame_check_sequence) in Out MAC.	Refer to Table 143 for destination_address. source_address is the MAC address of a source.
OutPort.ind(Port_State)	Shows the link status of Out MAC.	

Table 143 – Destination address for each type C PDU

Name of PDU	Destination address
Connect-PDU	FF-FF-FF-FF-FF-FF
ConnectAck-PDU	The MAC address of a node to which a response is given (source_address of corresponding Connect-PDU)
Scan-PDU	FF-FF-FF-FF-FF-FF
Collect-PDU	FF-FF-FF-FF-FF-FF
Select-PDU	FF-FF-FF-FF-FF-FF
Launch-PDU	FF-FF-FF-FF-FF-FF
Token-PDU	FF-FF-FF-FF-FF-FF

Name of PDU	Destination address
MyStatus-PDU	FF-FF-FF-FF-FF-FF
Transient1-PDU	dataType == 0:FF-FF-FF-FF-FF-FF
Transient2-PDU	A destination is within in the same network: The MAC address of a destination node A network whose destination address is different: The MAC address of a relaying node
NTNTest-PDU	MAC address of own node.
Dummy-PDU	MAC address of own node.
CyclicDataW-PDU	FF-FF-FF-FF-FF-FF
CyclicDataB-PDU	FF-FF-FF-FF-FF-FF
CycclicDataOut1-PDU	FF-FF-FF-FF-FF-FF
CyclicDataOut2-PDU	FF-FF-FF-FF-FF-FF
CyclicDataIn1-PDU	FF-FF-FF-FF-FF-FF
CyclicDataIn2-PDU	FF-FF-FF-FF-FF-FF

9.2 DMPM type F

The structure of the DMPM is shown in Figure 23. The DMPM maps the DMPM service to the data link layer service (DL service). DMPM uses services provided by the MAC sub layer of the data link layer as the DL services. The mapping of DMPM and DLL is shown in Table 144.

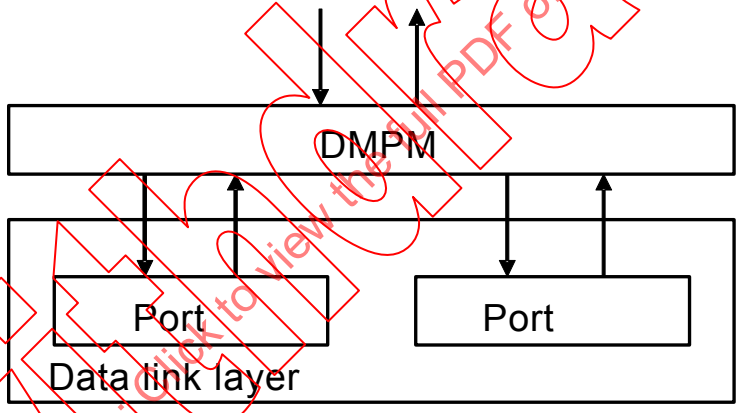


Figure 23 – Structure of type F DMPM

Table 144 – Mapping of type F DMPM service and DL service

DMPM service	DL service	Remarks
Send.req(port, DA, pdu)	MA_DATA.req (destination_address, source_address, mac_service_data_unit, frame_check_sequence) of port	destination_address = source_address is the MAC address of a node.
Receive.ind(port, DA, SA, ftype, pdu)	MA_DATA.ind (destination_address, source_address, mac_service_data_unit, frame_check_sequence) of port	DA = destination_address SA = source_address ftype = mac_service_data_unit. ArFType pdu = mac_service_data_unit

Bibliography

IEC 61784-1, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

IEC 61784-2, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

ISO/IEC 8822, *Information technology – Open Systems Interconnection – Presentation service definition*

_____

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-23: Spécification du protocole de la couche application – Éléments de type 23

AVANT-PROPOS

1) La Commission Électrotechnique 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. À 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 protocoles sont spécifiées dans la CEI 61784-1 et la CEI 61784-2.

La Norme internationale IEC 61158-6-23 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.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65C/764/FDIS	65C/774/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, publiée sous le titre général *Réseaux de communications industriels – Spécifications des bus de terrain*, est disponible 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. À cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

0 INTRODUCTION

0.1 Généralités

La présente partie de la CEI 61158 fait partie d'une série élaborée pour faciliter l'interconnexion des composants de systèmes d'automatisation. Elle renvoie aux autres normes de l'ensemble défini par le modèle de référence de bus de terrain "à trois couches" décrit dans la CEI 61158-1:2014.

Le protocole d'application fournit le service d'application au moyen des services disponibles au niveau de la couche Liaison de données ou de la couche immédiatement inférieure. Le principal objectif de la présente norme est de définir un ensemble de règles de communication, exprimées en termes de procédures que doivent suivre les entités d'application (Application Entity, AE) homologues au moment de la communication. Ces règles de communication visent à fournir une base saine pour le développement, dans divers buts:

– en tant que guide pour les développeurs et les concepteurs;
– dans une optique d'utilisation lors de l'essai et de l'achat de matériel;
– dans le cadre d'un accord pour l'admission de systèmes dans l'environnement de systèmes ouverts;
– en tant que précision apportée à la compréhension des communications en temps critique dans le modèle OSI.

Cette norme traite, en particulier, de la communication et de l'interfonctionnement des capteurs, effecteurs et autres appareils d'automatisation. L'utilisation conjointe de la présente norme avec d'autres normes entrant dans les modèles de référence OSI ou de bus de terrain permet à des systèmes qui ne pourraient pas, sans cela, fonctionner ensemble dans toute combinaison.

0.2 Mention des brevets

La Commission Electrotechnique Internationale (CEI) attire l'attention sur le fait qu'il est déclaré que la conformité avec les dispositions du présent document peut impliquer l'utilisation d'un brevet intéressant des éléments de type 23 et éventuellement d'autres types indiqués en 8.1 et 8.2 comme suit:

JP 05106658 US 7983177 DE 112006003943.1 KR 10-1029201 TW I338476	[MEC]	Communication management device, communication node, communication system, and data communication method
JP 4503678 DE 112006003895.8 KR 10-1024472 CN 201110218295.6 TW I333356	[MEC]	Communication management device, communication device, and communication method
JP 2010-045463 US 12/774377 DE 112006004225.4 KR 10-1024482 CN 201010148761.3 TW 099112461	[MEC]	Communication node, and token issuing method and token-ring communication method in ring communication system
JP 05127977	[MEC]	Synchronization system, time master nodes, time slave nodes and synchronization method

JP 05106658 US 13/334863 DE 112008004265.9 KR 10-2011-7030535 CN 201210026699.X TW 101108048	[MEC]	Communication management device, communication node, communication system, and data communication method
JP 2011-128274 US 13/325125 DE 112008004268.3 KR 10-2011-7029114 CN 201210127058.3 TW 101102132	[MEC]	Communication management device, communication device, and communication method
JP 05084916 US 13/142244 DE 112008004245.4 KR 10-2011-7014492 CN 200880132546.5 TW 098100145	[MEC]	Communication management device, communication device, and communication method
JP 2011-518195 US 13/377397 DE 112009004913.3 KR 10-2011-7027858 CN 200980159835.9 TW 098119663	[MEC]	Communication managing apparatus, communication nodes, and data communication method
JP 2011-532954	[MEC]	Network performance estimating apparatus, network performance estimating method, network structure recognizing method, communication managing apparatus, and data communication method

La CEI ne prend pas position quant à la preuve, à la validité et à la portée de ces droits de propriété.

Les détenteurs de ces droits de propriété ont donné l'assurance à la CEI qu'il consentent à négocier des licences avec des demandeurs du monde entier, soit sans frais soit à des termes et conditions raisonnables et non discriminatoires. À ce propos, la déclaration des détenteurs des droits de propriété est enregistrée à la CEI. Des informations peuvent être demandées à:

[MEC] Mitsubishi Electric Corporation
Corporate Licensing Division
7-3, Marunouchi 2-chome, Chiyoda-ku,
Tokyo 100-8310, Japon

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 6-23: Spécification du protocole de la couche application – Éléments de type 23

1 Domaine d'application

1.1 Généralités

La couche Application de bus de terrain (Fieldbus Application Layer (FAL)) procure aux programmes de l'utilisateur un moyen d'accès à l'environnement de communication des bus de terrain. À cet égard, la FAL peut être vue comme une «fenêtre entre des programmes d'application correspondants».

La présente norme donne les éléments communs visant à assurer les communications de messagerie de base en temps critique et en temps non critique entre des programmes d'application dans un environnement et des équipements d'automatisation spécifiques au bus de terrain de Type 23. Le terme "en temps critique" sert à représenter la présence d'une fenêtre temporelle, dans les limites de laquelle une ou plusieurs actions spécifiées sont tenues d'être parachevées avec un certain niveau défini de certitude. Le manquement à parachever les actions spécifiées dans les limites de la fenêtre temporelle risque d'entraîner la défaillance des applications qui demandent ces actions, avec le risque concomitant pour l'équipement, l'installation et éventuellement pour la vie humaine.

La présente norme définit de manière abstraite le comportement, visible par un observateur externe, assuré par les différents Types de la couche Application de bus de terrain, en termes

a) de syntaxe abstraite définissant les unités de données de protocole de couche Application, transmises entre les entités d'application en communication,

b) de syntaxe de transfert définissant les unités de données de protocole de couche Application, transmises entre les entités d'application en communication;

c) de diagramme d'états de contexte d'application définissant le comportement de service d'application observable entre les entités d'application en communication; et

d) de diagrammes d'états de relations entre applications définissant le comportement de communication visible entre les entités d'application en communication; et.

La présente norme vise à définir le protocole mis en place pour

a) définir la représentation câblée des primitives de service définies dans la CEI 61158-5-23, et

b) définir le comportement visible de l'extérieur associé à leur transfert.

La présente norme spécifie le protocole de la couche Application de bus de terrain de la CEI, en conformité avec le modèle de référence de base OSI (ISO/CEI 7498) et avec la structure de la couche Application OSI (ISO/CEI 9545).

Les services et protocoles de la FAL sont fournis par des entités d'application (application entity, AE) de la FAL contenues dans les processus d'application. L'AE de la FAL se compose d'un jeu d'Éléments de service application (ASE, Application Service Element) orientés objet et d'une Entité de gestion de couche (LME, Layer Management Entity) qui gère l'AE. Les ASE fournissent des services de communication qui fonctionnent sur un jeu de classes d'objets de processus d'application (APO, application process object) connexes. L'un des ASE de la FAL est un ASE de gestion qui fournit un jeu commun de services pour la gestion des instances de classes de la FAL.

Bien que ces services spécifient, du point de vue des applications, la manière dont la demande et les réponses sont émises et délivrées, ils n'incluent pas une spécification de ce que les applications qui demandent et qui répondent doivent en faire. À savoir, les aspects comportementaux des applications ne sont pas spécifiés; seule une définition des demandes et réponses qu'elles peuvent envoyer/recevoir est spécifiée. Cela permet une plus grande flexibilité aux utilisateurs de la FAL pour normaliser un tel comportement d'objet. En plus de ces services, certains services d'appui sont également définis dans la présente norme pour fournir l'accès à la FAL afin de maîtriser certains aspects de son fonctionnement.

1.2 Spécifications

L'objet principal de la présente norme est de spécifier la syntaxe et le comportement du protocole de couche application qui achemine les services de couche application définis dans la CEI 61158-5-23.

Un objectif secondaire est de fournir des trajets de migration à partir de protocoles de communications industrielles préexistants. Ce dernier objectif explique la diversité des protocoles normalisés dans les sous-parties de la CEI 61158-6.

1.3 Conformité

La présente norme ni ne spécifie de mises en œuvre individuelles ou de produits individuels ni ne contraint les mises en œuvre d'entités de couche application au sein des systèmes d'automatisation industriels.

Il n'existe pas de conformité de l'équipement à la norme de définition de service de couche Application. En revanche, la conformité est obtenue par le biais de la mise en œuvre de cette spécification de protocoles de couche Application.

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 61158-1:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 1: Présentation et lignes directrices des séries CEI 61158 et CEI 61784*

CEI 61158-5-23, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-23: Définition des services de la couche application – Éléments de type 23*

CEI 61158-6 (toutes les parties), *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6: Spécification du protocole de la couche application*

ISO/CEI 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Le modèle de base*

ISO/IEC 8824-1, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation* (disponible en anglais seulement)

ISO/CEI 9545, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Structure de la couche Application*

ISO/CEI 10731, *Technologies de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base – Conventions pour la définition des services OSI*

3 Termes, définitions, symboles, termes abrégés et conventions

Pour les besoins du présent document, les termes, définitions, symboles, abréviations et conventions suivants s'appliquent.

3.1 Termes et définitions référencés

3.1.1 Termes de l'ISO/CEI 7498-1

Pour les besoins du présent document, les termes suivants donnés dans l'ISO/CEI 7498-1 s'appliquent:

a) entité d'application
b) processus d'application
c) unité de données de protocole application
d) élément de service application
e) invocation d'entité d'application
f) invocation de processus d'application
g) transaction d'application
h) système ouvert réel
i) syntaxe de transfert

3.1.2 Termes de l'ISO/CEI 8822

Pour les besoins du présent document, les termes suivants donnés dans l'ISO/CEI 8822 s'appliquent:

a) syntaxe abstraite
b) contexte de présentation

3.1.3 Termes de l'ISO/CEI 9545

Pour les besoins du présent document, les termes suivants donnés dans l'ISO/CEI 9545 s'appliquent:

a) application-association (association d'applications)
b) application-context (contexte d'application)
c) application context name (nom de contexte d'application)
d) application-entity-invocation (invocation d'entité d'application)
e) application-entity-type (type d'entité d'application)
f) application-process-invocation (invocation de processus d'application)
g) application-process-type (type de processus d'application)
h) application-service-element (élément de service d'application)
i) application control service element (élément de service de contrôle d'application)

3.1.4 Termes de l'ISO/CEI 8824-1

Pour les besoins du présent document, les termes suivants donnés dans l'ISO/CEI 8824-1 s'appliquent:

a) identificateur d'objet

b) Type

3.1.5 Termes de la CEI 61158-1

Pour les besoins du présent document, les termes suivants donnés dans la CEI 61158-1 s'appliquent:

a) Machine de protocole de mapping de couche DL

b) couche application de bus de terrain

c) machine de protocole de service FAL

d) unité de données de protocole

3.2 Termes et définitions spécifiques au type 23

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

3.2.1
transmission cyclique
transmission qui est exécutée périodiquement et sert à mettre à jour l'appareil de liaison

3.2.2
station d'appareil intelligent
nœud capable d'effectuer la transmission cyclique et la transmission transitoire de données de bits 1:n et de données de mots avec la station maître, et la transmission transitoire avec les stations esclaves, à l'exclusion des stations d'E/S distantes et ayant des fonctions de client et des fonctions de serveur durant la transmission transitoire

3.2.3
Link Bit (Bit de liaison)
données de bit de relais de liaison, qui sont partagées par tous les nœuds au moyen de la transmission cyclique et qui sont utilisées en tant que mémoire partagée d'unité un bit du type n:n

3.2.4
appareil de liaison
bit de liaison, mot de liaison, liaison x et liaison y ou RX, RY, RWr et RWw

3.2.5
Link Word (Mot de liaison)
données d'unité deux octets de registre de liaison, qui sont partagées par tous les nœuds au moyen de la transmission cyclique et qui sont utilisées en tant que mémoire partagée d'unité deux octets du type n:n

3.2.6
liaison x
données de bit reçues sur une entrée de liaison, qui sont émises depuis chaque nœud au moyen de la transmission cyclique et qui sont utilisées en tant que mémoire partagée d'entrée du type 1:n

3.2.7
liaison y
données de bit de sortie de liaison, qui sont envoyées à chaque nœud au moyen de la transmission cyclique et qui sont utilisées en tant que mémoire partagée de sortie du type 1:n

3.2.8
station locale
nœud capable d'effectuer la transmission cyclique et la transmission transitoire de données de bits n:n et de données de mots avec la station maître et d'autres stations locales, et la

transmission transitoire avec les stations esclaves, à l'exclusion des stations d'E/S distantes et ayant des fonctions de serveur et des fonctions de client durant la transmission transitoire

3.2.9
nœud de gestion
nœud dans lequel des paramètres sont définis

3.2.10
station maître
nœud qui possède des informations de commande (paramètres) et qui gère la transmission cyclique

3.2.11
nœud
élément qui forme un réseau et qui effectue l'émission, la réception et le transfert de données

3.2.12
essai de nœud à nœud
essai de couche physique entre deux nœuds

3.2.13
nœud normal
nœud autre qu'un nœud de gestion

3.2.14
station à appareil distant
nœud capable d'effectuer la transmission cyclique et la transmission transitoire de données de bits 1:n et de données de mots avec la station maître, et la transmission transitoire avec les stations esclaves, à l'exclusion des stations d'E/S distantes et ayant des fonctions de serveur durant la transmission transitoire

3.2.15
station d'E/S déportées
nœud capable d'effectuer la transmission cyclique de données de bits 1:n avec la station maître

3.2.16
nœud de réserve
nœud qui n'est pas encore connecté, mais qui est compté dans le nombre de nœuds total du réseau; il n'effectue pas de transmission cyclique mais est toujours considéré comme normal par les applications

3.2.17
RX
entrée distante vue de la station maître avec des données de bits qui sont mises à jour périodiquement par transmission cyclique, de l'esclave vers le maître; ou dans une station locale vue de la station maître, il s'agit de la RY de la station locale

3.2.18
RY
sortie distante vue de la station maître avec des données de bits qui sont mises à jour périodiquement par transmission cyclique, du maître vers l'esclave; ou dans une station locale vue de la station maître, il s'agit de la RX de la station locale

3.2.19
RWr
registre distant (entrée) vu de la station maître avec des données de mots qui sont mises à jour périodiquement par transmission cyclique, de l'esclave vers le maître; ou dans une station locale vue de la station maître, il s'agit du RWw de la station locale

3.2.20
RWw
registre distant (sortie) vu de la station maître avec des données de mots qui sont mises à jour périodiquement par transmission cyclique, du maître vers l'esclave; ou dans une station locale vue de la station maître, il s'agit du RWw de la station locale

3.2.21
station esclave
nœud autre que la station maître

3.2.22
station
nœud

3.2.23
Synchronization manager (Gestionnaire de synchronisation)
nœud (rôle de station maître, à raison d'une par réseau) qui gère la synchronisation, en distribuant la synchronisation temporelle aux autres nœuds

3.2.24
transmission transitoire
transmission qui est exécutée lors de chaque demande

3.2.25
fonction de client de transmission transitoire
fonction qui émet une demande transitoire

3.2.26
fonction de serveur de transmission transitoire
fonction qui reçoit une demande transitoire et émet une réponse

3.2.27
gestionnaire de commande de transmission
nœud (rôle de station maître, à raison d'une par réseau) qui effectue la gestion du passage de jetons

3.2.28
mot (WORD)
unité représentant des données, de longueur 16 bits

3.3 Symboles et termes abrégés

AE	Application Entity (Entité d'application)
AL	Application Layer (Couche application)
AP	processus d'application
APDU	Application Protocol Data Unit (Unité de données de protocole d'application)
APO	Application Process Object (Objet de processus d'application)
AR	Application Relationship (Relation d'applications)
AREP	Application Relationship Endpoint (Point d'extrémité

	de relation d'applications)
ASE	Application Service Element (Élément de service d'application)
ASN.1	Abstract Syntax Notation 1 (Notation de syntaxe abstraite n° 1)
CRC	Contrôle de redondance cyclique
DLL	Data-link Layer (Couche liaison de données)
DMPM	DLL Mapping Protocol Machine (Machine de protocole de mapping DLL)
FAL	Fieldbus application layer (Couche application de bus de terrain)
FSPM	FAL Service Protocol Machine (Machine de protocole de service FAL)
LB	Link Bit (Bit de liaison)
LSB	Least Significant Bit (Bit de poids faible)
LW	Link Word (Mot de liaison)
LX	Liaison X
LY	Liaison Y
MSB	Most Significant Bit (Bit de poids fort)
OSI	Open Systems Interconnection (interconnexion des systèmes ouverts)
PDU	Protocol Data Unit (Unité de données de protocole)

3.4 Conventions

3.4.1 Concept général

La FAL se compose d'un ensemble d'ASE orientés objet. Chaque ASE est spécifié dans un paragraphe distinct. Chaque spécification d'ASE est constituée de trois parties, à savoir ses définitions de classe, ses services et sa spécification de protocole. Les deux premières sont contenues dans la CEI 61158-5-23. La spécification de protocole pour chaque élément ASE est définie dans la présente norme.

Les définitions de classe définissent les attributs des classes prises en charge par chaque élément ASE. Les attributs sont accessibles à partir d'instances de la classe en utilisant les services d'ASE de gestion ("Management") spécifiés dans la CEI 61158-5-23. La spécification de service définit les services fournis par l'ASE.

La présente norme utilise les conventions de description données dans l'ISO/CEI 10731.

3.4.2 Conventions d'encodage des bits et octets réservés

Il est admis d'utiliser le terme "Reserved" (réservé) pour décrire des bits compris dans des octets ou des octets entiers. Il convient que tous les bits ou octets réservés soient mis à zéro du côté émetteur et ils ne doivent pas être soumis à essai du côté récepteur, sauf si cela est explicitement énoncé ou si les bits ou octets réservés sont vérifiés par un diagramme d'états.

Il est également permis d'utiliser le terme "Reserved" pour indiquer que certaines valeurs, comprises dans la plage d'un paramètre, sont réservées en vue d'extensions futures. Dans ce cas, il convient de ne pas utiliser les valeurs réservées du côté émetteur et elles ne doivent pas être soumises en essai du côté récepteur, sauf si cela est explicitement énoncé ou si les valeurs réservées sont vérifiées par un diagramme d'états.

3.4.3 Conventions pour la description de la syntaxe abstraite

Cette description de la FAL de Type 23utilise un sous-ensemble de l'ASN.1, conformément à l'ISO/CEI 8824-1. Les structures suivantes sont utilisées.

Type sélectif (CHOICE) – Représente une sélection parmi les types de candidats

Type de séquence (SEQUENCE) – Représente une liste à ordre fixe

comme dans l'exemple suivant:

```
DLPDU::= SEQUENCE {
    preamble        Preamble,
    sfd             SFD,
    destaddr        DestAddr,
    srcaddr         SrcAddr,
    lt              LT,
    DLSDU           FAL-PDU,
    FCS             FCS
}
```

NOTE Cet exemple indique que la DLPDU, qui représente la trame Ethernet, est définie comme une SÉQUENCE. La DLPDU comprend ce qui suit: Preamble, SFD, DestAddr, SrcAddr, LT, FAL-PDU et FCS.

3.4.4 Conventions pour la description de bit en octets

Lors de l'identification de chaque bit d'un octet, chaque bit est identifié par un numéro, tel qu'indiqué à la Figure 1 et décrit comme le Bit n.

MSB							LSB	
7	6	5	4	3	2	1	0	Bit identification number

Anglais	Français
MSB	MSB (Bit de poids fort)
LSB	LSB (Bit de poids faible)
Bit identification number	Numéro d'identification de bit

Figure 1 – Description de bit en octets

Lors de la spécification de plusieurs bits situés de façon séquentielle, le symbole de plage (..) est utilisé (par ex.: 7..0 spécifie les bits 7 à 0 inclusifs).

Lors de la spécification de plusieurs octets, le bit de poids faible (LSB) de l'octet le plus bas est considéré comme étant le Bit 0, puis les numéros d'identification de bit sont attribués par ordre croissant.

NOTE Par exemple, lors de la spécification de 4 octets, le bit de poids fort (MSB) de l'octet le plus haut est le Bit 31, le MSB du deuxième octet est le Bit 23, le MSB du troisième octet est le Bit 15 et le MSB de l'octet le plus bas est le Bit 7.

3.4.5 Conventions relatives aux descriptions des diagrammes d'états

La description du diagramme d'états est définie sous forme de tableau, tel qu'indiqué dans le Tableau 1. La signification des éléments est indiquée dans le Tableau 2. Les conventions utilisées dans les diagrammes d'états sont présentées dans le Tableau 3.

Chaque ligne du tableau d'états représente une transition d'état. La première colonne indique le nom ou le numéro de la transition d'état. La deuxième colonne indique l'état actuel. La troisième colonne indique les événements, les conditions et les actions. La quatrième colonne indique l'état suivant. Lorsqu'un événement ou une condition se produit, l'action est exécutée et le diagramme d'états passe à l'état suivant.

Tableau 1 – Éléments de la description d'un diagramme d'états

N	État actuel	Événement /Condition => Action	État suivant

Tableau 2 – Description des éléments d'un diagramme d'états

Intitulé	Description
N	nom ou numéro de transition d'état
État actuel	état courant
État suivant	état de destination
Événement	description de l'événement
Condition	expression logique représentant la condition
=> Action	action exécutée après que l'événement ou la condition s'est produit(e)

Tableau 3 – Conventions utilisées dans les diagrammes d'états

Symbole	Description
=	Substitution du côté droit par le côté gauche
==	Condition logique indiquant qu'une entité de gauche est égale à une entité de droite.
!=	Condition logique indiquant qu'une entité de gauche n'est pas égale à une entité de droite.
<	Condition logique indiquant qu'une entité de gauche est inférieure à une entité de droite.
>	Condition logique indiquant qu'une entité de gauche est supérieure à une entité de droite.
&&	"AND" (c'est-à-dire: ET) logique
||	"OR" (c'est-à-dire: OU) logique
!	Opérateur de négation
+ - * /	Opérateur arithmétique
;	Point d'interruption

4 Description de la syntaxe de la couche FAL

4.1 Syntaxe abstraite des FAL-PDU de type C

4.1.1 Syntaxe abstraite de base

Les définitions des FAL-PDU sont indiquées ci-dessous.

```
FAL-PDU::= CHOICE {
    connect-PDU          [0] Connect-PDU,
    connectAck-PDU       [1] ConnectAck-PDU,
    scan-PDU             [2] Scan-PDU,
```

```
        collect-PDU                 [3] Collect-PDU,
        select-PDU                  [4] Select-PDU,
        launch-PDU                  [5] Launch-PDU,
        token-PDU                   [6] Token-PDU,
        myStatus-PDU                [7] MyStatus-PDU,
        transient1-PDU              [8] Transient1-PDU,
        dummy-PDU                   [9] Dummy-PDU,
        transient2-PDU              [10] Transient2-PDU,
        ntnTest-PDU                 [11] NTNTest-PDU,
        cdata-PDU                   CData-PDU
    }

    CData-PDU::= CHOICE {
        cyclicDataW-PDU             [12] CyclicDataW-PDU,
        cyclicDataB-PDU             [13] CyclicDataB-PDU,
        cyclicDataOut1-PDU          [14] CyclicDataOut1-PDU,
        cyclicDataOut2-PDU          [15] CyclicDataOut2-PDU,
        cyclicDataIn1-PDU           [16] CyclicDataIn1-PDU,
        cyclicDataIn2-PDU           [17] CyclicDataIn2-PDU
    }
```

Les éléments FALARHeader à utiliser dans chaque PDU sont indiqués ci-dessous:

```
    FALARHeader::= SEQUENCE {
        arFType                     ARFType,
        priority                    Priority,
        scanNumber                  ScanNumber,
        reserved1                   Unsigned8,
        srcNodeNumber               NodeNumber,
        reserved2                   Unsigned16,
        hec                         Hec
    }
```

4.1.2 Connect-PDU

```
    Connect-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        portChoice                  PortChoice,
        padding                     OctetString SIZE(28),
        dcs                         DCS
    }
```

4.1.3 ConnectAck-PDU

```
    ConnectAck-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        portCheckResult             PortCheckResult,
        destPortInfo                DestPortInfo,
```

```
        padding                 OctetString SIZE(20),
        dcs                     DCS
    }
```

4.1.4 Scan-PDU

```
    Scan-PDU::= SEQUENCE {
        falArHeader             FALARHeader,
        scanState               ScanState,
        sendTime                SendTime,
        padding                 OctetString SIZE(20),
        dcs                     DCS
    }
```

4.1.5 Collect-PDU

```
    Collect-PDU::= SEQUENCE {
        falArHeader             FALARHeader,
        vendorCode              VendorCode,
        nodeType                NodeType,
        netNumber               NetNumber,
        sendTime                SendTime,
        loopState               LoopState,
        parmTypeCyclicStatus    ParmTypeCyclicStatus,
        commonParamId           CommonParamId,
        padding                 OctetString SIZE(8),
        dcs                     DCS
    }

    CommonParamId::= SEQUENCE {
        date                    ParamDate,
        timeNodeId              ParamTime,
        checksum                ParamChecksum
    }
```

4.1.6 Select-PDU

```
    Select-PDU::= SEQUENCE {
        falArHeader             FALARHeader,
        padding                 OctetString SIZE(28),
        dcs                     DCS
    }
```

4.1.7 Launch-PDU

```
    Launch-PDU::= SEQUENCE {
        falArHeader             FALARHeader,
        padding                 OctetString SIZE(28),
```

```
        dcs                         DCS
    }
```

4.1.8 Token-PDU

```
    Token-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        padding                     OctetString SIZE(28),
        dcs                         DCS
    }
```

4.1.9 MyStatus-PDU

```
    MyStatus-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        reserved1                   Unsigned16,
        nodeType                    NodeType,
        netNumber                   NetNumber,
        reserved2                   Unsigned16,
        loopState                   LoopState,
        parmTypeCyclicStatus        ParmTypeCyclicStatus,
        commonParamId               CommonParamId,
        inFarNodeMACAddr            MACAddress,
        inFarNodeNumber             NodeNumber,
        reserved3                   Unsigned8,
        outFarNodeMACAddr           MACAddress,
        outFarNodeNumber            NodeNumber,
        reserved4                   Unsigned8,
        opState                     OpState,
        errorState                  ErrorState,
        errorCode                   ErrorCode,
        vendorCode                  VendorCode,
        deviceType                  DeviceType,
        unitTypeName                UnitTypeName,
        unitTypeCode                UnitTypeCode,
        reserved5                   Unsigned16,
        nodeInfo                    NodeInfo,
        dcs                         DCS
    }
```

4.1.10 Transient1-PDU

```
    Transient1-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        destinationGroup            DestinationGroup,
        seqNumber                   SeqNumber,
        dataId                      TraDataId,
        wholeDataSize               TraWholeDataSize,
```

```
        offsetAddr                  TraOffsetAddr,
        dataSize                    TraDataSize,
        dataType                    TraDataType,
        data                        TraData,
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.1.11 Dummy-PDU

```
    Dummy-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        dummyData                   OctetString SIZE(28..1482),
        dcs                         DCS
    }
```

4.1.12 Transient2-PDU

```
    Transient2-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        l                           Length,
        gcnt                        GateCount,
        typeSeqF                    TypeSeqF,
        fno                         FrameSequence,
        dt                          DataFrameType,
        da                          TraDstAddr,
        sa                          TraSrcAddr,
        dat                         TraDstAppType,
        sat                         TraSrcAppType,
        dmf                         TraDstModuleFlag,
        smf                         TraSrcModuleFlag,
        dna                         TraDstNetAddr,
        ds                          TraDstStaNo,
        did                         TraDstID,
        sna                         TraSrcNetAddr,
        ss                          TraSrcStaNo,
        sid                         TraSrcID,
        l1                          TraCmdLen,
        ct                          TraCmdType,
        rsv                         Unsigned8,
        aps                         TraAppSeq,
        data                        [0] TraData OPTIONAL,
        evenPadding                 [1] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.1.13 NTNTest-PDU

```
NTNTest-PDU::= SEQUENCE {
    falArHeader          FALARHeader,
    ntnTestData          NTNTestData,
    dcs                  DCS
}
```

4.1.14 CyclicDataW-PDU

```
CyclicDataW-PDU::= SEQUENCE {
    falArHeader          FALARHeader,
    seqNumber            SeqNumber,
    byteValidity         ByteValidity,
    dataSize             CycDataSize,
    offsetAddr           CycOffsetAddr,
    exSeqNumber          CycExSeqNumber,
    reserved             Unsigned16,
    wData                CycWData,
    evenPadding          [0] Unsigned8 OPTIONAL,
    dcs                  DCS
}
```

4.1.15 CyclicDataB-PDU

```
CyclicDataB-PDU::= SEQUENCE {
    falArHeader          FALARHeader,
    seqNumber            SeqNumber,
    byteValidity         ByteValidity,
    dataSize             CycDataSize,
    offsetAddr           CycOffsetAddr,
    reserved1            Unsigned16,
    reserved2            Unsigned16,
    bData                CycBData,
    evenPadding          [0] Unsigned8 OPTIONAL,
    dcs                  DCS
}
```

4.1.16 CyclicDataOut1-PDU

```
CyclicDataOut1-PDU::= SEQUENCE {
    falArHeader          FALARHeader,
    seqNumber            SeqNumber,
    byteValidity         ByteValidity,
    dataSize             CycDataSize,
    offsetAddr           CycOffsetAddr,
    reserved1            Unsigned16,
    reserved2            Unsigned16,
    out1Data             CycOut1Data,
```

```
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.1.17 CyclicDataOut2-PDU

```
    CyclicDataOut2-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        seqNumber                   SeqNumber,
        byteValidity                ByteValidity,
        dataSize                    CycDataSize,
        offsetAddr                  CycOffsetAddr,
        reserved1                   Unsigned16,
        reserved2                   Unsigned16,
        out2Data                    CycOut2Data,
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.1.18 CyclicDataIn1-PDU

```
    CyclicDataIn1-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        seqNumber                   SeqNumber,
        byteValidity                ByteValidity,
        dataSize                    CycDataSize,
        offsetAddr                  CycOffsetAddr,
        reserved1                   Unsigned16,
        reserved2                   Unsigned16,
        in1Data                     CycIn1Data,
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.1.19 CyclicDataIn2-PDU

```
    CyclicDataIn2-PDU::= SEQUENCE {
        falArHeader                 FALARHeader,
        seqNumber                   SeqNumber,
        byteValidity                ByteValidity,
        dataSize                    CycDataSize,
        offsetAddr                  CycOffsetAddr,
        reserved1                   Unsigned16,
        reserved2                   Unsigned16,
        in2Data                     CycIn2Data,
        evenPadding                 [0] Unsigned8 OPTIONAL,
        dcs                         DCS
    }
```

4.2 Syntaxe abstraite des FAL PDU de type F

4.2.1 Syntaxe abstraite de base

Les définitions des FAL-PDU sont indiquées ci-dessous.

```
FAL-PDU::= CHOICE {
    f-channelControl-PDU        F-ChannelControl-PDU,
    f-sync-PDU                  F-Sync-PDU,
    f-cyclicData-PDU            F-CData-PDU,
    f-transientData-PDU         F-TraData-PDU
}

F-ChannelControl-PDU::= CHOICE {
    persuasion-PDU              [16] Persuasion-PDU,
    testData-PDU                [17] TestData-PDU,
    testDataAck-PDU             [18] TestDataAck-PDU,
    setup-PDU                   [19] Setup-PDU,
    setupAck-PDU                [20] SetupAck-PDU,
    token-PDU                   [21] F-Token-PDU,
    myStatus-PDU                [32] F-MyStatus-PDU
}

F-Sync-PDU::= CHOICE {
    measure-PDU                 [50] F-Measure-PDU,
    measureAck-PDU              [51] F-Measure-PDU,
    offset-PDU                  [52] F-Offset-PDU,
    update-PDU                  [53] F-Update-PDU
}

F-CData-PDU::= CHOICE {
    cyclicDataRWw-PDU           [130] F-CyclicData-PDU,
    cyclicDataRY-PDU            [131] F-CyclicData-PDU,
    cyclicDataRWr-PDU           [132] F-CyclicData-PDU,
    cyclicDataRX-PDU            [133] F-CyclicData-PDU
}

F-TraData-PDU::= CHOICE {
    transient1-PDU              [34] Transient1-PDU,
    transientAck-PDU            [35] TransientAck-PDU,
    transient2-PDU              [37] Transient2-PDU,
    paramCheck-PDU              [40] ParamCheck-PDU,
    parameter-PDU               [41] Parameter-PDU,
    timer-PDU                   [44] Timer-PDU
}
```

Les éléments FALARHeader utilisés dans chaque PDU sont indiqués ci-dessous.

```
FALAR-FHeader::= SEQUENCE {
    arFType                 ARFType,
    dataType                DataType,
    varField                CHOICE {
        vField0 [0] SEQUENCE {
            persPriority    PersPriority,
            nodeType        NodeType
        },
        vField1 [1] SEQUENCE {
            reserved1       OCTET STRING (SIZE (4))
        },
        vField2 [2] SEQUENCE {
            nodeId          NodeId,
            reserved2       OCTET STRING (SIZE (2))
        },
        vField3 [3] SEQUENCE {
            nodeId          NodeId,
            syncFlag        SyncFlag,
            nodeType        NodeType
        },
        vField4 [4] SEQUENCE {
            nodeId          NodeId,
            connectionInfo  ConnectionInfo,
            reserved4       OCTET STRING (SIZE (1))
        }
    },
    srcNodeNumber           NodeNumber,
    protocolVerType         ProtocolVerType,
    reserved                OCTET STRING (SIZE (1)),
    hec                     Hec
}
```

4.2.2 Persuasion-PDU

```
Persuasion-PDU::= SEQUENCE {
    falArHeader             FALAR-FHeader,
    reserved1               OCTET STRING (SIZE (1)),
    myPorts                 Unsigned8,
    vendorCode              VendorCode,
    modelCode               ModelCode,
    reserved2               OCTET STRING (SIZE (20)),
    dcs                     DCS
}
```

4.2.3 TestData-PDU

```
TestData-PDU::= SEQUENCE {
```

```
        falArHeader                 FALAR-FHeader,
        tmMacAddr                   MACAddress,
        srcPort                     PortNumber,
        reserved                    OCTET STRING (SIZE (21)),
        dcs                         DCS
    }
```

4.2.4 TestDataAck-PDU

```
    TestDataAck-PDU::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        tdSrcMacAddr                MACAddress,
        tdSrcPort                   PortNumber,
        tdRcvPort                   PortNumber,
        reserved1                   OCTET STRING (SIZE (1)),
        myPorts                     Unsigned8,
        tokenKeepTime               Unsigned16,
        reserved2                   OCTET STRING (SIZE (4)),
        myConnectStatus             SEQUENCE {
            port2port1              PortStatus,
            port4port3              PortStatus,
            port6port5              PortStatus,
            port8port7              PortStatus,
            port10port9             PortStatus,
            port12port11            PortStatus,
            port14port13            PortStatus,
            port16port15            PortStatus,
            port18port17            PortStatus,
            port20port19            PortStatus,
            port22port21            PortStatus,
            port24port23            PortStatus
        },
        dcs                         DCS
    }
```

4.2.5 Setup-PDU

```
    Setup-PDU::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        tokenDstMacAddr             MACAddress,
        reserved1                   OCTET STRING (SIZE (2)),
        leaveTimerValue             LeaveTimer,
        portUsage                   PortUsage,
        reserved2                   OCTET STRING (SIZE (1)),
        netBehaviour                NetworkBehaviour,
        reserved3                   OCTET STRING (SIZE (12)),
        dcs                         DCS
```

```
}

NetworkBehaviour::= SEQUENCE {
    multipleTranmit          Unsigned8,
    frameInterval            Unsigned8,
    reserved                 OCTET STRING (SIZE (1)),
    multipleTokens           Unsigned8
}
```

4.2.6 SetupAck-PDU

```
SetupAck-PDU::= SEQUENCE {
    falArHeader              FALAR-FHeader,
    slaveNodeInfo            SlaveNodeInfo,
    fwVersion                Version,
    deviceType               DeviceType,
    reserved1                OCTET STRING (SIZE (2)),
    vendorCode               VendorCode,
    modelCode                ModelCode,
    rySize                   Unsigned16,
    rwwSize                  Unsigned16,
    rxSize                   Unsigned16,
    rwrSize                  Unsigned16,
    reserved2                OCTET STRING (SIZE (2)),
    availableFuncs           AvailableFuncs,
    reserved3                OCTET STRING (SIZE (5)),
    dcs                      DCS
}
```

4.2.7 F-Token-PDU

```
F-Token-PDU::= SEQUENCE {
    falArHeader              FALAR-FHeader,
    tokenDstMacAddr          MACAddress,
    tokenSeqNumber           Unsigned8,
    reserved1                OCTET STRING (SIZE (1)),
    tokenHopCounter          Unsigned16,
    traAvailHopCounter       Unsigned16,
    traLastHopCounter        Unsigned16,
    traAllows                Unsigned8,
    reserved2                OCTET STRING (SIZE (13)),
    dcs                      DCS
}
```

4.2.8 F-MyStatus-PDU

```
F-MyStatus-PDU::= SEQUENCE {
    falArHeader              FALAR-FHeader,
```

```
        seqNumber                   SeqNumber,
        netNumber                   NetNumber,
        masterCmd                   Unsigned16,
        cyclicStatus                Unsigned16,
        nodeStatus                  Unsigned16,
        errorCode                   ErrorCode,
        portStatus                  SEQUENCE {lower PortStatus,
                                    upper PortStatus },
        portStatistics              SEQUENCE {lower PortStatistics,
                                    upper PortStatistics },
        portIndex                   Unsigned8,
        reserved                    OCTET STRING (SIZE (3)),
        cyclicSeqNumber             Unsigned8,
        addrTableDistResult         Unsigned8,
        slaveSpfEventInfo1          Unsigned8,
        slaveSpfEventInfo2          Unsigned16,
        vendorSpfNodeInfo           OCTET STRING (SIZE (4)),
        dcs                         DCS
    }
```

4.2.9 Measure-PDU

```
    F-Measure-PDU::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        reserved                    OCTET STRING (SIZE (28)),
        dcs                         DCS
    }
```

4.2.10 F-Offset-PDU

```
    F-Offset-PDU::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        reserved                    OCTET STRING (SIZE (8)),
        syncOffset                  SyncOffset,
        reserved2                   OCTET STRING (SIZE (16)),
        dcs                         DCS
    }
```

4.2.11 F-Update-PDU

```
    F-Update-PDU::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        reserved                    OCTET STRING (SIZE (8)),
        syncOffset                  SyncOffset,
        reserved2                   OCTET STRING (SIZE (16)),
        dcs                         DCS
    }
```

4.2.12 F-CyclicData-PDU

```
F-CyclicData-PDU::= SEQUENCE {
    falArHeader             FALAR-FHeader,
    seqNumber               SeqNumber,
    bothEndsValidity        BothEndsValidity,
    cycDataSize             Unsigned16,
    offsetAddr              CycOffsetAddr,
    reserved                OCTET STRING (SIZE (4)),
    cycData                 CycData,
    dcs                     DCS
}
```

4.2.13 Transient1-PDU

```
Transient1-PDU::= SEQUENCE {
    falArHeader             FALAR-FHeader,
    traMsgHeader            TraMsgHeader,
    data                    OCTET STRING (SIZE
                            (12..1466)),
    dcs                     DCS
}

TraMsgHeader::= SEQUENCE {
    reserved                OCTET STRING (SIZE (4)),
    seqNumber               SeqNumber,
    dataId                  TraDataId,
    wholeDataSize           TraWholeDataSize,
    offsetAddr              TraOffsetAddr,
    dataSize                TraDataSize,
    dataSubType             TraDataSubType
}

FieldSpecificTransient::= SEQUENCE {
    opHeader                TraMsgCmdExHeader,
    fSTraData               CHOICE {
        nodeInfoDist        [721] TraSysNodeInfoDist,
        statisticsGet       [723] TraSysStatisticsGet,
        nodeInfoDetailGet   [724] TraSysNodeInfoDetailGet,
        ...
    }
}

TraMsgCmdExHeader::= SEQUENCE {
    command                 TraCommand,
    subCommand              TraSubCommand,
    rtn                     CHOICE {
        reserved            [0] OCTET STRING (SIZE (2)),
```

```
            value                       [1] Unsigned16
        },
        reserved1                       OCTET STRING (SIZE (1)),
        destNetNumber                   NetNumber,
        destNodeNumber                  NodeNumber,
        reserved2                       OCTET STRING (SIZE (5)),
        srcNetNumber                    NetNumber,
        srcNodeNumber                   NodeNumber,
        reserved3                       OCTET STRING (SIZE (4))
}

TraSysNodeInfoDist::= SEQUENCE {
        seqNumber                       SeqNumber,
        masterNetNumber                 NetNumber,
        masterDeviceType                DeviceType,
        masterModelCode                 ModelCode,
        masterVendorCode                VendorCode,
        masterNodeType                  NodeType,
        Reserved1                       OCTET STRING (SIZE (1)),
        masterMacAddress                MACAddress
        Reserved2                       OCTET STRING (SIZE (2)),
        dataNum                         Unsigned32,
        messages                        SEQUENCE OF
                                        NodeInfoMessage
}

NodeInfoMessage::= SEQUENCE {
        nodeNumber                      NodeNumber,
        reserved1                       OCTET STRING (SIZE (1)),
        availableFuncs                  AvailableFuncs,
        reserved2                       OCTET STRING (SIZE (1)),
        netNumber                       NetNumber,
        deviceType                      DeviceType,
        modelCode                       ModelCode,
        vendorCode                      VendorCode,
        nodeType                        NodeType,
        reserved3                       OCTET STRING (SIZE (1)),
        macAddress                      MACAddress,
        reserved4                       OCTET STRING (SIZE (2))
}

TraSysStatisticsGet::= CHOICE {
        statGetRequest                  [0] SEQUENCE {
        },
        statGetResponse                 [1] SEQUENCE {
            port1Mib1                   Unsigned32,
            port1Mib2                   Unsigned32,
```

```
        port1Mib3                   Unsigned32,
        port1Mib4                   Unsigned32,
        port1Mib5                   Unsigned32,
        port1Mib6                   Unsigned32,
        port1Mib7                   Unsigned32,
        reserved                    OCTET STRING (SIZE (4)),
        port2Mib1                   Unsigned32,
        port2Mib2                   Unsigned32,
        port2Mib3                   Unsigned32,
        port2Mib4                   Unsigned32,
        port2Mib5                   Unsigned32,
        port2Mib6                   Unsigned32,
        port2Mib7                   Unsigned32,
        healthStatusNum             Unsigned32,
        healthStatus                SEQUENCE SIZE (0..128) OF
                                    Unsigned32
    }
}

TraSysNodeInfoDetailGet::= CHOICE {
    nodeInfoDetailGetRequest        [0] SEQUENCE {
    },
    nodeInfoDetailGetResponse       [1] SEQUENCE {
        rySize                      Unsigned16,
        rwwSize                     Unsigned16,
        rxSize                      Unsigned16,
        rwrSize                     Unsigned16,
        reserved1                   OCTET STRING (SIZE (1)),
        ports                       Unsigned8,
        tokenKeepTime               Unsigned16,
        netBehaviour                NetworkBehaviour,
        nodeInfo                    SlaveNodeInfo,
        fwVersion                   Version,
        deviceType                  DeviceType,
        modelCode                   ModelCode,
        vendorCode                  VendorCode,
        reserved2                   OCTET STRING (SIZE (2)),
        modelName                   OCTET STRING (SIZE (20)),
        vendorName                  OCTET STRING (SIZE (32)),
        contInfo                    Unsigned8,
        contFwVersion               Version,
        contDeviceType              DeviceType,
        contModelCode               ModelCode,
        contVendorCode              VendorCode,
        reserved3                   OCTET STRING (SIZE (2)),
        contModelName               OCTET STRING (SIZE (20)),
        contVendorName              OCTET STRING (SIZE (32)),
```

```
        contVendorSpecificInfo          OCTET STRING (SIZE (4))
    }
}
```

4.2.14 TransientAck-PDU

```
TransientAck-PDU::= SEQUENCE {
    falArHeader                         FALAR-FHeader,
    acks                                Unsigned32,
    ackData                             SEQUENCE OF TraAckData,
    dcs                                 DCS
}
```

4.2.15 Transient2-PDU

```
Transient2-PDU::= SEQUENCE {
    falArHeader                         FALAR-FHeader,
    l                                   TraLength,
    reserved                            OCTET STRING (SIZE (1)),
    tp                                  TraType,
    fno                                 TraFrameSequence,
    dt                                  TraDataFrameType,
    da                                  TraDstAddr,
    sa                                  TraSrcAddr,
    dat                                 TraDstAppType,
    sat                                 TraSrcAppType,
    dmf                                 TraDstModuleFlag,
    smf                                 TraSrcModuleFlag,
    dna                                 TraDstNetAddr,
    ds                                  TraDstStaNo,
    did                                 TraDstID,
    sna                                 TraSrcNetAddr,
    ss                                  TraSrcStaNo,
    sid                                 TraSrcID,
    l1                                  TraCmdLen,
    ct                                  TraCmdType,
    dno                                 TraDataNo,
    aps                                 TraAppSeq,
    rsts                                TraReturnStatus,
    data                                Tra2Data,
    dcs                                 DCS
}
```

4.2.16 ParamCheck-PDU

```
ParamCheck-PDU::= SEQUENCE {
    falArHeader                         FALAR-FHeader,
    reserved1                           OCTET STRING (SIZE (4)),
```

```
        paramId                 CommonParamId,
        reserved2               OCTET STRING (SIZE (12)),
        dcs                     DCS
}

CommonParamId::= SEQUENCE {
        date                    ParamDate,
        timeNodeId              ParamTime,
        checksum                ParamChecksum
}
```

4.2.17 Parameter-PDU

```
Parameter-PDU::= SEQUENCE {
        falArHeader             FALAR-FHeader,
        paramSetFlag            ParamFlag,
        addressOrder            AddressOrder,
        cmdOrder                CmdOrder,
        cyclicParameter         CyclicParameter,
        dcs                     DCS
}

AddressOrder::= SEQUENCE {
        assignedNetNumber       NetNumber,
        assignedNodeNumber      NodeNumber
}

CmdOrder::= SEQUENCE {
        cmd                     Unsigned24,
        nodeType                NodeType
}

CyclicParameter::= SEQUENCE {
        paramId                 CommonParamId,
        reserved1               OCTET STRING (SIZE (2)),
        masterStatus            Unsigned16,
        rySeqNumber             SeqNumber,
        ryBothEndsValidity      BothEndsValidity,
        ryDataSize              Unsigned16,
        ryOffset                Unsigned16,
        reserved2               OCTET STRING (SIZE (2)),
        rwwSeqNumber            SeqNumber,
        reserved3               OCTET STRING (SIZE (1)),
        rwwDataSize             Unsigned16,
        rwwOffset               Unsigned16,
        reserved4               OCTET STRING (SIZE (3)),
        rxBothEndsValidity      BothEndsValidity,
```

```
        rxDataSize                  Unsigned16,
        rxOffset                    Unsigned32,
        reserved5                   OCTET STRING (SIZE (2)),
        rwrDataSize                 Unsigned16,
        rwrOffset                   Unsigned32,
        reserved6                   OCTET STRING (SIZE (4)),
        masterWatchTimer            Unsigned16,
        reserved7                   OCTET STRING (SIZE (3)),
        cmRyBothEndsValidity        BothEndsValidity,
        cmRyDataSize                Unsigned16,
        cmRyOffset                  Unsigned32,
        reserved8                   OCTET STRING (SIZE (2)),
        cmRwwDataSize               Unsigned16,
        cmRwwOffset                 Unsigned32,
        reserved9                   OCTET STRING (SIZE (1)),
        cmRxBothEndsValidity        BothEndsValidity,
        cmRxDataSize                Unsigned16,
        cmRxOffset                  Unsigned32,
        reserved10                  OCTET STRING (SIZE (2)),
        cmRwrDataSize               Unsigned16,
        cmRwrOffset                 Unsigned32
    }
```

4.2.18 Timer-PDU

```
    Timer-PDU::= SEQUENCE {
        falArHeader                 FALAR-FHeader,
        time                        Timer,
        reserved                    OCTET STRING (SIZE (22)),
        dcs                         DCS
    }
```

4.3 Affectation des types de données pour le type C

Les types de données utilisés dans la syntaxe abstraite de FAL-PDU de type C sont indiqués ci-dessous.

```
ARFType::= Unsigned8
DCS::= Unsigned8
Priority::= Unsigned8
ScanNumber::= Unsigned24
NodeNumber::= Unsigned16
Hec::= Unsigned32
PortChoice::= Unsigned32
PortCheckResult::= Unsigned32
DestPortInfo::= Unsigned32
ScanState::= Unsigned32
SendTime::= Unsigned16
VendorCode::= Unsigned16
NodeType::= Unsigned8
NetNumber::= Unsigned8
LoopState::= Unsigned8
ParmTypeCyclicStatus::= Unsigned8
ParamDate::= Unsigned32
```

```
ParamTime::= Unsigned32
ParamChecksum::= Unsigned32
OpState::= Unsigned16
ErrorState::= Unsigned16
ErrorCode::= Unsigned16
DeviceType::= Unsigned16
UnitTypeName::= VisibleString SIZE(20)
UnitTypeCode::= Unsigned16
NodeInfo::= OctetString SIZE(96)
DestinationGroup::= Unsigned32
SeqNumber::= Unsigned8
TraDataId::= Unsigned8
TraWholeDataSize::= Unsigned16
TraOffsetAddr::= Unsigned32
TraDataSize::= Unsigned16
TraDataType::= Unsigned16
TraData::= LOctetString SIZE(12..1466)
Length::= Unsigned16
GateCount::= Unsigned8
TypeSeqF::= Unsigned8
FrameSequence::= Unsigned8
DataFrameType::= Unsigned8
TraDstAddr::= Unsigned8
TraSrcAddr::= Unsigned8
TraDstAppType::= Unsigned8
TraSrcAppType::= Unsigned8
TraDstModuleFlag::= Unsigned8
TraSrcModuleFlag::= Unsigned8
TraDstNetAddr::= Unsigned8
TraDstStaNo::= Unsigned8
TraDstID::= Unsigned16
TraSrcNetAddr::= Unsigned8
TraSrcStaNo::= Unsigned8
TraSrcID::= Unsigned16
TraCmdLen::= Unsigned16
TraCmdType::= Unsigned8
TraAppSeq::= Unsigned16
Tra2Data::= LOctetString SIZE(12..1466)
NTNTestData::= OctetString SIZE(20..1480)
ByteValidity::= Unsigned8
CycDataSize::= Unsigned16
CycOffsetAddr::= Unsigned32
CycExSeqNumber::= Unsigned16
CycWData::= LOctetString SIZE(16..1468)
CycBData::= LOctetString SIZE(16..1468)
CycOut1Data::= LOctetString SIZE(16..1468)
CycOut2Data::= LOctetString SIZE(16..1468)
CycIn1Data::= LOctetString SIZE(16..1024)
CycIn2Data::= LOctetString SIZE(16..1024)
```


4.4 Affectation des types de données pour le type F

Les types de données utilisés dans la syntaxe abstraite des FAL-PDU de type F sont indiqués ci-dessous.

```
DCS::= Unsigned32
ARFType::= Unsigned8
DataType::= Unsigned8
NodeNumber::= Unsigned16
ProtocolVerType::= Unsigned8
Hec::= Unsigned32
PersPriority::= Unsigned24
NodeType::= Unsigned8
NodeId::= Unsigned16
ConnectionInfo::= Unsigned8
VendorCode::= Unsigned16
ModelCode::= Unsigned32
PortNumber::= Unsigned8
```

```
TraControl::= Unsigned8
PortStatus::= Unsigned8
LeaveTimer::= Unsigned16
PortUsage::= Unsigned8
SlaveNodeInfo::= Unsigned8
Version::= Unsigned8
DeviceType::= Unsigned16
AvailableFuncs::= Unsigned8
SeqNumber::= Unsigned8
NetNumber::= Unsigned8
PortStatistics::= Unsigned8
ErrorCode::= Unsigned32
ParamFlag::= Unsigned8
ParamDate::= Unsigned32
ParamTime::= Unsigned32
ParamChecksum::= Unsigned32
Timer::= Unsigned48
TraDataSubType::= Unsigned16
TraDataId::= Unsigned8
TraReturnValue::= Unsigned16
TraWholeDataSize::= Unsigned16
TraOffsetAddr::= Unsigned32
TraDataSize::= Unsigned16
TraCommand::= Unsigned8
TraSubCommand::= Unsigned8
TraLength::= Unsigned16
TraType::= Unsigned8
TraFrameSequence::= Unsigned8
TraDataFrameType::= Unsigned8
TraDstAddr::= Unsigned8
TraSrcAddr::= Unsigned8
TraDstAppType::= Unsigned8
TraSrcAppType::= Unsigned8
TraDstModuleFlag::= Unsigned8
TraSrcModuleFlag::= Unsigned8
TraDstNetAddr::= Unsigned8
TraDstStaNo::= Unsigned8
TraDstID::= Unsigned16
TraSrcNetAddr::= Unsigned8
TraSrcStaNo::= Unsigned8
TraSrcID::= Unsigned16
TraCmdLen::= Unsigned16
TraCmdType::= Unsigned8
TraDataNo::= Unsigned8
TraAppSeq::= Unsigned16
TraReturnStatus::= Unsigned16
Tra2Data::= LOctetString (SIZE(0..960))
BothEndsValidity::= Unsigned8
CycOffsetAddr::= Unsigned32
CycData::= LOctetString
OctetString::= OCTET STRING
BitString8::= OCTET STRING (SIZE (1))
BitString16::= OCTET STRING (SIZE (2))
BitString32::= OCTET STRING (SIZE (3))
Unsigned8::= OCTET STRING (SIZE (1))
Unsigned16::= OCTET STRING (SIZE (2))
Unsigned24::= OCTET STRING (SIZE (3))
Unsigned32::= OCTET STRING (SIZE (4))
Unsigned48::= OCTET STRING (SIZE (6))
MACAddress::= OCTET STRING (SIZE (6))
LOctetString::= OCTET STRING
SyncOffset::= Unsigned32
SyncFlag::= Unsigned8
```

5 Syntaxe de transfert FAL

5.1 Règles de codage

5.1.1 Codage de valeur Unsigned

Les valeurs non signées de longueur fixe Unsigned8, Unsigned16, Unsigned24 et Unsigned32 sont respectivement codées comme valeurs entières non signées d'un octet, deux octets, trois octets et quatre octets de long. Les valeurs Unsigned16, Unsigned24 et Unsigned32 sont codées comme gros-boutistes. L'octet de poids fort est considéré comme le premier octet, l'octet suivant comme le deuxième octet et l'octet de poids faible comme le dernier octet.

5.1.2 Codage de chaîne d'octets

La valeur OctetString, dont la chaîne d'octets est de longueur variable, est codée octet par octet, dans l'ordre séquentiel.

5.1.3 Codage de SEQUENCE

Le codage de SEQUENCE (et de SEQUENCE OF) est exécuté selon une séquence, en commençant par l'élément initial. Les identificateurs et la longueur utilisés dans l'ASN.1 ne sont pas utilisés.

5.1.4 Codage de LOctetString

La valeur LOctetString est de longueur variable et codée comme petit-boutiste. L'octet le plus bas, ou le moins important, est le premier octet, puis l'ordre séquentiel est suivi, jusqu'à l'octet le plus haut, ou le plus important, qui est le dernier octet.

5.2 Codage des éléments de FAL-PDU de type C

5.2.1 FALARHeader

5.2.1.1 arFType

Ce champ indique les types de PDU décrits dans le Tableau 4.

Tableau 4 – afFType

Valeur	Description
0x00	Connect-PDU
0x01	ConnectAck-PDU
0x02	Scan-PDU
0x03	Collect-PDU
0x04	Select-PDU
0x05	Launch-PDU
0x06	Token-PDU
0x07..0x1F	Reserved
0x20	MyStatus-PDU
0x22	Transient1-PDU
0x23	Reserved
0x24	Dummy-PDU
0x25	Transient2-PDU
0x26..0x2E	Reserved
0x2F	NTNTest-PDU

Valeur	Description
0x30..0x7F	Reserved
0x80	CyclicDataW-PDU
0x81	CyclicDataB-PDU
0x82..0x8B	Reserved
0x8C	CyclicDataOut1-PDU
0x8D	CyclicDataOut2-PDU
0x8E	CyclicDataIn1-PDU
0x8F	CyclicDataIn2-PDU

5.2.1.2 priority

Ce champ indique les priorités décrites dans le Tableau 5.

Tableau 5 – priority

Valeur	Description
0x00	Select-PDU, Launch-PDU, Token-PDU, Dummy-PDU, CyclicDataW-PDU, CyclicDataB-PDU, CyclicDataOut1-PDU, CyclicDataOut2-PDU, CyclicDataIn1-PDU, CyclicDataIn2-PDU
0x01	Collect-PDU, Connect-PDU, ConnectAck-PDU, Scan-PDU
0x02	MyStatus-PDU
0x03	Transient1-PDU
0x04	Transient2-PDU, NTNTest-PDU

5.2.1.3 scanNumber

Ce champ contient un identificateur de trame utilisé dans Scan-PDU. Le numéro est incrémenté à chaque envoi de Scan-PDU. La valeur par défaut de 0x000000 est utilisée dans tous les autres PDU.

5.2.1.4 reserved1

Ce champ est réservé à un usage futur. La valeur est 0x00.

5.2.1.5 srcNodeNumber

Ce champ contient le numéro identificateur du nœud source.

5.2.1.6 reserved2

Ce champ est réservé à un usage futur. La valeur est 0x0000.

5.2.1.7 hec

Ce champ est un code de vérification d'erreur de DestAddr (DLPDU) jusqu'à reserved2 (FALARHeader). Les définitions DLPDU sont les suivantes.

```
DLPDU::= SEQUENCE {
    preamble        Preamble,
    sfd             SFD,
    destaddr        DestAddr,
    srcaddr         SrcAddr,
    lt              LT,
    dlsdu           FAL-PDU,
    fcs             FCS
}

Preamble::= OctetString SIZE(7)
SFD::= OctetString SIZE(1)
DestAddr::= MACAddress
SrcAddr::= MACAddress
LT::= Unsigned16
FCS::= OctetString SIZE(4)
```

Le polynôme générateur est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$.

5.2.2 Connect-PDU

5.2.2.1 falArHeader

Voir 5.2.1.

5.2.2.2 portChoice

Ce champ indique les types de ports décrits dans le Tableau 6.

Tableau 6 – portChoice

Valeur	Description
0x0	Côté "entrée"
0x1	Côté "sortie"
0xF	Côté "sortie", pendant l'essai entre les nœuds

5.2.2.3 padding

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.2.4 dcs

Ce champ est un code de vérification d'erreur à partir de DestAddr (DLPDU). Pour connaître les définitions des DLPDU, voir 5.2.1.7.

Le polynôme générateur est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$.

5.2.3 ConnectAck-PDU

5.2.3.1 falArHeader

Voir 5.2.1.

5.2.3.2 portCheckResult

Ce champ indique les résultats de la vérification de la connexion Entrée-Sortie. Les valeurs sont spécifiées dans le Tableau 7.

Tableau 7 – portCheckResult

Valeur	Description
0x0	OK
0x1	Pas OK
0xE	Essai entre les nœuds OK
0xF	Essai entre les nœuds pas OK

5.2.3.3 dstPortInfo

Ce champ indique les types de ports de destination. Les valeurs sont spécifiées dans le Tableau 8.

Tableau 8 – dstPortInfo

Valeur	Description
0	Côté "entrée"
1	Côté "sortie"

5.2.3.4 Remplissage

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.3.5 dcs

Voir 5.2.2.4.

5.2.4 Scan-PDU

5.2.4.1 falArHeader

Voir 5.2.1.

5.2.4.2 scanState

Ce champ indique l'état du chemin de transmission. Les valeurs sont spécifiées dans le Tableau 9.

Tableau 9 – scanState

Valeur	Description
0	État de traversée
1	État du rebouclage du côté "entrée"
2	État du rebouclage du côté "sortie"

5.2.4.3 sendTime

Ce champ indique le temps envoyé.

5.2.4.4 Remplissage

Ce champ concerne un bourrage. La valeur est 0x00.

5.2.4.5 dcs

Voir 5.2.2.4.

5.2.5 Collect-PDU

5.2.5.1 falArHeader

Voir 5.2.1.

5.2.5.2 vendorCode

Ce champ indique le code du fournisseur. Pour connaître les codes des fournisseurs, voir les règles de mise en œuvre.

5.2.5.3 nodeType

Ce champ indique les types de nœuds décrits dans le Tableau 10.

Tableau 10 – nodeType

Valeur	Description
0x00	Nœud de gestion
0x01	Nœud de réserve (pour une extension ultérieure)
0x02	Nœud normal
0x10	Nœud de réserve (pour une extension ultérieure)
0x12	Nœud de réserve (pour une extension ultérieure)
0x20	Nœud de réserve (pour une extension ultérieure)
0x21	Nœud maître dans l'essai entre nœuds

5.2.5.4 netNumber

Ce champ contient l'identificateur du réseau auquel le nœud appartient. La plage est 1..239.

5.2.5.5 sendTime

Voir 5.2.4.3.

5.2.5.6 loopState

Ce champ indique l'état de boucle du nœud. Les valeurs sont spécifiées dans le Tableau 11.

Tableau 11 – loopState

Valeur	Description
0x00	Traversée
0x12	Rebouclage vers le côté "Entrée"; côté "Sortie" déconnecté

Valeur	Description
0x13	Retour de boucle sur le côté "Entrée", erreur de vérification d'entrée/sortie côté "Sortie"
0x14	Rebouclage vers le côté "entrée"; vérification d'entrée/sortie en cours du côté "sortie"
0x21	Côté "Entrée" déconnecté, retour de la boucle sur le côté "Sortie"
0x31	Erreur de vérification d'entrée/sortie du côté "entrée"; rebouclage vers le côté "sortie"
0x41	Vérification d'entrée/sortie en cours du côté "entrée"; rebouclage vers le côté "sortie"

5.2.5.7 parmTypeCyclicStatus

Ce champ est divisé de la façon suivante:

Bit 0..5 État cyclique – suivre les valeurs du Tableau 12

Bit 6..7 Mode paramétrage – suivre les valeurs du Tableau 13

Tableau 12 – État cyclique

Valeur	Description
0x00	La transmission cyclique n'est exécutée
0x01	La transmission cyclique est exécutée
0x02	Paramètres communs non reçus
0x03	Réception des paramètres communs en cours
0x04	Erreur dans les paramètres communs
0x05	Réservé
0x06	Le numéro de nœud est interdit
0x07	Paramétrage du nœud de réserve
0x08	Instruction d'arrêt de la transmission cyclique
0x09	Exécution d'essai hors ligne
0x0A	Expiration du temporisateur de surveillance
0x0B	Numéro de nœud non défini
0x0C	Erreur d'UC de nœud
0x0D	Le numéro de nœud est en double
0x0E	Duplication de nœud de gestion
0x0F	Duplication de numéro de nœud et de nœud de gestion
0x10	Erreur de numéro de réseau

Tableau 13– Mode paramétrage

Valeur	Description
0x0	Paramètre commun
0x1	Réservé

5.2.5.8 commonParamId

5.2.5.8.1 date

Les bits de ce champ sont définis comme suit:

Bit 28..31	Année (chiffre des centaines)
Bit 24..27	Année (chiffre des milliers)
Bit 20..23	Année (chiffre des unités)
Bit 16..19	Année (chiffre des dizaines)
Bit 12..15	Mois (chiffre des unités)
Bit 8..11	Mois (chiffre des dizaines)
Bit 4..7	Jour (chiffre des unités)
Bit 0..3	Jour (chiffre des dizaines)

Si tous les bits sont égaux à 0, il n'existe pas de paramètre commun.

5.2.5.8.2 timeNodeId

Les bits de ce champ sont définis comme suit:

Bit 28..31	Heures (chiffre des unités)
Bit 24..27	Heures (chiffre des dizaines)
Bit 20..23	Minutes (chiffre des unités)
Bit 16..19	Minutes (chiffre des dizaines)
Bit 12..15	Secondes (chiffre des unités)
Bit 8..11	Secondes (chiffre des dizaines)
Bit 0..7	Numéro de nœud du réglage de référence

Si tous les bits sont égaux à 0, il n'existe pas de paramètre commun.

5.2.5.8.3 Somme de contrôle

Ce champ contient la somme de contrôle "date" et "time_nodeId" dans commandParmId. Si tous les bits sont égaux à 0, il n'existe pas de paramètre commun.

5.2.5.9 Remplissage

Ce champ comprend 8 octets. La valeur est 0x00 pour les 8 octets.

5.2.5.10 dcs

Voir 5.2.2.4.

5.2.6 Select-PDU

5.2.6.1 falArHeader

Voir 5.2.1.

5.2.6.2 Remplissage

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.6.3 dcs

Voir 5.2.2.4.

5.2.7 Launch-PDU

5.2.7.1 falArHeader

Voir 5.2.1.

5.2.7.2 Remplissage

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.7.3 dcs

Voir 5.2.2.4.

5.2.8 Token-PDU

5.2.8.1 falArHeader

Voir 5.2.1.

5.2.8.2 Remplissage

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.8.3 dcs

Voir 5.2.2.4.

5.2.9 MyStatus-PDU

5.2.9.1 falArHeader

Voir 5.2.1.

5.2.9.2 reserved1

Ce champ est réservé à un usage futur. La valeur est 0x0000.

5.2.9.3 nodeType

Voir 5.2.5.3.

5.2.9.4 netNumber

Voir 5.2.5.4.

5.2.9.5 reserved2

Ce champ est réservé à un usage futur. La valeur est 0x0000.

5.2.9.6 loopState

Voir 5.2.5.6.

5.2.9.7 parmTypeCyclicStatus

Voir 5.2.5.7.

5.2.9.8 commonParamId

Voir 5.2.5.8.

5.2.9.9 inFarNodeMACAddr

Ce champ indique l'adresse MAC du nœud connecté au côté "Entrée". Si le côté "Entrée" n'est pas connecté correctement, la valeur est 0.

5.2.9.10 inFarNodeNumber

Ce champ indique le numéro de nœud du nœud connecté au côté "Entrée".

5.2.9.11 reserved3

Ce champ est réservé à un usage futur. La valeur est 0x00.

5.2.9.12 outFarNodeMACAddr

Ce champ indique l'adresse MAC du nœud connecté au côté "Sortie". Si le côté "Sortie" n'est pas connecté correctement, la valeur est 0.

5.2.9.13 outFarNodeNumber

Ce champ indique le numéro de nœud du nœud connecté au côté "Sortie".

5.2.9.14 reserved4

Ce champ est réservé à un usage futur. La valeur est 0x00.

5.2.9.15 opState

Ce champ indique l'état d'un nœud. Les valeurs sont spécifiées dans le Tableau 14.

Tableau 14 – opState

Valeur	Description
0	Contrôleur inexistant
1	Contrôleur arrêté
2	Contrôleur en cours de fonctionnement

5.2.9.16 errorState

Ce champ indique l'état d'erreur d'un nœud. Les valeurs sont spécifiées dans le Tableau 15.

Tableau 15 – errorState

Valeur	Description
0	Absence d'erreurs
1	Erreur mineure
2	Erreur majeure
3	Erreur grave

5.2.9.17 errorCode

Ce champ indique les codes d'erreur des erreurs survenues au niveau d'un nœud. Les codes d'erreur et leur signification ne sont pas définis ici.

5.2.9.18 vendorCode

Voir 5.2.5.2.

5.2.9.19 DeviceType

Ce champ spécifie le type d'appareil. Pour connaître les types d'appareils, voir le profil de l'appareil.

5.2.9.20 unitTypeName

Ce champ contient la chaîne de caractères du nom du modèle.

5.2.9.21 unitTypeCode

Ce champ contient le code du nom du modèle.

5.2.9.22 reserved5

Ce champ est réservé à un usage futur. La valeur est 0x0000.

5.2.9.23 nodeInfo

Ce champ indique l'état d'un nœud défini par l'utilisateur.

5.2.9.24 dcs

Voir 5.2.2.4.

5.2.10 Transient1-PDU

5.2.10.1 falArHeader

Voir 5.2.1.

5.2.10.2 destinationGroup

Ce champ spécifie le groupe de destination. Chaque bit représente une adresse de groupe. Le Bit 0 indique l'adresse du groupe 1. Le Bit 31 indique l'adresse du groupe 32. Si aucun groupe de destination n'est spécifié, les valeurs de tous les bits sont 0.

5.2.10.3 seqNumber

La signification de chaque bit de ce champ est la suivante:

Bit 0..6	Indique le numéro des fragments de données.
Bit 7	Indique si les données constituent le dernier fragment. La valeur 0 indique qu'il ne s'agit pas de la dernière PDU. La valeur 1 indique qu'il s'agit de la dernière PDU du dernier fragment.

5.2.10.4 dataId

Ce champ contient le numéro d'identification des données transitoires. La plage de valeurs est 0x00..0xFF. Tous les fragments d'une transmission de données transitoires possèdent le même numéro d'identification.

5.2.10.5 wholeDataSize

Ce champ spécifie la quantité de données transitoires en octets.

5.2.10.6 offsetAddr

Ce champ spécifie l'adresse de décalage. Dans la PDU initiale, la valeur est 0x0. Dans les PDU suivantes, la position au sein de l'ensemble des données transitoires est indiquée sous la forme d'une adresse de décalage par rapport à la PDU initiale.

5.2.10.7 dataSize

Ce champ spécifie la taille des données transitoires fragmentées en octets. La plage de valeurs est 0x0000..0x05BA.

5.2.10.8 DataType

Ce champ spécifie les types de données décrits dans le Tableau 16.

Tableau 16 – Data type

Valeur	Description
0	Remise de paramètres
1	Réservé

5.2.10.9 donnée

5.2.10.9.1 Vue d'ensemble

Le champ de données contient les données transitoires. La structure dépend du type de données transitoires. Si la valeur wholeDataSize dépasse 1 466 octets, il s'agit des données transitoires comprises entre le décalage décrit dans offsetAddr et la taille des données décrite dans dataSize. Si les données transitoires comptent moins de 16 octets, le bourrage est rempli avec 0x00.

5.2.10.9.2 Structure de la remise de paramètres

Lorsque le type de données est la remise de paramètres, les données sont sélectionnées parmi CPW, CPWC ou CPWCR. La structure de CPW est décrite dans le Tableau 17, celle de CPWC dans le Tableau 18 et celle de CPWCR dans le Tableau 19. Chaque champ est codé comme LOctetString.

Tableau 17 – CPW

Nom du champ	Taille (octets)	Description
cpfType	4	Type de trame. La valeur est 0.
rcvNodeList	32	Liste des nœuds de réception. Chaque bit représente un nœud. Le bit de poids faible (LSB) de l'octet le plus bas représente le nœud 1 et le bit de poids fort (MSB) de l'octet le plus bas représente le nœud 8. La valeur 1 signifie qu'il s'agit d'un nœud de réception et la valeur 0 signifie que le

Nom du champ	Taille (octets)	Description
		nœud n'est pas un nœud de réception.
commonParamId	12	Voir 5.2.5.8.
cmParam	–	Voir Tableau 20.

Tableau 18 – CPWC

Nom du champ	Taille (octets)	Description
cpfType	4	Type de trame. La valeur est 1.
commonParamId	12	Voir 5.2.5.8.

Tableau 19 – CPWCR

Nom du champ	Taille (octets)	Description
cpfType	4	Type de trame. La valeur est 2.
checkResult	4	Résultat de la vérification des paramètres communs 0: Paramètres communs non reçus 1: Vérification des paramètres communs 2: Vérification OK 3: Échec de la vérification
errorCode	4	Code d'erreur en cas d'échec de la vérification
srcNodeNumber	1	Numéro de nœud source
padding	3	Bourrage

Tableau 20 – cmParam

Nom du champ	Taille (octets)	Description	
Parameter Name (Nom de paramètre)	8	Spécification par l'utilisateur	
Total Size (Taille totale)	2	Taille des données du bloc de début au bloc de fin en octets	
Sum Check Enabled (Contrôle de somme activé)	1	0: Exécution du contrôle de la totalisation 1: Non-exécution du contrôle de la totalisation (par défaut)	
Réservé	1	Réservé	
Sum Check Value (Valeur de somme de contrôle)	4	Valeur de somme de contrôle du bloc de début au bloc de fin	
Create Time (Date et heure de création)	12	Année/mois/jour/heure/minute/seconde de création	
		Taille (octets)	**Description**
		1	Deux derniers chiffres de A.D. (année)
		1	Deux premiers chiffres de A.D. (année)
		2	Mois
		2	Jour
		2	Heure
		2	Minute
		2	Seconde

Nom du champ	Taille (octets)	Description
Begin Marker (Marqueur de début)	4	0x5047532d
Param Area (zone de paramètres)	0..5 924	Voir Tableau 21.
End Marker (Marqueur de fin)	4	0x5047452d

Tableau 21 – Détails de param area

Nom du champ	Taille (octets)	Description
Parameter Availability	2	Bit0: Common Memory Area (Zone de mémoire commune) LB/LW Bit1: Zone additionnelle de mémoire commune LB/LW Bit8: Zone mémoire commune LX/LY Bit9: Zone mémoire commune LX/LY supplémentaire 0 signifie qu'aucun réglage n'est disponible; 1 signifie que des réglages sont disponibles
Version de paramètre commun	1	Version de paramètre commun
Réservé	1	Réservé
LB/LW CM Area Offset	2	Décalage de zone mémoire commune LB/LW
LB/LW CM Additional Area Offset	2	Décalage de zone mémoire commune LB/LW supplémentaire
LX/LY CM 1 Area Offset	2	Décalage de zone mémoire commune 1 LX/LY
LX/LY CM 2 Area Offset	2	Décalage de zone mémoire commune 2 LX/LY
Réservé	24	Réservé
Application parameters	80	Voir Tableau 22
LB/LW CM Area	0 ou 1 452	Voir Tableau 23 Si la valeur Bit0 de Parameter Availability est 0, la taille est 0; si la valeur est 1, la taille est 1 452
LB/LW CM Additional Area	0 ou 1 452	Voir Tableau 23 Si la valeur Bit1 de Parameter Availability est 0, la taille est 0; si la valeur est 1, la taille est 1 452
LX/LY CM 1 Area	0 ou 1 452	Voir Tableau 24 Si la valeur Bit8 de Parameter Availability est 0, la taille est 0; si la valeur est 1, la taille est 1 452
LX/LY CM 2 Area	0 ou 1 452	Voir Tableau 24 Si la valeur Bit9 de Parameter Availability est 0, la taille est 0; si la valeur est 1, la taille est 1 452

Tableau 22 – Détails des paramètres d'application

Nom du champ	Taille (octets)	Description
Control Block	4	0x0
Reserved1	1	Réservé 1

Nom du champ	Taille (octets)	Description
Reserved2	1	Réservé 2
Total Nodes	1	Nombre total de nœuds
Network Type	1	Type de réseau
Acyclic Times (Temps acycliques)	2	Temps de transition
Supervisory Period	2	Temps de surveillance (en ms)
Reserved3	20	Réservé 3
Reserved4	16	Réservé 4
Reserved5	16	Réservé 5
Reserved6	16	Réservé 6

Tableau 23 – Détails de LB/LW CM area et de LB/LW CM additional area

Nom du champ		Taille (octets)	Description
LW CM Head Address		4	Adresse relative de début de LW
LW CM Total Size		4	Nombre total de mots dans la plage de réglage LW
LB CM Head Address		2	Adresse relative de début de LB
LB CM Total Size		2	Taille totale de la plage de réglage LB (en unités de 2 octets)
LB/LW CM Table List		1 440 (12 x 120)	Liste des paramètres LB et LW
	LW CM Head Address Of Node	4	Adresse relative de début de LW dans chaque nœud
	LW CM Size	4	Taille de LW dans chaque nœud (en unités de 2 octets)
	LB CM Head Address Of Node	2	Adresse relative de début de LB dans chaque nœud
	LB CM Size	2	Taille de LB dans chaque nœud (en unités de 2 octets)

Tableau 24 – Détails de LX/LY CM 1 area et de LX/LY CM 2 area

Nom du champ		Taille (octets)	Description
Master Node Number		1	Numéro de nœud maître
Réservé		3	Réservé
LY CM Head Address		2	Adresse relative de début de LY
LY CM Total Size		2	Nombre total de mots dans la plage de réglage LY
LX CM Head Address		2	Adresse relative de début de LX
LX CM Total Size		2	Nombre total de mots dans la plage de réglage LX
LX/LY CM Table List		1 440 (12 x 120)	Liste des paramètres LX/LY
	LY CM Head Address Sent	2	Adresse relative de début de LY envoyée par chaque nœud
	LY CM Size	2	Taille de LY envoyée par chaque nœud (en unités de 2 octets)
	LX CM Head Address Master Received	2	Adresse relative de début de LX reçue par le nœud maître

Nom du champ		Taille (octets)	Description
	LX CM Head Address Received	2	Adresse relative de début de LX reçue par chaque nœud
	LX CM Size	2	Taille de LX reçue par chaque nœud (en unités de 2 octets)
	LY CM Head Address Master Sent	2	Adresse relative de début de LY envoyée par le nœud maître

5.2.10.10 evenPadding

Ce champ n'est utilisé que si le champ de données est un nombre impair d'octets. La valeur est 0x00.

5.2.10.11 dcs

Voir 5.2.2.4.

5.2.11 Dummy-PDU

5.2.11.1 falArHeader

Voir 5.2.1.

5.2.11.2 dummyData

Ce champ contient des données fictives. La taille est comprise entre 28 octets et 1 482 octets. La valeur n'a aucune signification.

5.2.11.3 dcs

Voir 5.2.2.4.

5.2.12 Transient2-PDU

5.2.12.1 falArHeader

Voir 5.2.1.

5.2.12.2 l

Ce champ spécifie la longueur des données de fno à data (en octets).

5.2.12.3 gcnt

Ce champ représente le temps maximal de retransmission sous la forme d'un nombre de portes. La valeur par défaut est 0x07. La valeur est décrémentée au passage de chaque relais.

5.2.12.4 typeSeqF

Ce champ est divisé de la façon suivante:

Bit 7..4 Indique le type. La valeur est 0x00.

Bit 3..0 Représente le numéro de séquence.

5.2.12.5 fno

Ce champ est divisé de la façon suivante:

Bit 7	Indique l'identification du support. Si la valeur est 0, cela signifie que la trame est une trame sans début. Si la valeur est 1, cela signifie que la trame est une trame support.
Bit 6..0	Indique le numéro de trame divisée. La valeur 0 indique que la trame n'a pas été divisée. Les valeurs 1 à 7 indiquent le numéro de trame divisée. Le numéro de trame divisée commence par la même valeur que le numéro de division et est soustrait selon une séquence. Dans une trame divisée en 3, par exemple, l'ordre des numéros de trames est: 3, 2, 1.

5.2.12.6 dt

Ce champ est divisé de la façon suivante:

Bit 7	Indique la priorité. Si la valeur est 0, cela signifie que la priorité est faible. Si la valeur est 1, cela signifie que la priorité est élevée.
Bit 6	Indique la présence de trames de réponse. Si la valeur est 0, cela signifie qu'une trame de réponse est requise. Si la valeur est 1, cela signifie qu'aucune trame de réponse n'est nécessaire.
Bit 5..0	Réservé à un usage futur.

5.2.12.7 da

Ce champ contient le numéro de nœud pour un nœud de relais. Si un nœud de destination se trouve dans le même réseau, ce champ identifie le numéro de nœud de destination.

5.2.12.8 sa

Ce champ contient le numéro de nœud pour un nœud de relais. Si un nœud de destination se trouve dans le même réseau, ce champ identifie le numéro de nœud source.

5.2.12.9 dat

Ce champ contient le type d'application cible. La valeur est fixée à 0x22.

5.2.12.10 sat

Ce champ contient le type d'application source. La valeur est fixée à 0x22.

5.2.12.11 dmf

Ce champ contient l'indicateur de module de destination. Les valeurs sont décrites dans le Tableau 25.

Tableau 25 – Indicateur de module de destination

Nom du module	Description
0	À l'intérieur du module de réseau
1	À l'intérieur du contrôleur

5.2.12.12 smf

Ce champ contient l'indicateur de module source. Les valeurs sont décrites dans le Tableau 25.

5.2.12.13 dna

Ce champ contient le numéro de réseau du nœud de destination.

5.2.12.14 ds

Ce champ contient le numéro de nœud du nœud de destination.

5.2.12.15 did

Ce champ est divisé de la façon suivante:

Bit 10..15	Zone de contrôle (stocke le numéro du nœud de destination)
Bit 0..9	Numéro d'identification de cible. La valeur est fixée à 0x3FF.

5.2.12.16 sna

Ce champ contient le numéro de réseau du nœud source.

5.2.12.17 ss

Ce champ contient le numéro de nœud du nœud source.

5.2.12.18 sid

Ce champ est divisé de la façon suivante:

Bit 10..15	Zone de contrôle (stocke le numéro du nœud source)
Bit 0..9	Numéro d'identification de source. La valeur est fixée à 0x3FF.

5.2.12.19 l1

Ce champ spécifie la longueur des données de ct à data (en octets).

5.2.12.20 ct

Ce champ contient le type de commande tel qu'indiqué dans le Tableau 26.

Tableau 26 – Types de commandes

Type de commande	Description
0x00	Non disponible
0x01	Réservé
0x02	Réservé
0x03	Réservé
0x04	Obtention d'informations sur l'accès à la mémoire
0x05..0x07	Réservé
0x08	Exécution
0x09	Arrêt
0x0A..0x0E	Réservé
0x0F	Réservé
0x10	Lire mémoire
0x11	Réservé

Type de commande	Description
0x12	Écriture dans la mémoire
0x13..0x5F	Réservé
0x60..0x7F	Spécifique à un fournisseur

5.2.12.21 rsv

Ce champ est réservé à un usage futur. La valeur est 0x0.

5.2.12.22 aps

Ce champ est divisé de la façon suivante:

Bit 8..15 Numéro de tâche. La plage de valeurs est 0..255.

Bit 0..7 Numéro d'identification d'application source. La plage de valeurs est 0..255.

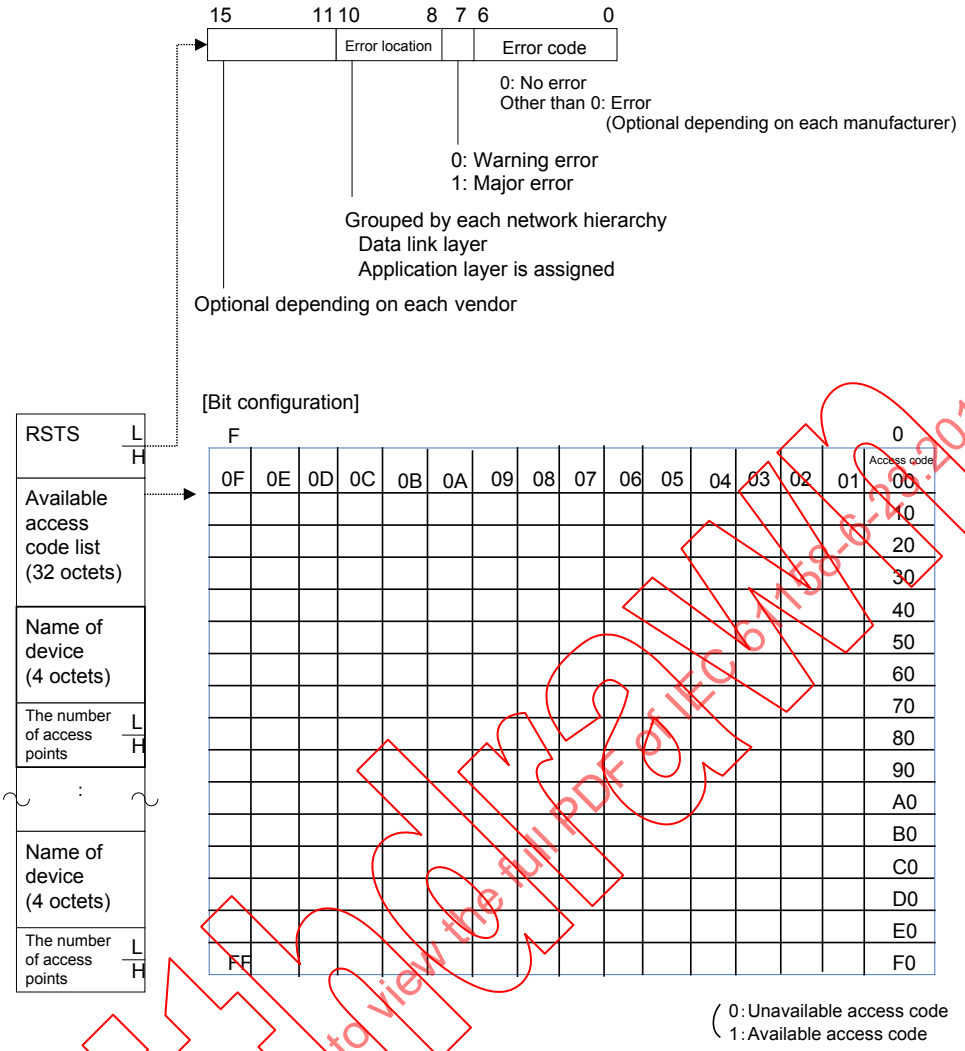
5.2.12.23 Données

5.2.12.23.1 Vue d'ensemble

Ce champ contient les données transitoires. La structure des données transitoires dépend de ct (voir 5.2.12.20).

5.2.12.23.2 Obtention d'informations sur l'accès à la mémoire

En cas de demande d'obtention d'informations sur l'accès à la mémoire, cette zone n'est pas utilisée. La structure de la réponse est indiquée à la Figure 2. La définition de l'attribut est indiquée à la Figure 3. La définition du code d'accès est indiquée à la Figure 4.

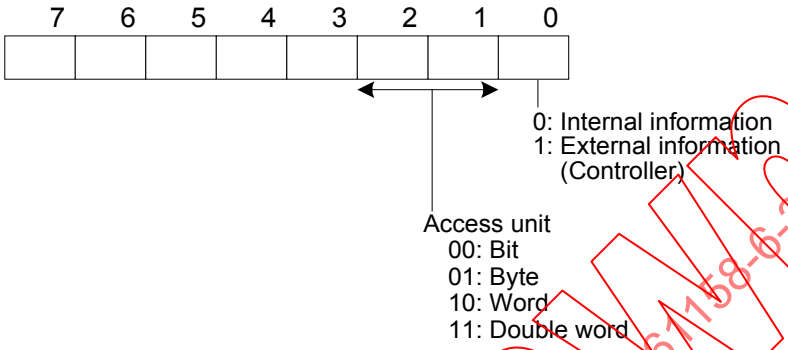


Légende

Anglais	Français
Error location	Position d'erreur
Error code	Code d'erreur
0: No error	0: Pas d'erreur
Other than 0: Error (Optional depending on each manufacturer)	Autre que 0: Erreur (En option en fonction du fabricant)
0: Warning error	0: Erreur d'avertissement
1: Major error	1: Erreur majeure
Grouped by each network hierarchy	Regroupement par hiérarchie de réseau
Data link layer	Couche liaison de données
Application layer is assigned	Couche application attribuée
Optional depending on each vendor	En option en fonction du fournisseur
Bit configuration	Configuration de bit
Access code	Code d'accès
RSTS	RSTS
L/H	Bas/Haut
Available access code list (32 octets)	Liste des codes d'accès disponibles (32 octets)
Name of device (4 octets)	Nom de l'appareil (4 octets)

Anglais	Français
The number of access points	Nombre de points d'accès
0: Unavailable access code	0: Code d'accès indisponible
1: Available access code	1: Code d'accès disponible

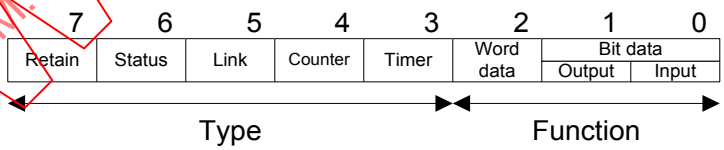
Figure 2 – Structure de réponse à une demande de récupération d'informations sur l'accès à la mémoire



Légende

Anglais	Français
0: Internal information	0: Informations internes
1: External information (controller)	1: Informations externes (contrôleur)
Access unit	Unité d'accès
00: Bit	00: Bit
01: Byte	01: Octet
10: Word	10: Mot
11: Double word	11: Double mot

Figure 3 – Définitions des attributs



Légende

Anglais	Français
Retain	Rétention
Status	État
Link	Liaison
Counter	Compteur
Timer	Temporisateur
Word data	Données de mot
Bit data	Données de bit
Output	Sortie

Anglais	Français
Input	Entrée
Type	Type
Function	Fonction

Figure 4 – Définitions des codes d'accès

Un exemple de code d'accès à la mémoire du module de réseau est décrit dans le Tableau 27. Un exemple de code d'accès à la mémoire du contrôleur est décrit dans le Tableau 28.

Tableau 27 – Codes d'accès à la mémoire du module de réseau

Contenu de la mémoire		Code d'accès
Mémoire tampon	Tampon normalisé	0x00
Tampon d'état	Nœud d'appareil intelligent Tampon à rafraîchissement automatique	0x40
Tampon de liaison	Tampon à accès aléatoire	0x20
Appareil de liaison	Entrée de liaison	0x21
	Sortie de liaison	0x22
	Registre de liaison	0x24
	Relais spécial de liaison	0x63
	Registre spécial de liaison	0x64
NOTE Les champs et les valeurs de propriété spécifiques dépendent du fournisseur.		

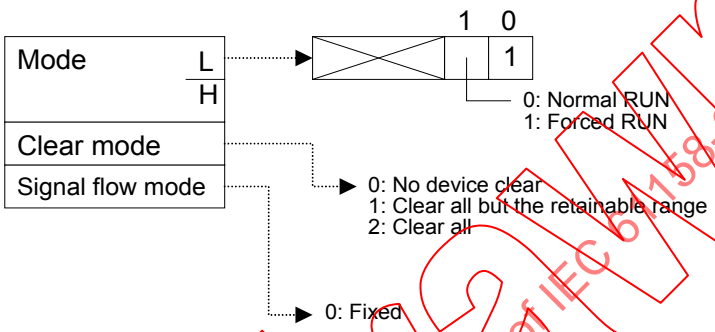
Tableau 28 – Codes d'accès à la mémoire du contrôleur

Contenu de la mémoire	Code d'accès	Type	
		B	W
Relais d'entrée	0x01	X	
Relais de sortie	0x02	X	
Relais spécial	0x43	X	
Registre spécial	0x44		X
Relais interne	0x03	X	
Relais à verrouillage	0x83	X	
Temporisateur (contact)	0x09	X	
Temporisateur (bobine)	0x0A	X	
Temporisateur (valeur courante)	0x0C		X
Temporisateur cumulatif (contact)	0x89	X	
Temporisateur cumulatif (bobine)	0x8A	X	
Temporisateur cumulatif (valeur courante)	0x8C		X
Compteur (contact)	0x11	X	
Compteur (bobine)	0x12	X	
Compteur (valeur courante)	0x14		X
Registre de données	0x04		X
Registre	0x84		X
Relais de liaison	0x23	X	
Registre de liaison	0x24		X

Contenu de la mémoire	Code d'accès	Type	
		B	W
Relais spécial de liaison	0x63	X	
Registre spécial de liaison	0x64		X
NOTE Le nom, le numéro et la plage d'accès à la mémoire de l'appareil dépendent du PLC.			

5.2.12.23.3 Exécution

La structure en cas de demande RUN est indiquée à la Figure 5.

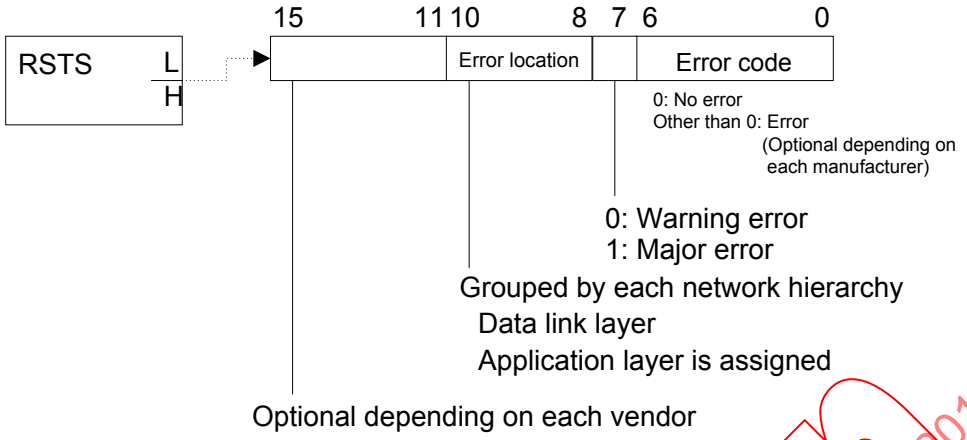


Légende

Anglais	Français
Mode	Mode
Clear mode	Mode Annulation
Signal flow mode	Mode Flux de signaux
Normal RUN	Exécution normale
Forced RUN	Exécution forcée
0: No device clear	0: Aucune annulation d'appareil
1: Clear all but the retainable range	1: Tout annuler sauf la plage à conserver
2: Clear all	2: Annuler tout
0: Fixed	0: Fixe

Figure 5 – Structure en cas de demande RUN

La structure en cas de réponse est indiquée à la Figure 6.



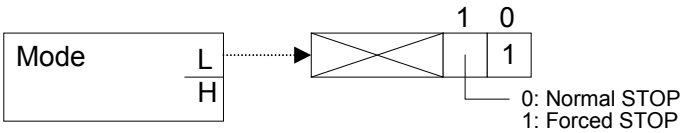
Légende

Anglais	Français
RSTS	RSTS
L/H	L/H
Error location	Emplacement d'erreur
Error code	Code d'erreur
0: No error	0: Pas d'erreur
Other than 0: Error (Optional depending on each manufacturer)	Autre que 0: Erreur (en option en fonction du fabricant)
0: Warning error	0: Erreur d'avertissement
1: Major error	1: Erreur majeure
Grouped by each network hierarchy	Groupement par hiérarchie de réseau
Data link layer	Couche liaison de données
Application layer is assigned	Couche application affectée
Optional depending on each vendor	En option en fonction de chaque fournisseur

Figure 6 – Structure en cas de réponse à une demande RUN

5.2.12.23.4 Stop (Arrêt)

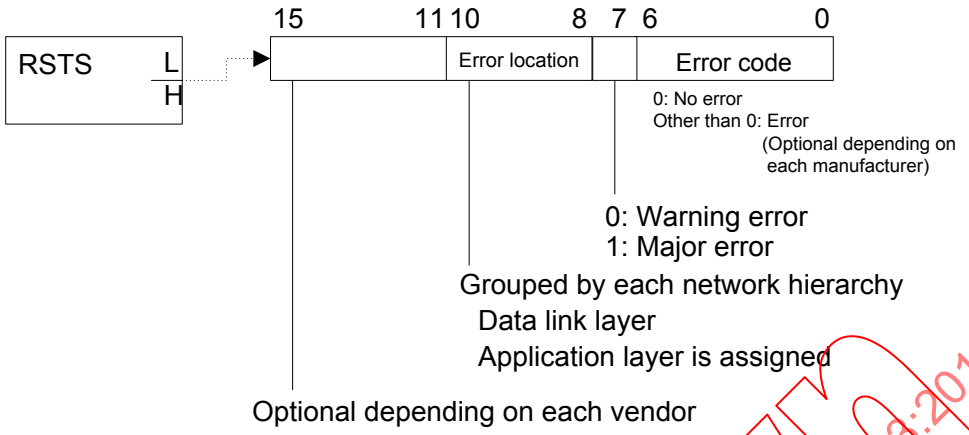
La structure en cas de demande STOP est indiquée à la Figure 7. La structure en cas de réponse est indiquée à la Figure 8.



Légende

Anglais	Français
Mode	Mode
L/H	L/H
0: Normal STOP	0: ARRÊT normal
1: Forced STOP	1: ARRÊT forcé

Figure 7 – Structure en cas de demande STOP



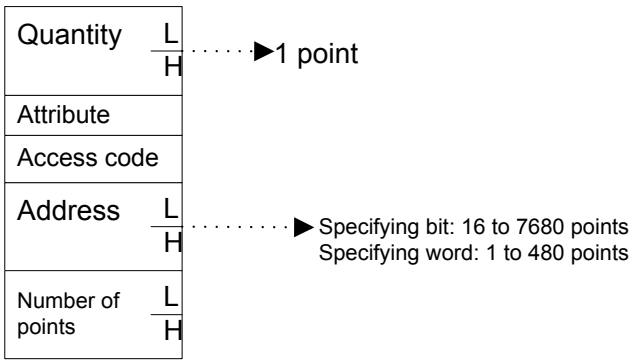
Légende

Anglais	Français
RSTS	RSTS
L/H	L/H
Error location	Emplacement d'erreur
Error code	Code d'erreur
0: No error	0: Pas d'erreur
Other than 0: Error (Optional depending on each manufacturer)	Autre que 0: Erreur (en option en fonction du fabricant)
0: Warning error	0: Erreur d'avertissement
1: Major error	1: Erreur majeure
Grouped by each network hierarchy	Groupement par hiérarchie de réseau
Data link layer	Couche liaison de données
Application layer is assigned	Couche application affectée
Optional depending on each vendor	En option en fonction de chaque fournisseur

Figure 8 – Structure en cas de réponse à une demande STOP

5.2.12.23.5 Read memory (Lire mémoire)

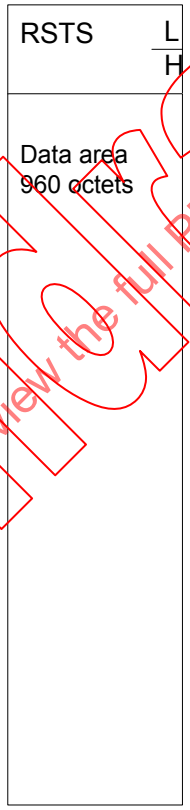
La structure en cas de demande de lecture de mémoire en lot est indiquée à la Figure 9. La structure en cas de réponse est indiquée à la Figure 10.



Légende

Anglais	Français
Quantity	Quantité
L/H	L/H
1 point	1 point
Attribute	Attribut
Access code	Code d'accès
Address	Adresse
Number of points	Nombre de points
Specifying bit: 16 to 7680 points	Spécification de bit: 16 points à 7 680 points
Specifying word 1 to 480 points	Spécification de mot: 1 point à 480 points

Figure 9 – Structure en cas de demande de lecture de mémoire par lots

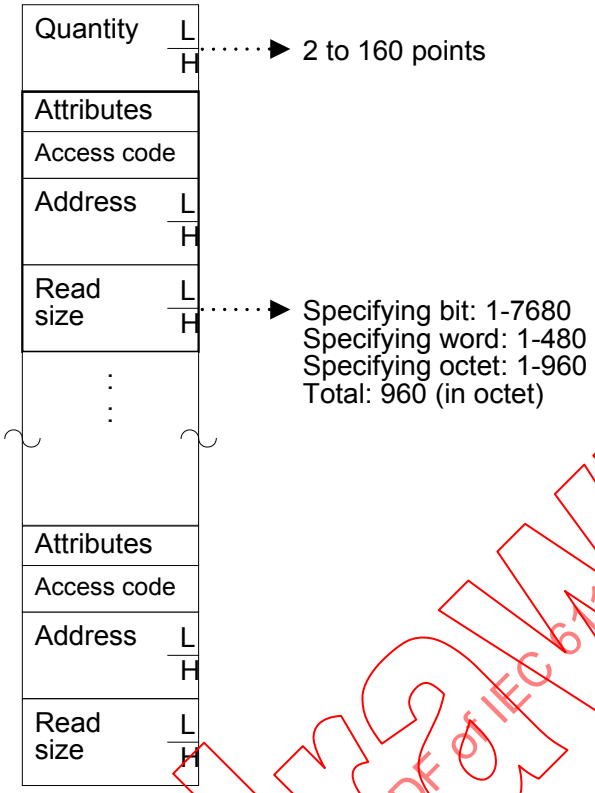


Légende

Anglais	Français
RSTS	RSTS
L/H	L/H
Data area	Zone de données
960 octets	960 octets

Figure 10 – Structure en cas de réponse à une demande de lecture de mémoire par lots

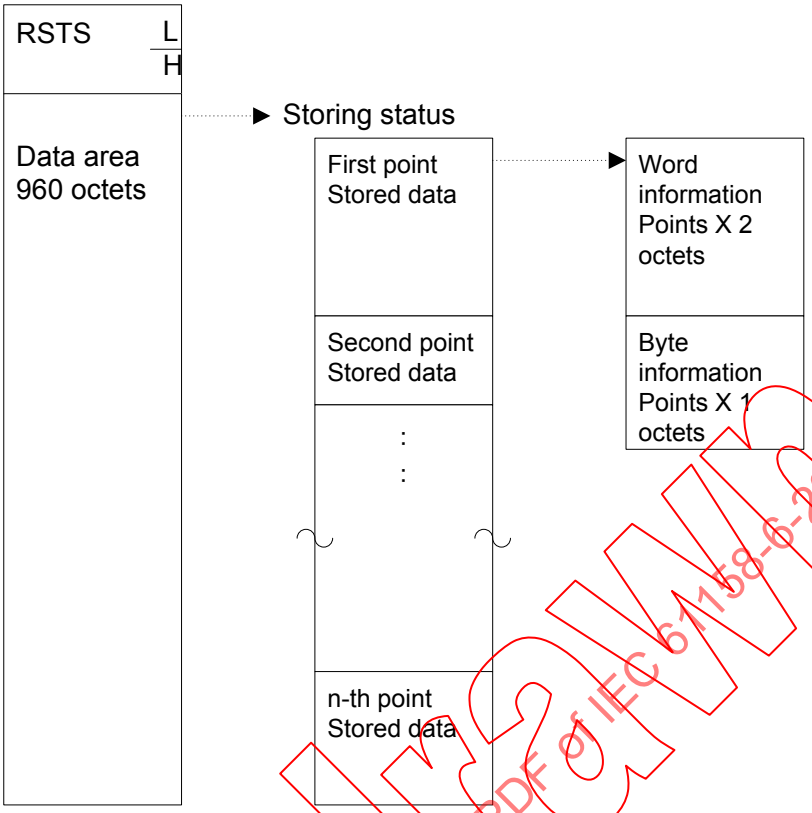
La structure en cas de demande de lecture de mémoire aléatoire est indiquée à la Figure 11. La structure en cas de réponse est indiquée à la Figure 12.



Légende

Anglais	Français
Quantity	Quantité
L/H	L/H
2 to 160 points	2 points à 160 points
Attributes	Attributs
Access code	Code d'accès
Address	Adresse
Read size	Taille de lecture
Specifying bit: 1-7680	Spécification de bit: 1-7 680
Specifying word: 1-480	Spécification de mot: 1-480
Specifying octet: 1-960	Spécification d'octet: 1-960
Total: 960 (in octet)	Total: 960 (en octets)

Figure 11 – Structure en cas de demande de lecture de mémoire aléatoire



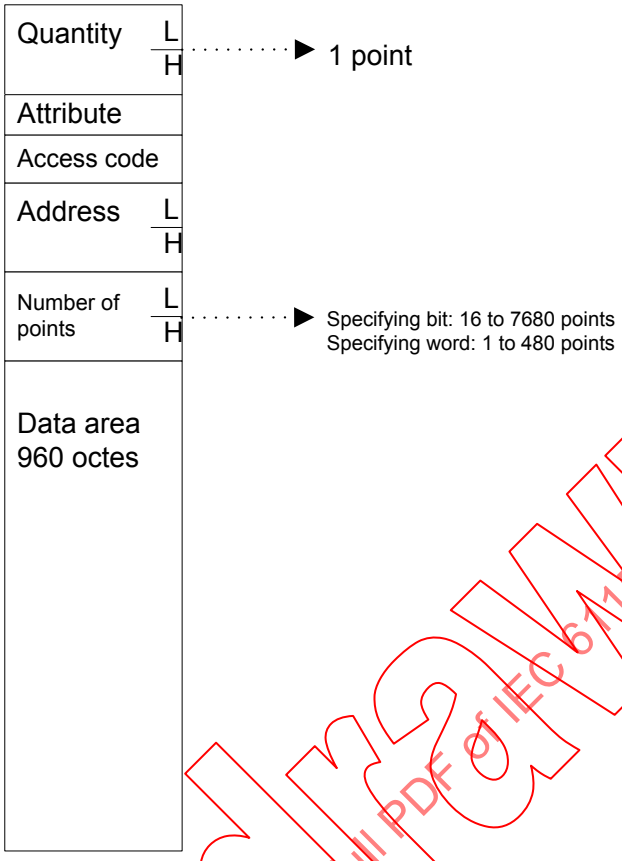
Légende

Anglais	Français
RSTS	RSTS
L/H	L/H
Data area	Zone de données
960 octets	960 octets
Storing status	État de stockage
First point	1er point
Stored data	Données stockées
Second point	2ème point
n-th point	nème point
Word information Points X 2 octets	Points d'informations de mot X 2 octets
Byte information Points X 1 octets	Points d'informations d'octet X 1 octet

Figure 12 – Structure en cas de réponse à une demande de lecture de mémoire aléatoire

5.2.12.23.6 Write memory (Écriture dans la mémoire)

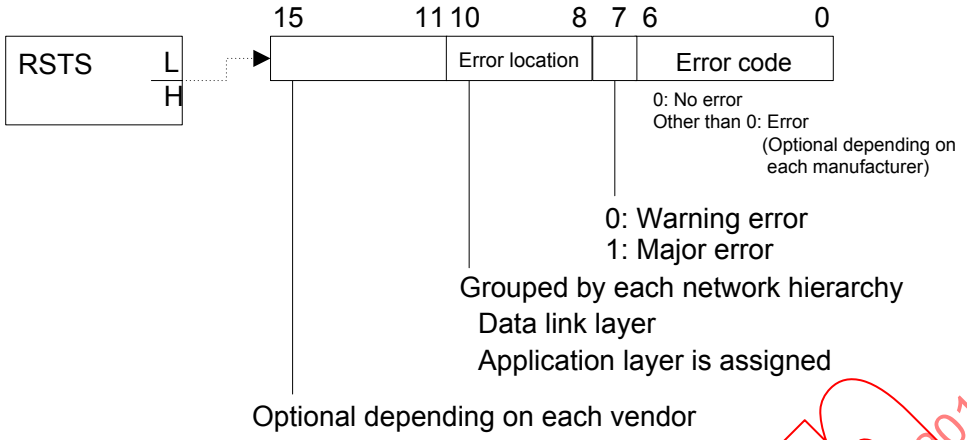
La structure en cas de demande d'écriture dans la mémoire en lot est indiquée à la Figure 13. La structure en cas de réponse est indiquée à la Figure 14.



Légende

Anglais	Français
Quantity	Quantité
L/H	L/H
1 point	1 point
Attribute	Attribut
Access code	Code d'accès
Address	Adresse
Number of points	Nombre de points
Specifying bit: 16 to 7680 points	Spécification de bit: 16 points à 7 680 points
Specifying word 1 to 480 points	Spécification de mot: 1 point à 480 points
Data area 960 octets	Zone de données 960 octets

Figure 13 – Structure en cas de demande d'écriture mémoire par lots

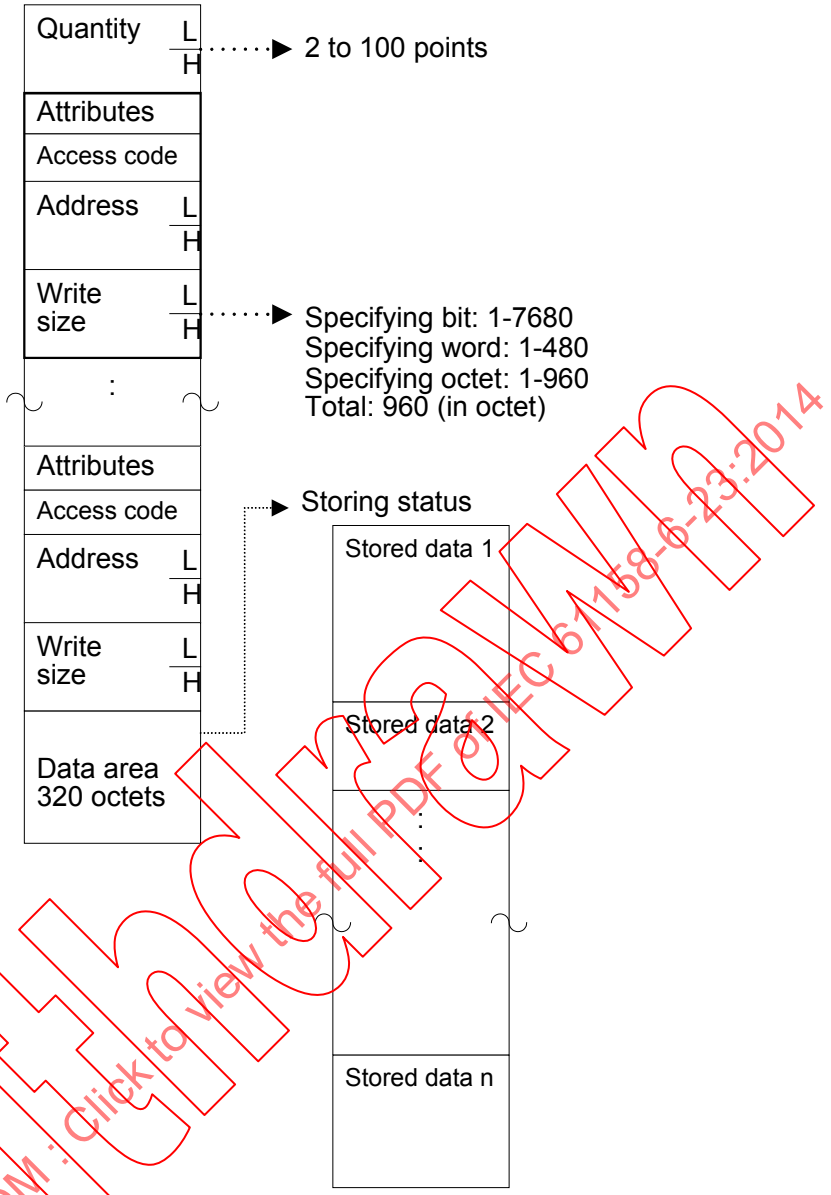


Légende

Anglais	Français
RSTS	RSTS
L/H	L/H
Error location	Emplacement d'erreur
Error code	Code d'erreur
0: No error	0: Pas d'erreur
Other than 0: Error (Optional depending on each manufacturer)	Autre que 0: Erreur (en option en fonction du fabricant)
0: Warning error	0: Erreur d'avertissement
1: Major error	1: Erreur majeure
Grouped by each network hierarchy	Groupement par hiérarchie de réseau
Data link layer	Couche liaison de données
Application layer is assigned	Couche application affectée
Optional depending on each vendor	En option en fonction de chaque fournisseur

Figure 14 – Structure en cas de réponse à une demande d'écriture mémoire par lots

La structure en cas de demande d'écriture dans la mémoire aléatoire est indiquée à la Figure 15. La structure en cas de réponse est indiquée à la Figure 16.

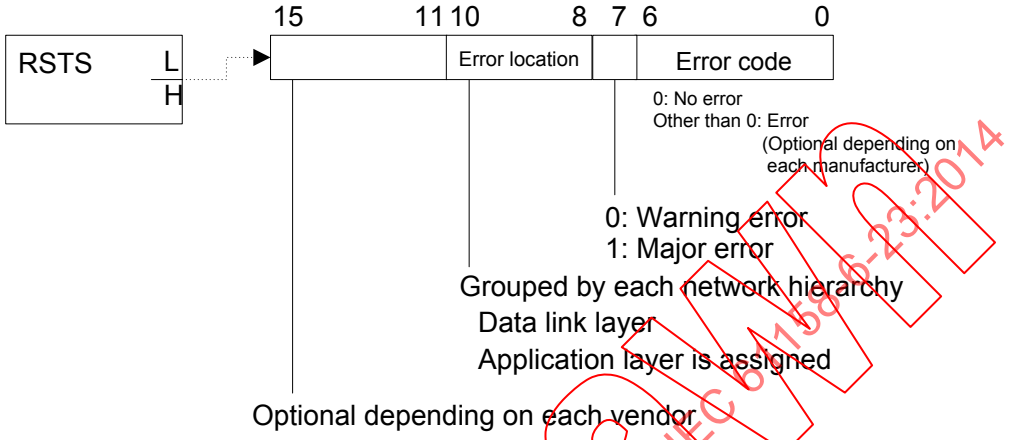


Légende

Anglais	Français
Quantity	Quantité
L/H	L/H
2 to 100 points	2 points à 100 points
Attributes	Attributs
Access code	Code d'accès
Address	Adresse
Write size	Taille d'écriture
Specifying bit: 1-7680	Spécification de bit: 1-7 680
Specifying word: 1-480	Spécification de mot: 1-480
Specifying octet: 1-960	Spécification d'octet: 1-960
Total: 960 (in octet)	Total: 960 (en octets)
Data area 320 octets	Zone de données 320 octets
Storing status	État de stockage
Stored data 1	Données stockées 1

Anglais	Français
Stored data 2	Données stockées 2
Stored data n	Données stockées n

Figure 15 – Structure en cas de demande d'écriture mémoire aléatoire



Légende

Anglais	Français
RSTS	RSTS
L/H	L/H
Error location	Emplacement d'erreur
Error code	Code d'erreur
0: No error	0: Pas d'erreur
Other than 0: Error (Optional depending on each manufacturer)	Autre que 0: Erreur (en option en fonction du fabricant)
0: Warning error	0: Erreur d'avertissement
1: Major error	1: Erreur majeure
Grouped by each network hierarchy	Groupement par hiérarchie de réseau
Data link layer	Couche liaison de données
Application layer is assigned	Couche application affectée
Optional depending on each vendor	En option en fonction de chaque fournisseur

Figure 16 – Structure en cas de réponse à une demande d'écriture mémoire aléatoire

5.2.12.24 evenPadding

Ce champ n'est utilisé que si le champ de données contient un nombre impair d'octets. La valeur est 0x00.

5.2.12.25 dcs

Voir 5.2.2.4.

5.2.13 NTNTest-PDU

5.2.13.1 falArHeader

Voir 5.2.1.

5.2.13.2 ntnTestData

Ce champ contient les données d'essai fictives. La taille est comprise entre 28 octets et 1 482 octets. La valeur n'a aucune signification.

5.2.13.3 dcs

Voir 5.2.2.4.

5.2.14 CyclicDataW-PDU

5.2.14.1 falArHeader

Voir 5.2.1.

5.2.14.2 seqNumber

Voir 5.2.10.3.

5.2.14.3 byteValidity

Ce champ spécifie si les 4 premiers octets et les 4 derniers octets de wData sont reflétés sur la mémoire partagée. Les définitions des bits sont indiquées dans le Tableau 29. Pour chaque bit, 1 = valide et 0 = invalide.

Tableau 29 – byteValidity

Bit	Octet de wData
0	premier octet
1	deuxième octet
2	troisième octet
3	quatrième octet
4	quatrième octet à partir de la fin
5	troisième octet à partir de la fin
6	deuxième octet à partir de la fin
7	dernier octet

5.2.14.4 dataSize

Ce champ spécifie la taille des données cycliques. La plage pour cette valeur est 0x0..0x16F. La taille est spécifiée par le nombre de groupes de 4 octets; les 4 premiers octets et les 4 derniers octets avec byteValidity sont également inclus dans la taille.

5.2.14.5 offsetAddr

Ce champ spécifie l'adresse de décalage de LW en octets. La plage pour cette valeur est 0x0..0x3FFFC; seuls les multiples de 4 sont autorisés. En d'autres termes, la valeur du Bit 0..1 est 0.

5.2.14.6 exSeqNumber

Ce champ identifie le numéro de séquence associé aux données fragmentées. La plage pour cette valeur est 0x0..0xFFFF. Lorsque ce champ est utilisé, la valeur seqNumber est 0x7F.

5.2.14.7 Réservés

Ce champ est réservé à un usage futur. La valeur est 0x0.

5.2.14.8 WData

Ce champ contient les données LW. Si la valeur de données contient moins de 16 octets, le bourrage est de 0x00.

5.2.14.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de wData est impair. La valeur est 0x00.

5.2.14.10 dcs

Voir 5.2.2.4.

5.2.15 CyclicDataB-PDU

5.2.15.1 falArHeader

Voir 5.2.1.

5.2.15.2 seqNumber

Voir 5.2.14.2.

5.2.15.3 byteValidity

Voir 5.2.14.3.

5.2.15.4 dataSize

Voir 5.2.14.4.

5.2.15.5 offsetAddr

Ce champ spécifie l'adresse de décalage de LB en octets. La plage pour cette valeur est 0x0..0x3FFFC; seuls les multiples de 4 sont autorisés. En d'autres termes, la valeur du Bit 0..1 est 0.

5.2.15.6 reserved1

Ce champ est réservé à un usage futur. La valeur est 0x0.

5.2.15.7 reserved2

Ce champ est réservé à un usage futur. La valeur est 0x0.

5.2.15.8 bData

Ce champ contient les données LB. Si les données comptent moins de 16 octets, le bourrage est de 0x00.

5.2.15.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de bData est impair. La valeur est 0x00.

5.2.15.10 dcs

Voir 5.2.2.4.

5.2.16 CyclicDataOut1-PDU

5.2.16.1 falArHeader

Voir 5.2.1.

5.2.16.2 seqNumber

Voir 5.2.14.2.

5.2.16.3 byteValidity

Ce champ n'est pas utilisé. La valeur est 0x00.

5.2.16.4 dataSize

Voir 5.2.14.4.

5.2.16.5 offsetAddr

Ce champ n'est pas utilisé. La valeur est 0x00.

5.2.16.6 reserved1

Ce champ est réservé. La valeur est 0x0.

5.2.16.7 reserved2

Ce champ est réservé. La valeur est 0x0.

5.2.16.8 out1Data

Ce champ contient les données LY1 envoyées par le nœud maître à tous les nœuds de réception. Si la valeur des données contient moins de 16 octets, le bourrage est de 0x00.

5.2.16.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de out1Data est impair. La valeur est 0x00.

5.2.16.10 dcs

Voir 5.2.2.4.

5.2.17 CyclicDataOut2-PDU

5.2.17.1 falArHeader

Voir 5.2.1.

5.2.17.2 seqNumber

Voir 5.2.14.2.

5.2.17.3 byteValidity

Ce champ n'est pas utilisé. La valeur est 0x00.

5.2.17.4 dataSize

Voir 5.2.14.4.

5.2.17.5 offsetAddr

Ce champ n'est pas utilisé. La valeur est 0x00.

5.2.17.6 reserved1

Ce champ est réservé. La valeur est 0x0.

5.2.17.7 reserved2

Ce champ est réservé. La valeur est 0x0.

5.2.17.8 out2Data

Ce champ contient les données LY2 envoyées par le nœud maître à tous les nœuds de réception. Si la valeur de ces données contient moins de 16 octets, le bourrage est de 0x00.

5.2.17.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de out2Data est impair. La valeur est 0x00.

5.2.17.10 dcs

Voir 5.2.2.4.

5.2.18 CyclicDataIn1-PDU

5.2.18.1 falArHeader

Voir 5.2.1.

5.2.18.2 seqNumber

Voir 5.2.14.2.

5.2.18.3 byteValidity

Voir 5.2.14.3.

5.2.18.4 dataSize

Voir 5.2.14.4.

5.2.18.5 offsetAddr

Ce champ spécifie l'adresse de décalage de LX en octets. La plage pour cette valeur est 0x0..0x3FFFC; seuls les multiples de 4 sont autorisés. En d'autres termes, la valeur du Bit 0..1 est 0.

5.2.18.6 reserved1

Ce champ est réservé. La valeur est 0x0.

5.2.18.7 reserved2

Ce champ est réservé. La valeur est 0x0.

5.2.18.8 in1Data

Ce champ contient les données LX1 envoyées par chaque nœud d'émission au nœud maître. Si la valeur des données contient moins de 16 octets, le bourrage est de 0x00.

5.2.18.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de in1Data est impair. La valeur est 0x00.

5.2.18.10 dcs

Voir 5.2.2.4.

5.2.19 CyclicDataIn2-PDU

5.2.19.1 falArHeader

Voir 5.2.1.

5.2.19.2 seqNumber

Voir 5.2.14.2.

5.2.19.3 byteValidity

Voir 5.2.14.3.

5.2.19.4 dataSize

Voir 5.2.14.4.

5.2.19.5 offsetAddr

Ce champ spécifie l'adresse de décalage de LX en octets. La plage pour cette valeur est 0x0..0x3FFFC; seuls les multiples de 4 sont autorisés. En d'autres termes, la valeur du Bit 0..1 est 0.

5.2.19.6 reserved1

Ce champ est réservé. Il est paramétré sur 0x0.

5.2.19.7 reserved2

Ce champ est réservé. Il est paramétré sur 0x0.

5.2.19.8 in2Data

Ce champ contient les données LX2 envoyées par chaque nœud d'émission au nœud maître. Si la valeur des données contient moins de 16 octets, le bourrage est de 0x00.

5.2.19.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de in2Data est impair. La valeur est 0x00.

5.2.19.10 dcs

Voir 5.2.2.4.

5.3 Codage des éléments FAL-PDU de type F

5.3.1 FALARHeader

5.3.1.1 arFType

Ce champ indique les types de PDU décrits dans le Tableau 30.

Tableau 30 – afFType

Valeur	Description
0x00..0x06	Utilisation avec le type C
0x07..0x09	Réservé à un usage futur
0x10	persuasion-PDU
0x11	testData-PDU
0x12	testDataAck-PDU
0x13	setup-PDU
0x14	setupAck-PDU
0x15	token-PDU
0x16..0x1B	Réservé à un usage futur
0x1C	timer-PDU
0x1D..0x1F	Réservé à un usage futur
0x20	myStatus-PDU
0x21	Réservé à un usage futur
0x22	transient1-PDU
0x23	transientAck-PDU
0x24	Utilisation avec le type C
0x25	transient2-PDU
0x26..0x27	Réservé à un usage futur
0x28	paramCheck-PDU
0x29	parameter-PDU
0x2A..0x3F	Réservé à un usage futur
0x40	measure-PDU
0x41	measureAck-PDU
0x42	offset-PDU
0x43	update-PDU
0x44..0x7F	Réservé à un usage futur
0x80..0x81	Utilisation avec le type C
0x82	cyclicDataRWw-PDU
0x83	cyclicDataRY-PDU
0x84	cyclicDataRWr-PDU
0x85	cyclicDataRX-PDU
0x86..0x8B	Réservé à un usage futur
0x8C..0x8F	Utilisation avec le type C
0x90..0xFF	Réservé à un usage futur

5.3.1.2 DataType

Ce champ indique le type de données; les valeurs sont classées par arFtype, tel que décrit dans le Tableau 31.

Tableau 31 – dataType

arFtype	Valeur	Description
0x10..0x15	0x00	Réservé à un usage futur
	0x01	Contrôle de transmission
	0x02..0xFF	Réservé à un usage futur
0x1C	0x00	Réservé à un usage futur
	0x01	Transmission transitoire (spécifique au type F)
	0x02..0xFF	Réservé à un usage futur
0x20	0x00..0x01	Réservé à un usage futur
	0x02	Contrôle de transmission (diagnostic)
	0x03..0xFF	Réservé à un usage futur
0x22	0x00..0x06	Réservé à un usage futur
	0x07	Transmission transitoire (spécifique au type F)
	0x08	Transmission transitoire (spécifique à l'entraînement)
	0x09	Transmission transitoire (spécifique à la sécurité)
	0x10..0xFF	Réservé à un usage futur
0x23	0x00..0xFF	Réponse à la transmission transitoire
0x25	0x00..0x03	Réservé à un usage futur
	0x04	Transmission transitoire (compatible CP8/1, CP8/2)
	0x05..0xFF	Réservé à un usage futur
0x28-0x29	0x00..0x02	Réservé à un usage futur
	0x03	Réglage de transmission cyclique
	0x04..0xFF	Réservé à un usage futur
0x82-0x85	0x00	Transmission cyclique
	0x01..0xFF	Réservé à un usage futur

5.3.1.3 varField

5.3.1.3.1 Vue d'ensemble

Ce champ contient les champs énumérés dans le Tableau 32, classés par arFtype comme décrit

Tableau 32 – varField

arFtype	Champ utilisé
persuasion-PDU testDataAck-PDU	persPriority nodeType
testData-PDU setupAck-PDU timer-PDU	reserved1
setup-PDU	nodeId

arFtype	Champ utilisé
token-PDU cyclicDataRWw-PDU cyclicDataRY-PDU cyclicDataRWr-PDU cyclicDataRX-PDU measure-PDU measureAck-PDU offset-PDU update-PDU	reserved2
myStatus-PDU	nodeId syncFlag nodeType
transient1-PDU transientAck-PDU transient2-PDU paramCheck-PDU parameter-PDU	nodeId connectionInfo reserved4

5.3.1.3.2 persPriority

Ce champ indique le niveau de priorité lorsque le gestionnaire de contrôle de transmission est sélectionné. La plage pour cette valeur est 0x0..0xFFFFFF. La valeur 0x0 indique que le nœud n'est pas le gestionnaire de contrôle de transmission.

5.3.1.3.3 nodeType

Ce champ indique le type de nœud. Les valeurs sont spécifiées dans le Tableau 33.

Tableau 33 – nodeType

Valeur	Description
0x00..0x2F	Utilisation avec le type C
0x30	Station maître
0x31	Réservé à un usage futur
0x32	Station locale
0x33	Station à appareil intelligent
0x34	Station à appareil distant
0x35	Station d'E/S déportées
0x36..0xFF	Réservé à un usage futur

5.3.1.3.4 reserved1

Réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.1.3.5 nodeId

Ce champ contient l'identificateur de nœud. Le gestionnaire de contrôle de transmission détermine les identificateurs des nœuds autres que le gestionnaire de contrôle de

transmission et définit les valeurs à l'aide de setup-PDU. La plage de valeurs est 0..255. Si un numéro de nœud n'a pas été défini, la valeur est 255.

5.3.1.3.6 reserved2

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.1.3.7 syncFlag

La signification de chaque bit est la suivante:

Bits 7..1	Réservé à un usage futur. La valeur est 0.
Bit 0	Définit l'indicateur de synchronisation. La valeur est 1 si la trame fournit des informations relatives à la séquence de synchronisation; dans le cas contraire, la valeur est 0.

5.3.1.3.8 connectionInfo

Ce champ identifie la valeur f-transientData-PDU à transmettre si un jeton est maintenu. Si un jeton est maintenu et si une PDU de transmission non cyclique en double présentant les mêmes valeurs destaddr et connectionInfo est reçue, le nœud de réception procède au rejet.

5.3.1.3.9 reserved4

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.1.3.10 srcNodeNumber

Ce champ contient le numéro d'identification du nœud source.

5.3.1.3.11 protocolVerType

La signification de chaque bit est la suivante:

Bits 7..4:	Version du protocole. La valeur est 0.
Bits 3..0:	Type de protocole. Les valeurs utilisées sont conformes au Tableau 34.

Tableau 34 – ProtocolVerType

Valeur	Description
0x0	type C
0x1	type F
0x2..0xF	Réservé à un usage futur

5.3.1.3.12 Réservés

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.1.3.13 hec

Ce champ contient un code de détection d'erreur qui cible l'élément DestAddr de la DLPDU. Les définitions DLPDU sont les suivantes.

```
DLPDU::= SEQUENCE {
    preamble                    Preamble,
```

```
                sfd                     SFD,
                destaddr                DestAddr,
                srcaddr                 SrcAddr,
                lt                      LT,
                DLSDU                   FAL-PDU,
                FCS                     FCS
            }

Preamble::= OctetString (SIZE(7))
SFD::= OctetString (SIZE(1))
DestAddr::= MACAddress
SrcAddr::= MACAddress
LT::= Unsigned16
FCS::= OctetString (SIZE(4))
```

Le polynôme générateur est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x1+1$.

5.3.2 Persuasion-PDU

5.3.2.1 falArHeader

Voir 5.3.1.

5.3.2.2 reserved1

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.2.3 myPorts

Ce champ indique le nombre de ports de communication physiques détenus par le nœud.

5.3.2.4 vendorCode

Ce champ contient le code du fournisseur.

5.3.2.5 modelCode

Ce champ contient le code du modèle spécifique au fournisseur.

5.3.2.6 reserved2

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.2.7 dcs

Ce champ contient le code de détection d'erreur qui cible l'élément destaddr de la DLPDU dans le champ de dcs précédent. Pour connaître les définitions des DLPDU, voir 5.3.1.3.13.

Le polynôme générateur est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+1$.

5.3.3 TestData-PDU

5.3.3.1 falArHeader

Voir 5.3.1.

5.3.3.2 tmMacAddr

Ce champ contient l'adresse MAC du gestionnaire de contrôle de transmission.

5.3.3.3 srcPort

Ce champ contient le numéro de port source de transmission de testData-PDU. La valeur 0 est invalide.

5.3.3.4 Réservés

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.3.5 dcs

Voir 5.3.2.7.

5.3.4 TestDataAck-PDU

5.3.4.1 falArHeader

Voir 5.3.1.

5.3.4.2 tdSrcMacAddr

Ce champ contient l'adresse MAC du nœud source de transmission de testData-PDU inclus comme valeur srcaddr dans la DLPDU du testData-PDU reçu. La cible est testData-PDU qui servait d'impulsion pour la transmission de testDataAck-PDU.

5.3.4.3 tdSrcPort

Ce champ contient le numéro de port source du nœud source de transmission de testData-PDU inclus comme valeur srcPort dans le testData-PDU reçu.

5.3.4.4 tdRcvPort

Ce champ contient le numéro de port de son propre nœud qui a reçu le testDataAck-PDU.

5.3.4.5 reserved1

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.4.6 myPorts

Voir 5.3.2.3.

5.3.4.7 tokenKeepTime

Ce champ spécifie la durée maximale pendant laquelle le nœud maintient un jeton après le début du passage de jeton. Le nœud avertit le gestionnaire de contrôle de transmission de la durée de maintien du jeton par l'intermédiaire de ce champ.

La signification de chaque bit est la suivante:

Bit 15:	Réservé à un usage futur. La valeur est 0.
Bits 14..0:	Indique la durée définie. L'unité est 1 µs. La plage de valeurs est 1..32 767.

NOTE Les points de départ et de fin de la durée de maintien du jeton sont identiques à ceux de token-PDU. Si le point de départ est le début de la réception du token-PDU, le point final est le début de la transmission de token-

PDU. Si le point de départ est défini comme la fin de la réception de token-PDU, le point final est défini comme la fin de la transmission de token-PDU.

5.3.4.8 reserved2

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.4.9 myConnectStatus

Ce champ indique l'état de chaque port. Le champ est utilisé pour indiquer l'état des 24 ports, au maximum, à l'aide de quatre bits pour chaque port. Le champ indique l'état du port 1 avec les bits 3..0 et l'état du port 2 avec les bits 7..4 du premier octet. Le champ indique l'état du port 3 avec les bits 3..0 et l'état du port 4 avec les bits 7..4 du deuxième octet. De la même façon, le champ utilise ensuite jusqu'à 12 octets pour indiquer les états des 24 ports. La signification de chaque bit d'un octet est la suivante:

La signification de chaque bit est la suivante:

Bits 7..6: Réservé à un usage futur. La valeur est 0.

Bits 5..4: Les valeurs sont conformes au Tableau 35.

Bits 3..2: Réservé à un usage futur. La valeur est 0.

Bits 1..0: Les valeurs sont conformes au Tableau 35.

Tableau 35 – État de liaison

Valeur	Description
00b	Liaison descendante
01b	Liaison établie (1 Gbit/s)
10b	Réservé à un usage futur
11b	Réservé à un usage futur

5.3.4.10 dcs

Voir 5.3.2.7.

5.3.5 Setup-PDU

5.3.5.1 falArHeader

Voir 5.3.1.

5.3.5.2 tokenDstMacAddr

Ce champ spécifie l'adresse MAC du nœud suivant qui maintiendra le jeton sur le chemin de passage du jeton. Le gestionnaire de contrôle de transmission transmet le setup-PDU qui a défini l'adresse MAC de la destination de transmission du token-PDU de ce champ au nœud qui sert d'origine de transmission du token-PDU. L'adresse MAC spécifiée dans ce champ est maintenue dans le nœud qui a reçu le setup-PDU.

Une fois le passage du jeton commencé, le gestionnaire de contrôle de transmission devient le premier nœud à détenir le jeton. Le nœud qui détient le jeton transmet ses propres données, puis transmet le token-PDU avec l'adresse MAC spécifiée dans ce champ dans le tokenDstMacAddr. Le nœud qui reçoit le token-PDU détermine si sa propre adresse MAC de nœud et la valeur de tokenDstMacAddr du token-PDU correspondent. Si tel est le cas, il évalue si le token-PDU est adressé à sa propre adresse de nœud. Un nœud qui reçoit un token-PDU qui lui est adressé devient le nœud qui détient le jeton.

5.3.5.3 reserved1

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.5.4 leaveTimerValue

Ce champ spécifie la valeur de réglage de LeaveTimer. Le gestionnaire de contrôle de transmission entre la valeur LeaveTimer à régler dans un nœud autre que le gestionnaire de contrôle de transmission dans ce champ du setup-PDU à transmettre. La valeur de ce champ est définie dans le nœud qui reçoit le setup-PDU comme valeur LeaveTimer. LeaveTimer est utilisé pour mesurer la durée de la détection de la déconnexion de son propre nœud.

La signification des bits de ce champ est la suivante:

Bit 15: Réservé à un usage futur. La valeur est 0.

Bits 14..0: Valeur LeaveTimer. La plage de valeurs est 1..32 767. L'unité est 400 µs.

5.3.5.5 portUsage

Ce champ indique le réglage d'activation/de désactivation de la fonction transmission/réception du port associé au nœud. La désactivation d'un port signifie que le port n'utilise pas le service DL. Les valeurs utilisées sont conformes au Tableau 36.

Dans un nœud avec deux ports qui reçoivent des valeurs de réglage, l'état activé/désactivé des ports du nœud est déterminé conformément au Tableau 36. Un nœud disposant de trois ports ou plus ayant reçu des valeurs de réglage active tous les ports sans utiliser les valeurs de réglage spécifiées.

Tableau 36 – Spécification d'activation/de désactivation de port

Valeur	Description
0x00	Activation des deux ports
0x01	Activation du port 1 uniquement et désactivation de tous les autres ports.
0x02	Activation du port 2 uniquement et désactivation de tous les autres ports.

5.3.5.6 reserved2

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.5.7 netBehaviour

5.3.5.7.1 multipleTransmit

Ce champ spécifie le nombre de fois où le nœud qui détient le jeton procède à la transmission des FAL-PDU autres que le token-PDU. La plage de valeurs est 1..255. 0 n'est pas valide.

Si ce champ contient la valeur 1, le nœud qui détient le jeton transmet myStatus-PDU, f-cyclicData-PDU et f-transientData-PDU, puis transmet enfin le token-PDU. Par conséquent, le nœud ne détient plus le jeton. Dans le cas contraire, le nœud ignore f-transientData-PDU. Le nœud qui détient le jeton transmet myStatus-PDU, f-cyclicData-PDU et f-transientData-PDU le nombre de fois spécifié dans ce champ, après quoi il transmet le token-PDU.

NOTE Par exemple, si une station à appareil intelligent, qui spécifie 2 dans ce champ, devient le nœud qui détient le jeton, la station transmet myStatus-PDU, cyclicRWr-PDU, cyclicRX-PDU et transient1-PDU, puis transmet à nouveau myStatus-PDU, cyclicRWr-PDU, cyclicRX-PDU et transient1-PDU, suivis par le token-PDU.

5.3.5.7.2 frameInterval

Ce champ spécifie l'intervalle entre la réception de token-PDU et la transmission de myStatus-PDU. La plage de valeurs est 1..255. 0 n'est pas valide. 1 indique que la transmission doit être effectuée dans un intervalle équivalent à l'écart entre les trames, ce qui permet d'obtenir le plus petit intervalle de transmission de trame Ethernet. 2 indique que la transmission doit être effectuée dans un intervalle d'une trame ajoutée à token-PDU plus l'écart entre les trames avant et après cette trame. Si la valeur spécifiée est égale ou supérieure à 2, elle indique que la transmission doit être effectuée, par rapport aux token-PDU, dans un intervalle d'un certain nombre de trames équivalent au nombre spécifié moins un, plus l'écart entre les trames avant et après chaque trame.

5.3.5.7.3 Réservés

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.5.7.4 multipleTokens

Ce champ spécifie le nombre de fois où la transmission de token-PDU a lieu au cours d'une période de détention de jeton. La plage de valeurs est 1..255. 0 n'est pas valide et ne doit pas être utilisé.

NOTE Par exemple, si une station à appareil intelligent, qui spécifie 2 dans ce champ, devient le nœud qui détient le jeton, la station transmet myStatus-PDU, cyclicRWr-PDU, cyclicRX-PDU, transient1-PDU et token-PDU, puis transmet à nouveau le token-PDU.

5.3.5.8 reserved3

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.5.9 dcs

Voir 5.3.2.7.

5.3.6 SetupAck-PDU

5.3.6.1 falArHeader

Voir 5.3.1.

5.3.6.2 slaveNodeInfo

La signification de chaque bit est la suivante:

Bits 7..2: Réservé à un usage futur. La valeur est 0.

Bits 1..0: Spécifie le type d'E/S.

00 = mixte ((l'entrée et la sortie partagent la même adresse)
01 = entrée
10 = sortie
11 = composite (entrée et sortie avec adresses uniques)

5.3.6.3 fwVersion

Ce champ spécifie la version de micrologiciel ("firmware").

5.3.6.4 DeviceType

Ce champ spécifie le type d'appareil.

5.3.6.5 reserved1

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.6.6 vendorCode

Voir 5.3.2.4.

5.3.6.7 modelCode

Voir 5.3.2.5.

5.3.6.8 rySize

Ce champ spécifie la taille RY (nombre d'octets).

5.3.6.9 rwwSize

Ce champ spécifie la taille RWw (nombre de mots).

5.3.6.10 rxSize

Ce champ spécifie la taille RX (nombre d'octets).

5.3.6.11 rwrSize

Ce champ spécifie la taille RWr (nombre de mots).

5.3.6.12 reserved2

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.6.13 availableFuncs

La signification de chaque bit est la suivante:

Bits 7..2: Réservé à un usage futur. La valeur est 0.

Bit 1: Indique la présence de la fonction de réglage du numéro de nœud dans parameter-PDU. 0 indique que la fonction n'est pas disponible; 1 indique que la fonction est disponible. Pour le réglage du numéro de nœud, parameter-PDU ne doit pas être envoyé à un nœud qui ne dispose pas de la fonction nécessaire.

Bit 0: Indique la présence de la fonction de réception transitoire. 0 indique que la fonction n'existe pas; 1 indique que la fonction existe.

5.3.6.14 reserved3

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.6.15 dcs

Voir 5.3.2.7.

5.3.7 F-Token-PDU

5.3.7.1 falArHeader

Voir 5.3.1.

5.3.7.2 tokenDstMacAddr

Ce champ contient l'adresse MAC du nœud de destination de transmission de token-PDU. Chaque nœud évalue si tokenDstMacAddr et l'adresse MAC de son propre nœud correspondent lorsque token-PDU est reçu et, si tel est le cas, évalue si token-PDU est adressé à son propre nœud et devient le nœud détenteur du jeton.

5.3.7.3 tokenSeqNumber

Ce champ contient le numéro de séquence de token-PDU. La plage de valeurs est 1..255.

Le gestionnaire de contrôle de transmission attribue un numéro de séquence différent à chaque passage de jeton. Chaque nœud rejette le deuxième token-PDU et les suivants dont les valeurs tokenSeqNumber sont identiques si des token-PDU présentant la même valeur tokenSeqNumber sont reçus. Dans les nœuds autres que le gestionnaire de contrôle de transmission, les valeurs ne doivent pas être modifiées. Dans les nœuds autres que le gestionnaire de contrôle de transmission, le token-PDU présentant la même valeur que la valeur tokenSeqNumber du token-PDU reçu est transmis.

5.3.7.4 reserved1

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.7.5 tokenHopCounter

Ce champ contient la valeur du compteur de passage de jeton.

La valeur tokenHopCounter du token-PDU à transmettre par le gestionnaire de contrôle de transmission, devenu le nœud détenteur du jeton, doit être 1. Un nœud autre que le gestionnaire de contrôle de transmission, devenu le nœud détenteur de jeton, doit ajouter 1 à la valeur tokenHopCounter du token-PDU reçu, puis procéder à la transmission.

5.3.7.6 traAvailHopCounter

Ce champ spécifie la valeur minimale du compteur de passage de jeton.

Le nœud détenteur du jeton utilise ce champ et la valeur tokenHopCounter pour évaluer si une transmission transitoire est possible. Le nœud détenteur du jeton peut procéder à une transmission transitoire si tokenHopCounter ≥ traAvailHopCounter et traAllows > 0.

Après le début du passage du jeton, le gestionnaire de contrôle de transmission, qui devient le nœud détenteur de jeton pour la première fois, transmet le token-PDU qui spécifie le premier nœud à devoir procéder à une transmission transitoire dans traAvailHopCounter. Si le gestionnaire de contrôle de transmission devient le nœud détenteur de jeton pour la deuxième fois ou plus, le gestionnaire de contrôle de transmission transmet le token-PDU qui définit la même valeur que le traLastHopCounter du token-PDU reçu lors du précédent passage de jeton dans le traAvailHopCounter du token-PDU.

Un nœud autre que le gestionnaire de contrôle de transmission, devenu le nœud détenteur de jeton, transmet le token-PDU qui présente la même valeur que la valeur de traAvailHopCounter du token-PDU reçu.

5.3.7.7 traLastHopCounter

Ce champ indique le dernier compteur de passage de jeton de transmission transitoire. Il s'agit de la valeur tokenHopCounter du nœud détenteur de jeton pour lequel traAllows a été défini sur 0 après l'exécution de la transmission transitoire.

Si le nœud détenteur de jeton procède à une transmission transitoire et si traAllows passe à 0, le nœud doit transmettre le token-PDU dans lequel la valeur tokenHopCounter du token-PDU reçu est définie dans traLastHopCounter.

5.3.7.8 traAllows

Ce champ spécifie le nombre de fois où la transmission transitoire peut avoir lieu. Le nombre est équivalent au nombre de trames de transmission transitoire qui peuvent être transmises par le nœud détenteur de jeton.

Le nœud détenteur de jeton ne doit pas transmettre le transientData-PDU si la valeur traAllows du token-PDU reçu est 0. Après la transmission de la valeur transientData-PDU, le nœud détenteur de jeton doit définir, dans l'élément traAllows du token-PDU à transmettre, une valeur équivalente à la valeur moins le nombre de fois où transientData-PDU a été transmis.

5.3.7.9 reserved2

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.7.10 dcs

Voir 5.3.2.7.

5.3.8 F-Measure-PDU

5.3.8.1 falArHeader

Voir 5.3.1.

5.3.8.2 reserved

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.8.3 dcs

Voir 5.3.2.7.

5.3.9 F-Offset-PDU

5.3.9.1 FalArHeader

Voir 5.3.1.

5.3.9.2 reserved

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.9.3 SyncOffset

Ce champ contient la valeur mesurée du retard associé au chemin de transmission.

5.3.9.4 reserved2

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.9.5 dcs

Voir 5.3.2.7.

5.3.10 F-Update-PDU

5.3.10.1 falArHeader

Voir 5.3.1.

5.3.10.2 reserved

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.10.3 syncOffset

Voir 8.3.9.3.

5.3.10.4 reserved2

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.10.5 dcs

Voir 5.3.2.7.

5.3.11 F-MyStatus-PDU

5.3.11.1 falArHeader

Voir 5.3.1.

5.3.11.2 seqNumber

Ce champ contient un numéro de séquence qui représente l'ordre de transmission des éléments myStatus-PDU et f-cyclicData-PDU à transmettre une fois par le nœud. La signification de chaque bit est la suivante:

Bit 7:	Identification de la dernière trame. 0 indique qu'il existe des trames suivantes; 1 indique qu'il n'existe pas de trame suivante.
Bits 6..0:	Numéro de séquence. La valeur de myStatus-PDU est 0.

5.3.11.3 netNumber

Ce champ contient le numéro de réseau du nœud. La plage est 1..239.

5.3.11.4 masterCmd

La signification de chaque bit est la suivante:

Bits 15..12:	Réservé à un usage futur. La valeur est 0.
Bit 11:	Indique l'exigence de remise de table d'adresses. 0 indique que la remise est requise. Ce champ est utilisé uniquement dans myStatus-PDU. Pour les stations maîtres, cette valeur est 0.
Bit 10..8:	Numéro de séquence de transient1-PDU pour la remise de table d'adresses. Ce champ est utilisé uniquement pour myStatus-PDU transmis par les stations esclaves, à l'exclusion des stations locales. Pour les stations maîtres et locales, cette valeur est 0.
Bit 7:	Réservé à un usage futur. La valeur est 0.
Bit 6:	Réservé à un usage futur. La valeur est 0.
Bit 5:	État d'erreur d'application. 0 indique l'absence d'erreur; 1 indique l'existence d'une erreur. Utilisation uniquement avec myStatus-PDU transmis par la station maître. Pas d'utilisation et définition sur 0 pour les stations esclaves.

Bit 4: État de fonctionnement d'application. 0 indique l'arrêt et 1 l'exécution. Utilisation uniquement avec myStatus-PDU transmis par la station maître. La station esclave utilise 0.

Bit 3: Instruction de fonctionnement cyclique. 0 indique une instruction d'exécution et 1 une instruction d'arrêt. Utilisation uniquement avec MyStatus-PDU transmis par la station maître. Utilisation par la station maître pour attribuer une instruction d'exécution cyclique aux stations esclaves. Une station esclave qui reçoit l'instruction d'arrêt rejette cyclicDataRWw-PDU et cyclicDataRY-PDU reçus et arrête la transmission de cyclicDataRWr-PDU et cyclicDataRX-PDU. Les stations esclaves n'utilisent pas ce champ.

Bits 2..1: Réservé à un usage futur. La valeur est 0.

Bit 0: Identification de la station maître. 0 indique que le nœud n'est pas une station maître; 1 indique que le nœud est une station maître.

5.3.11.5 cyclicStatus

Si la valeur du bit 8 est 1 et la valeur des bits 9 et 10 est 0, la transmission cyclique et la réception sont exécutées. La signification de chaque bit est la suivante:

Bit 15: État d'instruction de fonctionnement cyclique, spécifique au nœud. 0 indique l'exécution; 1 indique l'arrêt. Utilisation uniquement avec myStatus-PDU transmis par la station esclave. La station esclave reflète le bit 19 de la commande parameter-PDU reçue. La station maître n'utilise pas ce champ.

Bit 14: État d'instruction de fonctionnement cyclique, tous les nœuds. 0 indique le fonctionnement; 1 indique l'arrêt. Utilisation uniquement avec myStatus-PDU transmis par la station esclave. La station esclave reflète le bit 3 de la commande myStatus-PDU masterCmd reçue de la station maître. La station maître n'utilise pas ce champ.

Bit 13: État du réglage du nœud réservé. 0 indique que le nœud n'est pas un nœud réservé; 1 indique que le nœud est un nœud réservé. Utilisation uniquement avec myStatus-PDU transmis par la station esclave. La station esclave reflète la valeur de réglage du bit 16 de la commande parameter-PDU reçue. La station maître n'utilise pas ce champ.

Bit 12: État du réglage du numéro de nœud. 0 indique que le numéro est compris dans la plage; 1 indique que le numéro est hors de la plage. Utilisation uniquement avec myStatus-PDU transmis par la station esclave. La station esclave reflète le bit 18 de la commande myStatus-PDU reçue. La station maître n'utilise pas ce champ.

Bit 11: État de confirmation du paramètre de transmission cyclique. 0 indique que l'état est confirmé; 1 indique que la confirmation est en cours.

Bits 10..8: État de maintien du paramètre de transmission cyclique. Les valeurs utilisées sont conformes au Tableau 37.

Bit 7: État d'arrêt pour raisons propres. 0 indique l'absence d'arrêt; 1 indique l'arrêt de la transmission cyclique pour des raisons autres que celles mentionnées ci-avant.

NOTE Par exemple, si une demande d'arrêt est reçue de la couche application ou au démarrage.

Bit 6: État de déconnexion. 0 indique l'absence de déconnexion; 1 indique la déconnexion. La station maître n'utilise pas ce champ.

Bit 5: Réservé à un usage futur. Utilise 0

Bit 4: État invalide du type/numéro de nœud. 0 indique que l'état est valide; 1 indique que l'état est invalide. Utilisation uniquement avec myStatus-PDU transmis par la station esclave. Absence d'utilisation et définition sur 0 pour la station maître.

Bit 3: État de duplication de la station maître. 0 indique l'absence de duplication; 1 indique la duplication. Utilisation uniquement avec myStatus-PDU transmis par la station esclave. La station esclave n'utilise pas ce champ.

Bit 2: État de duplication du numéro de nœud. 0 indique l'absence de duplication; 1 indique la duplication. Utilisation uniquement avec myStatus-PDU transmis par la station esclave. La station esclave reflète le bit 17 de la commande parameter-PDU reçue. La station maître n'utilise pas ce champ.

Bit 1: Erreur d'impossibilité de poursuivre la transmission cyclique. 0 indique l'absence d'erreur; 1 indique l'existence d'une erreur qui empêche de

poursuivre la transmission cyclique.

NOTE Par exemple, une erreur matériel ou une erreur micrologiciel sur son propre nœud.

Bit 0: Réservé à un usage futur. La valeur est 0.

Tableau 37 – État de maintien du paramètre de transmission cyclique

Valeur	Description
001B	Reçu. Paramètre normal.
010b	Non reçu ou absence de correspondance d'ID.
011b	Confirmation en cours.
100B	Reçu. Erreur de paramètre.

5.3.11.6 nodeStatus

La signification de chaque bit est la suivante:

Bits 15..12: Pour un usage futur. La valeur du bit 15 et des bits 13..12 est 0.

Bits 11..10: État d'erreur d'application détaillé. Les valeurs sont définies dans le Tableau 39.

Bits 9..8: État de fonctionnement d'application détaillé. La station maître et la station locale utilisent les valeurs conformément au .Tableau 38. Les bits sont en option pour les stations esclaves (à l'exception de la station locale) et doivent être définis sur 0 en cas de non-utilisation.

Bit 7: Réservé à un usage futur.

Bit 6: État d'activation/désactivation de réception transitoire. 0 indique la désactivation de la réception; 1 indique l'activation de la réception.

Bits 5..4: Réservé à un usage futur. La valeur est 0.

Bit 3: État d'erreur de taille. 0 indique que la taille est normale; 1 indique une erreur de taille. Non utilisé et toujours défini sur 0 pour la station maître et la station locale. Les stations esclaves (à l'exception de la station locale) indiquent une erreur de valeur si les données adressées à leur propre nœud ne sont pas incluses dans le CyclicDataRY-PDU ou le CyclicDataRWw-PDU reçu. La valeur est toujours définie sur 0 pour les stations esclaves (à l'exception de la station locale) qui ne présentent pas de RY ou de RWw.

Bits 2..0: Réservé à un usage futur. La valeur est 0.

Tableau 38 – État de fonctionnement d'application détaillé

Valeur	Description
0	Notification d'état de fonctionnement d'application détaillé non prise en charge
1	Application arrêtée
2	Application en cours d'exécution
3	Utilisateur de l'application inexistant

Tableau 39 – État de détection d'erreur

Valeur	Description
0	Absence d'erreurs
1	Erreur mineure
2	Erreur majeure
3	Erreur grave

5.3.11.7 errorCode

Ce champ contient les codes des erreurs survenues au niveau du nœud.

5.3.11.8 portStatus

Ce champ indique l'état des quatre ports à l'aide de quatre bits chacun. Les bits 3..0 et les bits 7..4 du premier octet indiquent l'état du premier port et l'état du deuxième port, respectivement; les bits 3..0 et les bits 7..4 du deuxième octet indiquent l'état du troisième port et l'état du quatrième port, respectivement. Le premier port est spécifié dans portIndex. La signification de chaque bit du premier octet est la suivante:

Bits 7..6:	Réservé à un usage futur. La valeur est 0.
Bits 5..4:	Les valeurs utilisées sont conformes au Tableau 35.
Bits 3..2:	Réservé à un usage futur. La valeur est 0.
Bits 1..0:	Les valeurs utilisées sont conformes au Tableau 35.

5.3.11.9 portStatistics

Ce champ contient les informations statistiques des quatre ports, avec quatre bits chacun. Les bits 3..0 et les bits 7..4 du premier octet indiquent l'état du premier port et l'état du deuxième port, respectivement; les bits 3..0 et les bits 7..4 du deuxième octet indiquent l'état du troisième port et l'état du quatrième port, respectivement. Le premier port est spécifié dans portIndex. La signification de chaque bit du premier octet est la suivante. La valeur des informations statistiques d'un port non existant est 0.

Bit 7, 3:	Réservé à un usage futur. La valeur est 0.
Bit 6, 2:	Indique la présence d'une erreur de temporisation de la réception d'une token-PDU. 0 indique l'absence d'erreur; 1 indique une erreur. Si le nœud reçoit une token-PDU adressé à son propre nœud, commence la transmission tout en maintenant le jeton, puis reçoit une token-PDU présentant la même valeur tokenSeqNumber, une erreur de temporisation de réception a lieu. L'erreur est annulée si une token-PDU avec une valeur tokenSeqNumber différente est reçu ou si le nœud passe à un état indéterminé de ChannelGroup.
Bit 5, 1:	Indique la détection d'une duplication d'autorité de transmission. 0 indique l'absence de détection; 1 indique une détection. Une détection a lieu si le nœud détient le jeton et si une PDU autre que persuasion-PDU, testData-PDU, testDataAck-PDU, setup-PDU, setupAck-PDU et token-PDU est reçue.
Bit 4, 0:	Indique la présence d'une erreur de réception. 0 indique l'absence d'erreur; 1 indique une erreur. Une erreur est indiquée si une erreur FCS, une erreur de taille inférieure à la réception ou une erreur de taille supérieure à la réception a lieu.

5.3.11.10 portIndex

Ce champ spécifie le numéro de port du premier port de portStatus et portStatistics.

5.3.11.11 réservé

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.11.12 cyclicSequenceNumber

Ce champ est utilisé uniquement avec myStatus-PDU transmis par les stations locales. La station maître et la station esclave (pas la station locale) n'utilisent pas les bits et les définissent sur 0. Les bits sont définis comme suit:

Bit 7:	Identificateur de début. 1 indique le début de la transmission cyclique. 0 indique que la transmission cyclique est en cours.
Bits 6..0:	Diviseur de numéro de séquence. Pour la valeur 0, le numéro de séquence

n'est pas divisé. Un numéro de séquence divisé est dérivé par le biais d'une soustraction séquentielle, en commençant par la valeur du nombre de divisions. Par exemple, un diviseur de 3 donne la séquence de numéros suivante: 3, 2, 1.

5.3.11.13 SlaveSpfEventInfo1

Ce champ est utilisé avec slaveSpfEventInfo2 si une station esclave avertit la station maître d'un événement survenu spécifique aux stations esclaves. Dans myStatus-PDU transmis par la station maître, le champ indique l'état de réception de l'événement spécifique aux stations esclaves. Dans myStatus-PDU transmis par une station esclave, le champ indique le nombre de fois où l'événement spécifique aux stations esclaves est survenu.

When the master station transmits the field, the value used is in accordance with Tableau 40. La valeur est 0x00 au démarrage et 0x01 une fois l'initialisation terminée. Si myStatus-PDU est reçu d'une station esclave, une vérification est menée à bien pour déterminer si la valeur slaveSpfEventInfo1 reçue est différente de la valeur précédemment reçue. Si tel est le cas, la valeur passe à 0x02. Après l'enregistrement du code détaillé de l'événement spécifique aux stations esclaves, indiqué par slaveSpfEventInfo2 dans le myStatus-PDU reçu, la valeur passe à 0x01.

Tableau 40 – État de réception d'événement spécifique à une station esclave

Valeur	Description
0x00	État initial
0x01	En attente de réception
0x02	Réception/enregistrement en cours
0x03..0xFF	Utilisation indisponible

Si une station esclave transmet le champ, la valeur est 0x01..0xFF. Si la valeur slaveSpfEventInfo1 du myStatus-PDU reçu de la station maître est 0x01 et si la valeur slaveSpfEventInfo2 a changé par rapport à la valeur précédente, la station esclave transmet de nouvelles informations sur l'événement spécifique aux stations esclaves à la station maître. Si les informations sur l'événement spécifique aux stations esclaves sont transmises à la station maître, le numéro incrémenté est à nouveau attribué à l'événement spécifique aux stations esclaves à transmettre et enregistrer. La station esclave transmet la valeur myStatus-PDU contenant le nombre d'occurrences attribué à slaveSpfEventInfo1 et le code détaillé de l'événement spécifique aux stations esclaves à enregistrer dans slaveSpfEventInfo2.

5.3.11.14 SlaveSpfEventInfo2

Ce champ est utilisé avec slaveSpfEventInfo1 si une station esclave avertit la station maître d'un événement survenu spécifique aux stations esclaves.

Dans myStatus-PDU transmis par la station maître, le champ contient la valeur du compteur de réception d'événements spécifiques aux stations esclaves. La valeur est 0x0000 au démarrage et 0x0001 une fois l'initialisation terminée. La valeur est incrémentée dès la fin du processus de réception et d'enregistrement de l'événement spécifique aux stations esclaves.

Dans myStatus-PDU transmis par une station esclave, la valeur spécifie le code détaillé de l'événement spécifique aux stations esclaves.

Pour la méthode d'utilisation, voir 5.3.11.13.

5.3.11.15 vendorSpfNodeInfo

Ce champ contient les informations sur le nœud spécifique au fournisseur.

5.3.11.16 dcs

Voir 5.3.2.7.

5.3.12 F-CyclicData-PDU

5.3.12.1 Vue d'ensemble

L'arFType indique si la PDU stocke RWw, RY, RWr ou RX. Dans les descriptions suivantes de RWw, RY, RWr et RX, un arFType de 0x82 (CyclicDataRWw-PDU) fait référence à RWw, 0x83 (CyclicDataRY-PDU) à RY, 0x84 (CyclicDataRWr-PDU) à RWr et 0x85 (CyclicDataRX-PDU) à RX.

5.3.12.2 falArHeader

Voir 5.3.1.

5.3.12.3 seqNumber

Voir 5.3.11.2.

5.3.12.4 bothEndsValidity

Ce champ indique si les quatre premiers octets et les quatre derniers octets de cycData doivent être reflétés dans la mémoire partagée. La signification de chaque bit est la suivante:

Bit 7:	Informations relatives à la validité du dernier octet. 1 indique la validité; 0 indique l'invalidité.
Bit 6:	Informations relatives à la validité du deuxième octet par rapport au dernier. 1 indique la validité; 0 indique l'invalidité.
Bit 5:	Informations relatives à la validité du troisième octet par rapport au dernier. 1 indique la validité; 0 indique l'invalidité.
Bit 4:	Informations relatives à la validité du quatrième octet par rapport au dernier. 1 indique la validité; 0 indique l'invalidité.
Bit 3:	Informations relatives à la validité du quatrième octet par rapport au premier. 1 indique la validité; 0 indique l'invalidité.
Bit 2:	Informations relatives à la validité du troisième octet par rapport au premier. 1 indique la validité; 0 indique l'invalidité.
Bit 1:	Informations relatives à la validité du deuxième octet par rapport au premier. 1 indique la validité; 0 indique l'invalidité.
Bit 0:	Informations relatives à la validité du premier octet. 1 indique la validité; 0 indique l'invalidité.

5.3.12.5 cycDataSize

La signification de chaque bit est la suivante:

Bits 15..12:	Réservé à un usage futur.
Bits 11..0:	Indique la taille de RWw, RY, RWr ou RX. Spécifie la taille en unités de quatre octets. Les quatre premiers octets et les quatre derniers octets à ne pas écrire dans bothEndsValidity sont aussi inclus dans la taille.

5.3.12.6 offsetAddr

Ce champ spécifie l'adresse de décalage à partir du début de RWw, Ry, RWr ou RX. Il est spécifié par une valeur en unités de quatre octets. En d'autres termes, la valeur des bits 1..0 est 0.

5.3.12.7 Réservés

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.12.8 cycData

Ce champ contient les données de RWw, RY, RWr ou RX. Si les 16 octets ne sont pas remplis, le bourrage est de 0x00.

5.3.12.9 dcs

Voir 5.3.2.7.

5.3.13 Transient1-PDU

5.3.13.1 falArHeader

Voir 5.3.1.

5.3.13.2 traMsgHeader

5.3.13.2.1 Réservés

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.13.2.2 seqNumber

La signification de chaque bit de ce champ est la suivante:

Bit 7:	Indique si la PDU est la dernière PDU. Une valeur de 0 indique que la PDU n'est pas la dernière PDU. Une valeur de 1 indique que la PDU est la dernière PDU.
Bits 6..0:	Indique le nombre en cas de division des données.

5.3.13.2.3 dataId

Ce champ contient le numéro d'identification des données transitoires. La plage est 0x00..0xFF. Les données transitoires divisées possèdent le même numéro d'identification.

5.3.13.2.4 wholeDataSize

Ce champ spécifie la quantité de données transitoires en unités d'octets.

5.3.13.2.5 offsetAddr

Ce champ spécifie l'adresse de décalage. Le champ est utilisé pour assembler les données transitoires transmises après division. Dans la première PDU, la valeur est 0x00. Dans les PDU suivantes, la position des données vis-à-vis de l'ensemble du volume des données transitoires est spécifiée par une adresse de décalage par rapport au début.

5.3.13.2.6 dataSize

Ce champ spécifie la taille des données transitoires en unités d'octets.

5.3.13.2.7 dataSubType

Ce champ spécifie le sous-type de données. Les valeurs sont spécifiées dans le Tableau 41. Voir 5.3.1.2.

Tableau 41 – dataSupType de dataType (0x07)

Valeur	Description
0x0000	Non applicable
0x0001	Réservé à un usage futur
0x0002	Spécifique à un système
0x0003..0xFFFF	Réservé à un usage futur

5.3.13.3 data

5.3.13.3.1 Vue d'ensemble

Ce champ contient les données transitoires. La structure et la taille diffèrent en fonction du dataType en 5.3.1.2 et du dataSubType en 5.3.13.2.7. Si la valeur wholeDataSize en 5.3.13.2.4 dépasse 1 466 octets, la taille des données transitoires est indiquée par la valeur dataSize en 5.3.13.2.6 à partir du décalage indiqué par la valeur offsetAddr de 5.3.13.2.5. Si les 16 octets ne sont pas remplis, le bourrage est de 0x00.

5.3.13.3.2 FieldSpecificTransient

5.3.13.3.2.1 opHeader

La structure de opHeader est indiquée dans le Tableau 42.

Tableau 42 – FieldSpecificTransient opHeader

Champ	Description
command	spécifie le type de commande spécifique au champ. Les valeurs utilisées pour chaque dataType et dataSubType sont conformes au Tableau 43. Pour le dataType, voir 5.3.1.2 et pour le dataSubType, voir 5.3.13.2.7.
subCommand	spécifie le type de sous-commande. Les valeurs pour chaque commande sont conformes au Tableau 44.
rtn	non utilisé, contient une valeur de 0x0000 si le type de sous-commande est "demande" (request). Le champ contient une valeur de retour si le type de sous-commande est "réponse" (response).
reserved1	Réservé à un usage futur. La valeur de chaque octet est 0x00.
destNetNumber	Spécifie le numéro de réseau destination. 0 indique une diffusion. Pour la remise des informations relatives au nœud, la valeur 0 est utilisée.
destNodeNumber	Spécifie le numéro de nœud destination. 0xFFFF indique une diffusion. Pour la remise des informations relatives au nœud, la valeur 0xFFFF est utilisée.
reserved2	Réservé à un usage futur. La valeur de chaque octet est 0x00.
srcNetNumber	Spécifie le numéro de réseau source de transmission.
srcNodeNumber	Spécifie le numéro du nœud source de transmission.
reserved3	Réservé à un usage futur. La valeur de chaque octet est 0x00.

Tableau 43 – command (dataType: 0x07, dataSubType: 0x0002)

Valeur	Description
0x00	Non applicable
0x01	Remise des informations relatives au nœud (nodeInfoDist)
0x02	Réservé à un usage futur
0x03	Obtention d'informations statistiques (Statistiques)
0x04	Acquisition d'informations détaillées relatives au nœud (nodeInfoDetail)

Valeur	Description
0x05..0xFF	Réservé à un usage futur

Tableau 44 – Type subCommand pour chaque type de commande

Valeur de Command	Valeur	Description
0x01	0x00	Request (demande)
0x03	0x00	Request (demande)
	0x80	Response
0x04	0x00	Request (demande)
	0x80	Response

5.3.13.3.2.2 fSTraData

La structure de ce champ diffère en fonction de dataType et de dataSubType. Pour le type de données, voir 5.3.1.2. Pour le sous-type de données, voir 5.3.13.2.7.

La structure de la remise des informations relatives au nœud (Deliver node information) est indiquée dans le Tableau 45.

Tableau 45 – Structure de la "Remise des informations relatives au nœud "

Champ	Description
seqNumber	contient le numéro de séquence de remise.
masterNetNumber	contient le numéro de réseau de la station maître.
masterDeviceType	contient le numéro d'appareil de la station maître.
masterModelCode	contient le numéro de modèle de la station maître. Le code du modèle est spécifique au fournisseur.
masterVendorCode	contient le code du fournisseur de la station maître. Pour le code du fournisseur, voir l'édition sur le profil de l'appareil.
masterNodeType	contient le type de nœud de la station maître.
reserved1	Réservé à un usage futur.
masterMacAddress	contient l'adresse MAC de la station maître.
reserved2	Réservé à un usage futur.
dataNum	contient le nombre de remises d'informations relatives au nœud.
message	contient les informations relatives au nœud telles que spécifiées dans le Tableau 46. Les informations relatives au nœud dont la quantité est indiquée dans dataNum sont continues.

Tableau 46 – Structure de la remise des informations relatives au nœud – message

Champ	Description
nodeNumber	contient le numéro de nœud.
reserved1	Réservé à un usage futur. La valeur de chaque octet est 0x00.
availableFuncs	Voir 5.3.6.13.
reserved2	Réservé à un usage futur. La valeur de chaque octet est 0x00.
netNumber	contient le numéro de réseau.
DeviceType	contient le type d'appareil.
modelCode	Voir 5.3.2.5.

Champ	Description
vendorCode	Voir 5.3.2.4.
nodeType	contient le type de nœud.
reserved3	Réservé à un usage futur.
MACAddress	contient l'adresse MAC.
reserved4	Réservé à un usage futur. La valeur de chaque octet est 0x00.

Il n'existe aucun champ pour la demande d'obtention d'informations statistiques. La structure de l'obtention d'informations statistiques (Get statistical information) est indiquée dans le Tableau 47.

Tableau 47 – Structure de la réponse à une demande d'obtention d'informations statistiques (Get statistical informantion response)

Champ	Description
port1Mib1	Indique le nombre de trames en erreur HEC dans le port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib2	indique le nombre de trames d'erreur DCS/FCS du port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib3	indique le nombre de trames d'erreur de taille inférieure (28 octets) du port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib4	indique le nombre de trames transférées du port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib5	indique le nombre de trames remises par le port 1 à la couche supérieure. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib6	indique le nombre de trames rejetées en raison d'un tampon de transfert plein dans le port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib7	indique le nombre de trames remises à la couche supérieure et rejetées en raison d'un tampon plein dans le port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
réservé	Réservé à un usage futur. La valeur de chaque octet est 0x00.
port2Mib1	Indique le nombre de trames en erreur HEC dans le port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib2	indique le nombre de trames d'erreur DCS/FCS du port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib3	indique le nombre de trames d'erreur de taille inférieure (28 octets) du port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib4	indique le nombre de trames transférées du port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib5	indique le nombre de trames remises par le port 2 à la couche supérieure. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib6	indique le nombre de trames rejetées en raison d'un tampon de transfert plein dans le port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib7	indique le nombre de trames remises à la couche supérieure et rejetées en raison d'un tampon plein dans le port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
healthStatusNum	indique le nombre de données d'état de robustesse. La plage de valeurs est 0..128.
healthStatus	indique l'état de robustesse. Les informations sur l'état de robustesse dont le nombre est indiqué dans healthStatusNum sont continues. Cette valeur est spécifique à un fournisseur.

Il n'existe aucun champ pour la demande d'acquisition des détails du nœud. Les champs pour la réponse à la demande d'acquisition des détails du nœud (Acquistion of node details response) sont spécifiés dans le Tableau 48.

Tableau 48 – Structure de la réponse à une demande d'acquisition des détails du nœud (Acquisition of node details response)

Champ	Description
rySize	Voir 5.3.6.8.
rwwSize	Voir 5.3.6.9.
rxSize	Voir 5.3.6.10.
rwrSize	Voir 5.3.6.11.
reserved1	Réservé à un usage futur. La valeur de chaque octet est 0x00.
ports	Indique le nombre de ports.
tokenKeepTime	Voir 5.3.4.7.
netBehaviour	Voir 5.3.5.7.
nodeInfo	Voir 5.3.6.2.
fwVersion	Indique la version du micrologiciel ("firmware") du réseau.
DeviceType	contient le type d'appareil.
modelCode	contient le code du modèle.
vendorCode	contient le code du fournisseur.
reserved2	Réservé à un usage futur. La valeur de chaque octet est 0x00.
modelName	contient le nom du modèle.
vendorName	contient le nom du fournisseur.
contInfo	contient l'indicateur d'informations. 1 indique que les informations pertinentes sont contenues dans les champs qui suivent. 0 indique qu'aucune information pertinente ne suit.
contFwVersion	conitient la version du micrologiciel ("firmware") du contrôleur.
contDeviceType	contient le type d'appareil du contrôleur.
contModelCode	contient le code du modèle de contrôleur.
contVendorCode	contient le code du fournisseur du contrôleur.
reserved3	Réservé à un usage futur. La valeur de chaque octet est 0x00.
contModelName	contient le nom du modèle du contrôleur.
contVendorName	contient le nom du fournisseur du contrôleur.
contVendorSpecificInfo	contient les informations sur l'appareil spécifiques à un fournisseur.

5.3.14 TransientAck-PDU

5.3.14.1 falArHeader

Voir 5.3.1.

5.3.14.2 acks

Ce champ spécifie le nombre de ackData. La valeur est 1.

5.3.14.3 ackData

5.3.14.3.1 nodeNumber

Ce champ contient le numéro de nœud.

5.3.14.3.2 reserved1

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.14.3.3 connectionInfo

Ce champ contient les informations de connexion du f-transientData-PDU reçu. La plage de valeurs est 0x01..0xFF.

5.3.14.3.4 dataSubType

Si le type de trame est 0x22, le sous-type de données reçu est défini. Voir 5.3.13.2.7. Si le type de trame est 0x25, la valeur 0x0000 est fixée.

5.3.14.3.5 ret

Ce champ indique le résultat de la réponse à la demande du f-transientData-PDU reçu. 0 indique un résultat normal; une valeur autre que 0 indique le code d'erreur en cas d'anomalie.

5.3.14.4 dcs

Voir 5.3.2.7.

5.3.15 Transient2-PDU

5.3.15.1 falArHeader

Voir 5.3.1.

5.3.15.2 l

Ce champ spécifie la longueur des données de fno à data (en octets).

5.3.15.3 reserved

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.15.4 tp

La signification de chaque bit de ce champ est la suivante:

Bits 7..4: Indique le type. La valeur est 0.

Bits 3..0: Indique le numéro de séquence.

5.3.15.5 fno

La signification de chaque bit de ce champ est la suivante:

Bit 7: Indique l'identification de la première trame. Une valeur de 0 indique que la trame est une trame sans début. Une valeur de 1 indique que la trame est une trame support.

Bits 6..0: Indique le numéro de trame divisée. Une valeur de 0 indique l'absence de division. Une valeur de 1 à 7 indique le numéro de trame divisée. Le numéro de trame divisée diminue de façon séquentielle, en commençant par le même numéro que le nombre de divisions.

Exemple Si une trame est divisée en trois, les numéros de trame divisée sont envoyés dans l'ordre suivant: 3, 2, 1.

5.3.15.6 dt

La signification de chaque bit de ce champ est la suivante:

Bit 7:	Indique la priorité. 0 indique une priorité basse; 1 indique une priorité haute.
Bit 6:	Indique la présence d'une trame de réponse. Si la valeur est 0, cela signifie qu'une trame de réponse est requise. Si la valeur est 1, cela signifie qu'aucune trame de réponse n'est nécessaire.
Bits 5-0:	Réservé à un usage futur.

5.3.15.7 da

Ce champ spécifie le numéro de nœud du nœud qui procède au transfert dans le réseau local si le nœud de destination se trouve dans un réseau différent.

5.3.15.8 sa

Ce champ indique le numéro de nœud du nœud source.

5.3.15.9 dat

Ce champ spécifie le type d'application de destination. La valeur est fixée à 0x22.

5.3.15.10 sat

Ce champ indique le type d'application source. La valeur est fixée à 0x22.

5.3.15.11 dmf

Ce champ spécifie la destination du module d'exécution. Les valeurs utilisées sont conformes au Tableau 49.

Tableau 49 – Spécification du module d'exécution

Valeur	Description
0x00	Au sein du module
0x01	Au sein du contrôleur
0x02	Réservé à un usage futur
0x03..0xFF	Au sein d'un autre module

5.3.15.12 smf

Ce champ spécifie la source du module d'exécution. Les valeurs utilisées sont conformes au Tableau 49.

5.3.15.13 dna

Ce champ spécifie le numéro de réseau du nœud de destination. La plage de valeurs est 0x00..0xEF et 0xFE. 0x00 indique qu'aucun réseau n'est spécifié; 0xFE indique le réseau par défaut.

5.3.15.14 ds

Ce champ spécifie le numéro de nœud du nœud de destination cible de l'autre réseau pendant le transfert.

5.3.15.15 did

La signification de chaque bit de ce champ est la suivante:

Bits 15..10: Zone de spécification du système. Stocke le numéro du nœud de destination cible.

Bits 9..0: Indique le numéro d'identification de destination cible.

5.3.15.16 sna

Ce champ indique le numéro de réseau du nœud source de démarrage. La plage de valeurs est 0x00..0xEF et 0xFE. 0x00 indique qu'aucun réseau n'est spécifié; 0xFE indique la spécification du réseau par défaut.

5.3.15.17 ss

Ce champ indique le numéro de nœud du nœud source de transmission de l'autre réseau pendant le transfert.

5.3.15.18 sid

La signification de chaque bit de ce champ est la suivante:

Bits 15..10: Zone de spécification du système. Stocke le numéro du nœud source de démarrage.

Bits 9..0: Indique le numéro d'identification source de démarrage.

5.3.15.19 l1

Ce champ spécifie la longueur des données (en unités d'octets) de ct à data.

5.3.15.20 ct

Ce champ spécifie le type de commande. Les valeurs utilisées sont conformes au Tableau 50.

Tableau 50 – Type de commande

Valeur	Description
0x00	Non applicable
0x01..0x03	Réservé à un usage futur
0x04	Obtenir info d'accès mémoire
0x05..0x07	Réservé à un usage futur
0x08	Run (Exécution)
0x09	STOP (Arrêt=
0x0A..0x0F	Réservé à un usage futur
0x10	Lire mémoire
0x11	Réservé à un usage futur
0x12	Écriture dans la mémoire
0x13..0x5F	Réservé à un usage futur
0x60..0x7F	Spécifique au fournisseur

5.3.15.21 rsv

Réservé à un usage futur.

5.3.15.22 aps

La signification de chaque bit de ce champ est la suivante:

Bits 15..11:	Indique le numéro de tâche attribué à la tâche. La plage de valeurs est 0..255.
Bits 8..0:	Indique le numéro d'identification de l'application source de démarrage. La plage de valeurs est 0..255.

5.3.15.23 rsts

Ce champ est utilisé pour les réponses uniquement. La signification de chaque bit est la suivante:

Bits15..12:	Définition du fournisseur
Bits 11..8:	Emplacement d'occurrence d'erreur
Bit 7:	Criticité d'erreur. 0 indique une erreur d'avertissement; 1 indique une erreur majeure.
Bits 6..0:	Code d'erreur

5.3.15.24 donnée

Voir 5.2.12.23.

5.3.15.25 dcs

Voir 5.3.2.7.

5.3.16 ParamCheck-PDU

5.3.16.1 falArHeader

Voir 5.3.1.

5.3.16.2 reserved1

Ce champ est réservé à un usage futur. La valeur des premier, deuxième et quatrième octets est 0x00. La valeur des bits 0..3 du troisième octet est 0.

5.3.16.3 paramId

5.3.16.3.1 Vue d'ensemble

Ce champ contient l'identification du paramètre. Une valeur de 0 indique une instruction de rejet de paramètre.

5.3.16.3.2 date

La signification de chaque bit des éléments de ce champ est la suivante:

Bits 31..28	Jour (dizaines)
Bits 27..24	Jour (unités)
Bits 23..20	Mois (dizaines)
Bits 19..16	Mois (unités)

Bits 15..12	Année (dizaines)
Bits 11..8	Année (unités)
Bits 7..4	Année (milliers)
Bits 3..0	Année (centaines)

5.3.16.3.3 timeNodeId

La signification de chaque bit des éléments de ce champ est la suivante:

Bits 31..24	Numéro de nœud de source de réglage
Bits 23..20	Secondes (dizaines)
Bits 19..16	Secondes (unités)
Bits 15..12	Minutes (dizaines)
Bits 11..8	Minutes (unités)
Bits 7..4	Heure (dizaines)
Bits 3..0	Heure (unités)

5.3.16.3.4 Somme de contrôle

Ce champ contient la valeur sum-check des paramètres communs détenus par la station maître.

5.3.16.4 reserved3

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.16.5 dcs

Voir 5.3.2.7.

5.3.17 Parameter-PDU

5.3.17.1 falArHeader

Voir 5.3.1.

5.3.17.2 paramSetFlag

Ce champ contient la spécification du paramètre. La signification de chaque bit est la suivante:

Bits 7..3:	Réservé à un usage futur. La valeur est 0.
Bit 2:	Réglage du paramètre commun. 0 indique que le paramètre n'est pas valide; 1 indique que le paramètre est valide.
Bit 1:	Réglage de la commande de fonctionnement. 0 indique que le paramètre n'est pas valide; 1 indique que le paramètre est valide.
Bit 0:	Réglage du numéro de nœud et du numéro de réseau. 0 indique que le paramètre n'est pas valide; 1 indique que le paramètre est valide.

5.3.17.3 addressOrder

5.3.17.3.1 Vue d'ensemble

Ce champ est utilisé si le bit 0 de paramSetFlag indique un paramètre valide. Le champ n'est pas utilisé et la valeur est 0 si paramSetFlag indique un paramètre invalide.

5.3.17.3.2 assignedNetNumber

Ce champ spécifie la valeur de réglage du numéro de réseau. La plage est 1..239.

5.3.17.3.3 assignedNodeNumber

Ce champ spécifie la valeur de réglage du numéro de nœud.

5.3.17.4 cmdOrder

5.3.17.4.1 Vue d'ensemble

Ce champ est utilisé si le bit 1 de paramSetFlag indique un paramètre valide. Le champ n'est pas utilisé et la valeur est 0 si paramSetFlag indique un paramètre invalide.

5.3.17.4.2 CMD

La signification de chaque bit des éléments de ce champ est la suivante:

Bits 23..20:	Réservé à un usage futur. La valeur est 0.
Bit 19:	Instruction d'arrêt de transmission cyclique causée par l'évaluation de la station maître. 0 indique l'activation; 1 indique la désactivation. La station maître envoie cette instruction aux stations esclaves. La station esclave reflète l'état d'instruction dans le bit 15 du cyclicStatus du myStatus-PDU.
Bit 18:	Instruction d'arrêt de transmission cyclique causée par un numéro de nœud invalide. 0 indique l'activation; 1 indique la désactivation. La station maître envoie cette instruction aux stations esclaves. La station esclave reflète l'état d'instruction dans le bit 3 du cyclicStatus du myStatus-PDU.
Bit 17:	Instruction d'arrêt de transmission cyclique causée par un numéro de nœud dupliqué. 0 indique l'activation; 1 indique la désactivation. La station maître envoie cette instruction aux stations esclaves. La station esclave reflète l'état d'instruction dans le bit 13 du cyclicStatus du myStatus-PDU.
Bit 16:	Réglage de nœud réservé. 0 indique que le nœud est un nœud réservé; 1 indique que le nœud n'est pas un nœud réservé. La station maître utilise ce réglage en cas de spécification d'une station esclave comme nœud réservé. Une station esclave spécifiée comme nœud réservé ne procède pas à une transmission cyclique. La station esclave reflète l'état de ce réglage dans le bit 2 du cyclicStatus du myStatus-PDU.
Bits 15..5:	Réservé à un usage futur. La valeur est 0.
Bit 4:	Indique si le type de nœud est valide, tel que déterminé par comparaison des champs relatifs au type de nœud, à savoir le champ nodeType et le type de station de la station. 0 indique que le type de nœud est invalide et que la transmission cyclique n'est pas effectuée.
Bits 3..0:	Réservé à un usage futur. La valeur est 0.

5.3.17.4.3 nodeType

Voir 5.3.1.3.3.

5.3.17.5 cyclicParameter

5.3.17.5.1 Vue d'ensemble

Ce champ est utilisé si le bit 2 de paramSetFlag indique un réglage. Le champ n'est pas utilisé et la valeur est 0 si paramSetFlag indique une absence de réglage.

5.3.17.5.2 paramId

Voir 5.3.16.3.

5.3.17.5.3 reserved1

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.4 masterStatus

La signification de chaque bit des éléments de ce champ est la suivante:

Bits 15..1:	Réservé à un usage futur. La valeur est 0.
Bit 0:	Fonction de garantie de nœud. 0 indique que le nœud n'est pas garanti; 1 indique que le nœud est garanti. Utilisation par la station maître pour indiquer à la station esclave si la station maître possède une fonction de garantie de nœud.

5.3.17.5.5 rySeqNumber

La signification de chaque bit des éléments de ce champ est la suivante:

Bit 7:	Réservé à un usage futur. La valeur est 0.
Bits 6..0:	Numéro de séquence de cyclicDataRY-PDU à recevoir. La plage de valeurs est 1..127.

5.3.17.5.6 ryBothEndsValidity

Ce champ indique s'il convient de refléter dans la mémoire partagée les quatre premiers octets et les quatre derniers octets de la zone spécifiée par ryDataSize et ryOffset dans les valeurs cycData du cyclicDataRY-PDU présentant le numéro de séquence spécifié par rySeqNumber. Si les valeurs ryDataSize et ryOffset sont spécifiées de sorte que les données chevauchent plusieurs cyclicDataRY-PDU, les quatre derniers octets sont inclus dans le dernier cyclicDataRY-PDU.

La signification de chaque bit est la suivante:

Bit 7:	Informations du dernier octet activées. 1 indique l'activation; 0 indique la désactivation.
Bit 6:	Informations du deuxième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 5:	Informations du troisième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 4:	Informations du quatrième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 3:	Informations du quatrième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 2:	Informations du troisième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 1:	Informations du deuxième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 0:	Informations de l'octet initial activées. 1 indique l'activation; 0 indique la désactivation.

5.3.17.5.7 ryDataSize

Ce champ spécifie la taille des données à refléter dans la mémoire partagée parmi les valeurs cycData du cyclicDataRY-PDU spécifiées dans rySeqNumber, depuis l'adresse précisée dans ryOffset. La valeur est spécifiée en unités de quatre octets. Les quatre octets initiaux et les quatre derniers octets spécifiés dans ryBothEndsValidity sont aussi inclus dans la taille.

En fonction des valeurs de ryDataSize et ryOffset, la valeur reflétée dans la mémoire partagée, avec le cyclicDataRY-PDU spécifié dans rySeqNumber comme début, peut chevaucher plusieurs cyclicDataRY-PDU.

5.3.17.5.8 ryOffset

Ce champ spécifie le début de la zone parmi les valeurs cycData du cyclicDataRY-PDU à refléter dans la mémoire partagée, en octets depuis le début de cycData. La valeur est spécifiée en unités de quatre octets.

En fonction des valeurs de ryDataSize et ryOffset, la valeur reflétée dans la mémoire partagée, avec le cyclicDataRY-PDU spécifié dans rySeqNumber comme début, peut chevaucher plusieurs cyclicDataRY-PDU.

5.3.17.5.9 reserved2

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.10 rwwSeqNumber

Ce champ est défini comme suit:

Bit 7:	Réservé à un usage futur. La valeur est 0.
Bits 6..0:	Numéro de séquence du cyclicDataRWw-PDU à recevoir. La plage de valeurs est 1..127.

5.3.17.5.11 reserved3

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.12 rwwDataSize

Ce champ spécifie la taille parmi les valeurs cycData du cyclicDataRY-PDU présentant le numéro de séquence spécifié dans rwwSeqNumber à refléter dans la mémoire partagée à partir de l'adresse spécifiée par rwwOffset. La valeur est spécifiée en unités de quatre octets.

En fonction des valeurs de rwwDataSize et rwwOffset, la valeur reflétée dans la mémoire partagée, avec le cyclicDataRWw-PDU spécifié dans rwwSeqNumber comme début, peut chevaucher plusieurs cyclicDataRWw-PDU.

5.3.17.5.13 rwwOffset

Ce champ spécifie le début de la zone à refléter dans la mémoire partagée parmi les valeurs cycData du cyclicDataRWw-PDU présentant le numéro de séquence spécifié dans rwwSeqNumber, avec un décalage par rapport au début de cycData. La valeur est spécifiée en unités de quatre octets.

En fonction des valeurs de rwwDataSize et rwwOffset, la valeur reflétée dans la mémoire partagée, avec le cyclicDataRWw-PDU spécifié dans rwwSeqNumber comme début, peut chevaucher plusieurs cyclicDataRWw-PDU.

5.3.17.5.14 reserved4

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.15 rxBothEndsValidity

Ce champ spécifie si les quatre premiers octets et les quatre derniers octets de la zone spécifiée par rxDataSize et rxOffset parmi les valeurs cycData du cyclicDataRX-PDU constituent la zone reflétée depuis la mémoire partagée.

La signification de chaque bit est la suivante:

Bit 7:	Informations du dernier octet activées. 1 indique l'activation; 0 indique la désactivation.
Bit 6:	Informations du deuxième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 5:	Informations du troisième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 4:	Informations du quatrième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 3:	Informations du quatrième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 2:	Informations du troisième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 1:	Informations du deuxième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 0:	Informations de l'octet initial activées. 1 indique l'activation; 0 indique la désactivation.

5.3.17.5.16 rxDataSize

Ce champ spécifie la taille des données de cycData du cyclicDataRY-PDU reflétées par la mémoire partagée à l'aide de l'adresse spécifiée dans rxOffset comme début. Les quatre premiers octets et les quatre derniers octets spécifiés dans rxBothEndsValidity sont aussi inclus dans la taille.

5.3.17.5.17 rxOffset

Ce champ spécifie le début de la zone parmi les valeurs cycData du cyclicDataRX-PDU reflété par la mémoire partagée, en octets depuis le début de cycData. La valeur est spécifiée en unités de quatre octets. La valeur des bits 1..0 est 0.

5.3.17.5.18 reserved5

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.19 rwrDataSize

Ce champ indique la taille parmi les valeurs cycData du cyclicDataRY-PDU reflété par la mémoire partagée à l'aide de l'adresse spécifiée par rwrOffset comme début. La valeur est spécifiée en unités de quatre octets.

5.3.17.5.20 rwrOffset

Ce champ spécifie le début de la zone reflétée par la mémoire partagée parmi les valeurs cycData du cyclicDataRWr-PDU, avec un décalage par rapport au début de cycData. La valeur est spécifiée en unités de quatre octets. La valeur des bits 1..0 est 0.

5.3.17.5.21 reserved6

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.22 masterWatchTimer

Ce champ spécifie la valeur de réglage de MasterWatchTimer.

Bit 15:	Réservé à un usage futur. 0 est utilisé.
Bits 14..0:	Valeur de réglage de MasterWatchTimer. La plage de valeurs est 1..32 767. L'unité est 400 µs.

5.3.17.5.23 reserved7

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.24 cmRyBothEndsValidity

Ce champ spécifie s'il convient de refléter dans la mémoire partagée les quatre premiers octets et les quatre derniers octets de la zone spécifiée par cmRyDataSize et cmRyOffset dans les valeurs RY transmises par la station maître. Le champ est utilisé uniquement avec la station locale. Si les valeurs cmRyDataSize et cmRyOffset sont spécifiées de sorte que les données chevauchent plusieurs cyclicDataRY-PDU, les quatre derniers octets sont inclus dans le dernier cyclicDataRY-PDU.

La signification de chaque bit est la suivante:

Bit 7:	Informations du dernier octet activées. 1 indique l'activation; 0 indique la désactivation.
Bit 6:	Informations du deuxième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 5:	Informations du troisième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 4:	Informations du quatrième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 3:	Informations du quatrième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 2:	Informations du troisième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 1:	Informations du deuxième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 0:	Informations de l'octet initial activées. 1 indique l'activation; 0 indique la désactivation.

5.3.17.5.25 cmRyDataSize

Ce champ spécifie la taille des données à refléter dans la mémoire partagée parmi les valeurs RY transmises par la station maître, depuis l'adresse précisée dans cmRyOffset. Les quatre premiers octets et les quatre derniers octets spécifiés dans cmRyBothEndsValidity sont aussi inclus dans la taille. Ce champ est utilisé par la station locale uniquement.

5.3.17.5.26 cmRyOffset

Ce champ spécifie le début de la zone parmi les valeurs RY transmises par la station maître à refléter dans la mémoire partagée, en octets depuis le début de RY. La valeur est spécifiée en unités de quatre octets. Ce champ est utilisé par la station locale uniquement.

5.3.17.5.27 reserved8

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.28 cmRwwDataSize

Ce champ spécifie la taille parmi les valeurs RWw transmises par la station maître à refléter dans la mémoire partagée, depuis l'adresse précisée dans cmRwwOffset. La valeur est spécifiée en unités de quatre octets. Le champ est utilisé par la station locale uniquement.

5.3.17.5.29 cmRwwOffset

Ce champ spécifie le début de la zone à refléter dans la mémoire partagée parmi les valeurs RWw transmises par la station maître, avec un décalage par rapport au début de RWw. La valeur est spécifiée en unités de quatre octets. Le champ est utilisé par la station locale uniquement.

5.3.17.5.30 reserved9

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.31 cmRxBothEndsValidity

Ce champ spécifie si les quatre premiers octets et les quatre derniers octets de la zone spécifiée par cmRxDataSize et cmRxOffset parmi les valeurs RX à recevoir par la station maître depuis une station esclave et construites au niveau de la station maître doivent être reflétés dans la mémoire partagée. Le champ est utilisé par la station locale uniquement. Si les valeurs cmRyDataSize et cmRyOffset sont spécifiées de sorte que les données chevauchent plusieurs cyclicDataRX-PDU, les quatre derniers octets sont inclus dans le dernier cyclicDataRX-PDU.

La signification de chaque bit est la suivante:

Bit 7:	Informations du dernier octet activées. 1 indique l'activation; 0 indique la désactivation.
Bit 6:	Informations du deuxième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 5:	Informations du troisième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 4:	Informations du quatrième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 3:	Informations du quatrième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 2:	Informations du troisième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 1:	Informations du deuxième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 0:	Informations de l'octet initial activées. 1 indique l'activation; 0 indique la désactivation.

5.3.17.5.32 cmRxDataSize

Ce champ spécifie la taille des données à refléter dans la mémoire partagée parmi les valeurs RX transmises par une station esclave à la station maître et construites au niveau de la station maître, depuis l'adresse précisée dans cmRxyOffset. Les quatre premiers octets et les quatre derniers octets spécifiés dans cmRxBothEndsValidity sont aussi inclus dans la taille. Ce champ est utilisé par la station locale uniquement.

5.3.17.5.33 cmRxOffset

Ce champ spécifie le début de la zone à refléter dans la mémoire partagée parmi les valeurs RX transmises par une station esclave à la station maître et construites au niveau de la

station maître, en octets depuis le début de RX. Le champ est utilisé par la station locale uniquement. La valeur est spécifiée en unités de quatre octets. La valeur des bits 1..0 est 0.

5.3.17.5.34 reserved10

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.17.5.35 cmRwrDataSize

Ce champ spécifie la taille des données à refléter dans la mémoire partagée parmi les valeurs RWr transmises par une station esclave à la station maître et construites au niveau de la station maître, depuis l'adresse précisée par cmRwrOffset. Le champ est utilisé par la station locale uniquement.

5.3.17.5.36 cmRwrOffset

Ce champ spécifie le début de la zone à refléter dans la mémoire partagée parmi les valeurs RWr transmises par une station esclave à la station maître et construites au niveau de la station maître, avec un décalage par rapport au début de RWr. Ce champ est utilisé par la station locale uniquement. La valeur est spécifiée en unités de quatre octets. La valeur des bits 1..0 est 0.

5.3.17.6 dcs

Voir 5.3.2.7.

5.3.18 Timer-PDU

5.3.18.1 falArHeader

Voir 5.3.1.

5.3.18.2 time (durée)

Ce champ contient la valeur du temporisateur en unités de 15,258 789 062 5 µs, en utilisant le 1er janvier 2000, 00:00:00 comme point de référence.

5.3.18.3 réservé

Ce champ est réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.18.4 dcs

Voir 5.3.2.7.

6 Structure du diagramme d'états de protocole FAL

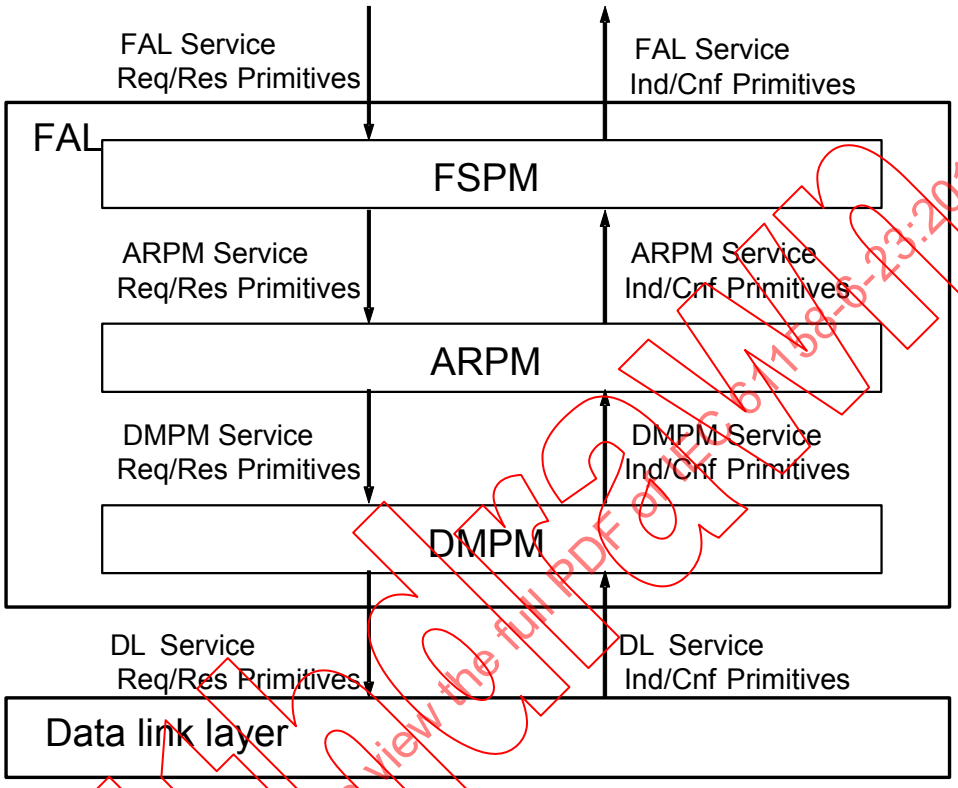
Le diagramme d'états de protocole FAL comprend trois diagrammes d'états de protocole, tel qu'indiqué à la Figure 17. Un diagramme d'états de protocole comprend, dans l'ordre depuis le côté couche liaison de données, un data link layer mapping protocol machine (DMPM, diagramme d'états de protocole de mapping de liaison de données), un application relationship protocol machine (ARPM, diagramme d'états de protocole de mapping d'application) et un FAL service protocol machine (FSPM, diagramme d'états de protocole de services de la FAL).

Le rôle du FSPM consiste à recevoir des primitives de service des utilisateurs de la FAL et à convertir les primitives en primitives internes, à sélectionner un ARPM (diagramme d'états de protocole de mapping d'application), à recevoir les primitives internes de l'ARPM et à

convertir ces dernières en primitives de service de la FAL et à les transmettre aux utilisateurs de la FAL.

Le rôle de l'ARPM consiste à convertir les primitives de service entre l'ARPM et le DMPM.

Le rôle du DMPM est le mapping dans la couche de liaison de données.



Légende

Anglais	Français
FAL Service	Services FAL
Req/Res Primitives	Primitives Req/Res
Ind/Cnf Primitives	Primitives Ind/Cnf
ARPM Service	Service ARPM
DMPM Service	Service DMPM
DL Service	Service DL
FAL	FAL
FSPM	FSPM
ARPM	ARPM
DMPM	DMPM
Data Link Layer	Couche liaison de données

Figure 17 – Relations entre les diagrammes de protocole

7 Machine de protocole FSPM

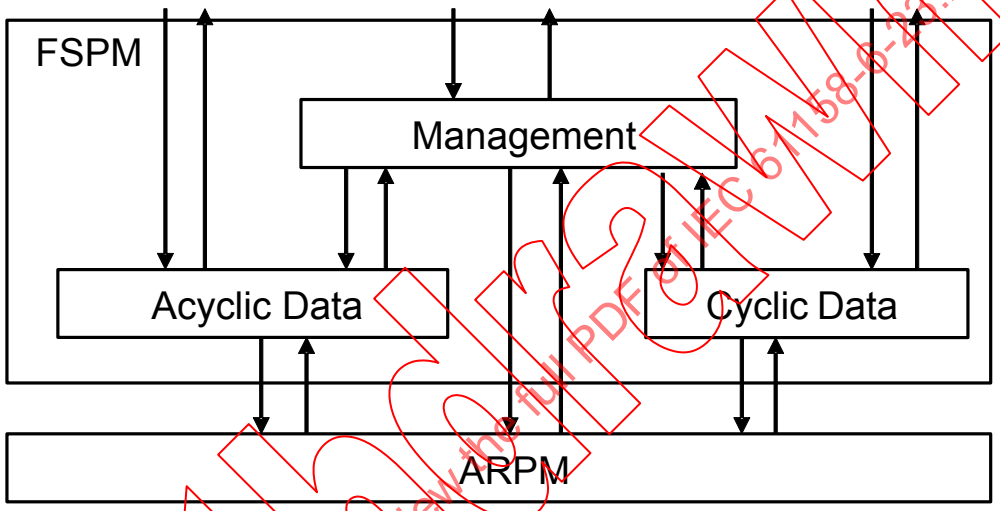
7.1 Vue d'ensemble

Le FSPM offre une interface aux utilisateurs de la FAL. Il procède au mapping entre les services des utilisateurs de la FAL et les services internes de la FAL.

7.2 Type C de FSPM

7.2.1 Vue d'ensemble

Le FSPM comprend trois diagrammes d'états de protocole: données cycliques, données acycliques et gestion. La relation entre les diagrammes d'états de protocole est indiquée à la Figure 18.



Légende

Anglais	Français
FSPM	FSPM
Management	Gestion
Acyclic Data	Données acycliques
Cyclic Data	Données cycliques
ARPM	ARPM

Figure 18 – Structure du type C de FSPM

Les primitives suivantes sont envoyées par l'utilisateur de la FAL au FSPM.

Write Cyclic Data.req
Read Cyclic Data.req
Send Acyclic Data.req
Request Acyclic Data.req
Request Acyclic Data.rsp
Get Attribute.req
Get Attribute.rsp
Set Attribute.req
Set Attribute.rsp

Les primitives suivantes sont envoyées par le FSPM à l'utilisateur de la FAL.

Read Cyclic Data.cnf

Get Attribute.ind

Get Attribute.cnf

Set Attribute.ind

Set Attribute.cnf

Send Acyclic Data.ind

Request Acyclic Data.ind

Request Acyclic Data.cnf

7.2.2 FSPM

7.2.2.1 Cyclic data (données cycliques)

La description détaillée du diagramme d'états des données cycliques est indiquée dans le Tableau 51.

Tableau 51 – Table d'états des données cycliques

N	État actuel	Événement /Condition => Action	État suivant
1	ACTIVE	Write Cyclic Data.req => Update BitCM Data and WordCM Data; CT Update.req	ACTIVE
2	ACTIVE	CT Update.ind => Update BitCM Data or WordCM Data.	ACTIVE
3	ACTIVE	Read Cyclic Data.req => Read Cyclic Data.cnf(BitCM Data, WordCM Data)	ACTIVE

7.2.2.2 Acyclic data (Données acycliques)

La description détaillée du diagramme d'états des données acycliques (Acyclic Data) est indiquée dans le Tableau 52.

Tableau 52 – Table d'états des données acycliques (Acyclic data)

N	État actuel	Événement /Condition => Action	État suivant
1	ACTIVE	Send Parameter 1.req => AC Send.req	ACTIVE
2	ACTIVE	AC Send.ind(Data) / Command == Send Parameter 1 => Send Parameter 1.ind	ACTIVE
3	ACTIVE	Send Parameter 2.req => AC Send.req	ACTIVE
4	ACTIVE	AC Send.ind(Data) / Command == Send Parameter 2 => Send Parameter 2.ind	ACTIVE
5	ACTIVE	Get System Info.req => AC Send.req	ACTIVE
6	ACTIVE	AC Send.ind(Data) / Command == Get System Info =>	ACTIVE

N	État actuel	Événement /Condition => Action	État suivant
		Get System Info.ind	
7	ACTIVE	Get System Info.rsp => AC Send.rsp	ACTIVE
8	ACTIVE	AC Send.cnf(Data) / Command == Get System Info => Get System Info.cnf	ACTIVE
9	ACTIVE	Get Memory Access Info.req => AC Send.req	ACTIVE
10	ACTIVE	AC Send.ind(Data) / Command == Get Memory Access Info => Get Memory Access Info.ind;	ACTIVE
11	ACTIVE	Get Memory Access Info.rsp => AC Send.rsp	ACTIVE
12	ACTIVE	AC Send.cnf(Data) / Command == Get Memory Access Info => Get Memory Access Info.cnf	ACTIVE
13	ACTIVE	Run.req => AC Send.req	ACTIVE
14	ACTIVE	AC Send.ind(Data) / Command == Run => Run.ind	ACTIVE
15	ACTIVE	Run.rsp => AC Send.rsp	ACTIVE
16	ACTIVE	AC Send.cnf(Data) / Command == Run => Run.cnf	ACTIVE
17	ACTIVE	Stop.req => AC Send.req	ACTIVE
18	ACTIVE	AC Send.ind(Data) / Command == Stop => Stop.ind	ACTIVE
19	ACTIVE	Stop.rsp => AC Send.rsp	ACTIVE
20	ACTIVE	AC Send.cnf(Data) / Command == Stop => Stop.cnf	ACTIVE
21	ACTIVE	Line Test.req => AC Send.req	ACTIVE
22	ACTIVE	AC Send.ind(Data) / Command == Line Test => Line Test.ind	ACTIVE
23	ACTIVE	Line Test.rsp => AC Send.rsp	ACTIVE
24	ACTIVE	AC Send.cnf(Data) / Command == Line Test =>	ACTIVE

N	État actuel	Événement /Condition => Action	État suivant
		Line Test.cnf	
25	ACTIVE	Read Memory.req => AC Send.req	ACTIVE
26	ACTIVE	AC Send.ind(Data) / Command == Read Memory => Read Memory.ind	ACTIVE
27	ACTIVE	Read Memory.rsp => AC Send.rsp	ACTIVE
28	ACTIVE	AC Send.cnf(Data) / Command == Read Memory => Read Memory.cnf	ACTIVE
29	ACTIVE	Write Memory.req => AC Send.req	ACTIVE
30	ACTIVE	AC Send.ind(Data) / Command == Write Memory => Write Memory.ind	ACTIVE
31	ACTIVE	Write Memory.rsp => AC Send.rsp	ACTIVE
32	ACTIVE	AC Send.cnf(Data) / Command == Write Memory => Write Memory.cnf	ACTIVE

7.2.2.3 Gestion

La description détaillée du diagramme d'états de gestion (Management) est indiquée dans le Tableau 53.

Tableau 53 – Table d'états de gestion (Management)

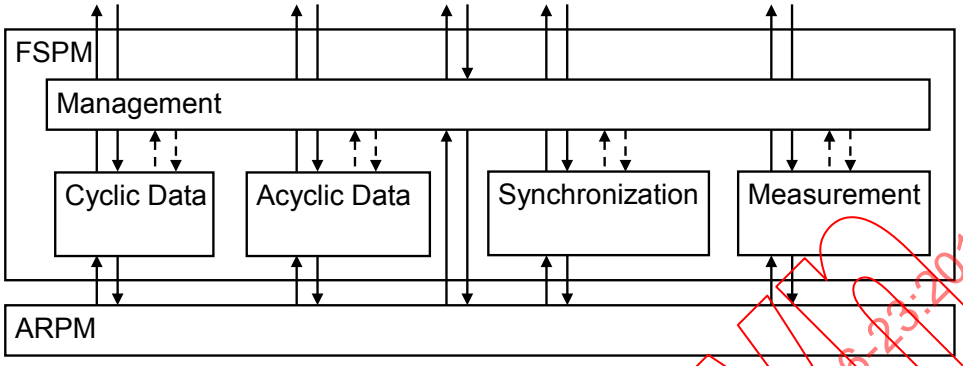
N	État actuel	Événement /Condition => Action	État suivant
1	ACTIVE	Get Attribute.req => Get Attribute.ind	ACTIVE
2	ACTIVE	Get Attribute.rsp => Get Attribute.cnf	ACTIVE
3	ACTIVE	Set Attribute.req => Set Attribute.ind	ACTIVE
4	ACTIVE	Set Attribute.rsp => Set Attribute.cnf	ACTIVE

7.3 Type F de FSPM

7.3.1 Vue d'ensemble

Le comprend cinq diagrammes d'états de protocole: Cyclic data (données cycliques), Acyclic data (données acycliques), Management (gestion), Synchronization (synchronisation) et Measurement (mesure). La relation entre les machines de protocole est indiquée à la Figure 19. La ligne continue représente un problème de service et la ligne en pointillés

représente une liaison entre les machines de protocole qui utilisent notamment des paramètres



Légende

Anglais	Français
FSPM	FSPM
Management	Gestion
Cyclic Data	Données cycliques
Acyclic Data	Données acycliques
Synchronization	Synchronisation
Measurement	Mesure
ARPM	ARPM

Figure 19 – Structure du type F de FSPM

Les primitives suivantes sont envoyées par l'utilisateur de la FAL au FSPM.

RX_Ld.req
RX_Set.req
RX_Reset.req
RX_Read.req
RX_Write.req
RY_Ld.req
RY_Set.req
RY_Reset.req
RY_Read.req
RY_Write.req
RWr_Ld.req
RWr_Set.req
RWr_Reset.req
RWr_Read.req
RWr_Write.req
RWw_Ld.req
RWw_Set.req
RWw_Reset.req

RWw_Read.req
RWw_Write.req
Get Attribute.req
Set Attribute.req
Get Memory Access Info.req
Get Memory Access Info.rsp
Run.req
Run.rsp
Stop.req
Stop.rsp
Read Memory.req
Read Memory.rsp
Write Memory.req
Write Memory.rsp
Vendor Command.req
Vendor Command.rsp
Distribute Node Info.req
Get Statistics.req
Get Statistics.rsp
Get Node Info Detail.req
Get Node Info Detail.rsp
AC Data.req
AC Data.rsp
AC Data ND.req
AC Data ND.rsp
Start Measure.req
Get Offset.req

Les primitives suivantes sont envoyées par le FSPM à l'utilisateur de la FAL.

RX_Ld.cnf
RX_Read.cnf
RY_Ld.cnf
RY_Read.cnf
RWr_Ld.cnf
RWr_Read.cnf
RWw_Ld.cnf
RWw_Read.cnf
Get Attribute.cnf
Set Attribute.cnf
Get Memory Access Info.ind
Get Memory Access Info.cnf
Run.ind
Run.cnf

Stop.ind
Stop.cnf
Read Memory.ind
Read Memory.cnf
Write Memory.ind
Write Memory.cnf
Vendor Command.ind
Vendor Command.cnf
Distribute Node Info.ind
Get Statistics.ind
Get Statistics.cnf
Get Node Info Detail.ind
Get Node Info Detail.cnf
AC Data.ind
AC Data.cnf
AC Data ND.ind
AC Data ND.cnf
Synchroous Triger.ind
Start Measure.cnf
Get Offset.cnf

7.3.2 FSPM

7.3.2.1 données cycliques

La description détaillée du diagramme d'états des données cycliques est indiquée dans le Tableau 54.

Tableau 54 – Table d'états des données cycliques

N	État actuel	Événement /Condition => Action	État suivant
1	ACTIVE	Ld.req(Type, Address) => Ld.cnf(Data)	ACTIVE
2	ACTIVE	Set.req(Type, Address) => met à jour les valeurs RX, RY, RY, RWr ou RWw spécifiées dans le Type. CT Update.req(Type, Address, 1, 1)	ACTIVE
3	ACTIVE	Reset.req(Type, Address) => met à jour les valeurs RX, RY, RY, RWr, ou RWw spécifiées dans le Type. CT Update.req(Type, Address, 1, 1)	ACTIVE
4	ACTIVE	Read.req(Type, Address, Size) => Read.cnf(Data)	ACTIVE
5	ACTIVE	Write.req(Type, Address, Size, Data) => met à jour les valeurs RX, RY, RY, RWr, ou RWw spécifiées dans le Type. CT Update.req(Type, Address, Size, Data)	ACTIVE
11	ACTIVE	CT Update.ind(Type, Offset, Size, Data) => met à jour les valeurs RX, RY, RY, RWr, ou RWw spécifiées	ACTIVE

N	État actuel	Événement /Condition => Action	État suivant
		dans le Type.	

7.3.2.2 données acycliques

La description détaillée du diagramme d'états des données acycliques (Acyclic Data) est indiquée dans le Tableau 55.

Tableau 55 – Table d'états des données acycliques (Acyclic data)

N	État actuel	Événement /Condition => Action	État suivant
1	ACTIVE	Get Memory Access Info.req => AC Send.req	ACTIVE
2	ACTIVE	AC Send.ind(Data) / Command == Get Memory Access Info => Get Memory Access Info.ind;	ACTIVE
3	ACTIVE	Get Memory Access Info.rsp => AC Send.rsp	ACTIVE
4	ACTIVE	AC Send.cnf(Data) / Command == Get Memory Access Info => Get Memory Access Info.cnf	ACTIVE
5	ACTIVE	Run.req => AC Send.req	ACTIVE
6	ACTIVE	AC Send.ind(Data) / Command == Run => Run.ind	ACTIVE
7	ACTIVE	Run.rsp => AC Send.rsp	ACTIVE
8	ACTIVE	AC Send.ind(Data) / Command == Run && Request type == Server Response => Run.cnf	ACTIVE
9	ACTIVE	Stop.req => AC Send.req	ACTIVE
10	ACTIVE	AC Send.ind(Data) / Command == Stop && Request type == Client Response => Stop.ind	ACTIVE
11	ACTIVE	Stop.rsp => AC Send.rsp	ACTIVE
12	ACTIVE	AC Send.cnf(Data) / Command == Stop && Request type == Server Response => Stop.cnf	ACTIVE
13	ACTIVE	Read Memory.req	ACTIVE

N	État actuel	Événement /Condition => Action	État suivant
		=> AC Send.req	
14	ACTIVE	AC Send.ind(Data) / Command == Read Memory && Request type == Client Request => Read Memory.ind	ACTIVE
15	ACTIVE	Read Memory.rsp => AC Send.rsp	ACTIVE
16	ACTIVE	AC Send.ind(Data) / Command == Read Memory && Request type == Server Response => Read Memory.cnf	ACTIVE
17	ACTIVE	Wirte Memory.req => AC Send.req	ACTIVE
18	ACTIVE	AC Send.ind(Data) / Command == Write Memory && Request type == Client Request => Write Memory.ind	ACTIVE
19	ACTIVE	Write Memory.rsp => AC Send.rsp	ACTIVE
20	ACTIVE	AC Send.cnf(Data) / Command == Write Memory && Request type == Server Response => Write Memory.cnf	ACTIVE
21	ACTIVE	Vendor Command.req => AC Send.req	ACTIVE
22	ACTIVE	AC Send.ind(Data) / Command == Vendor Command && Request type == Client Request => Vendor Command.ind	ACTIVE
23	ACTIVE	Vendor Command.rsp => AC Send.rsp	ACTIVE
24	ACTIVE	AC Send.cnf(Data) / Command == Vendor Command && Request type == Server Response => Vendor Command.cnf	ACTIVE
25	ACTIVE	Distribute Node Info.req => AC Send.req	ACTIVE
26	ACTIVE	AC Send.ind(Data) / Command == Distribute Node Info => Distribute Node Info.ind	ACTIVE
27	ACTIVE	Get Statistics.req => AC Send.req	ACTIVE

N	État actuel	Événement /Condition => Action	État suivant
28	ACTIVE	AC Send.ind(Data) / Command == Get Statistics && Request type == Client Request => Get Statistics.ind	ACTIVE
29	ACTIVE	Get Statistics.rsp => AC Send.rsp	ACTIVE
30	ACTIVE	AC Send.ind(Data) / Command == Get Statistics && Request type == Server Response => Get Statistics.cnf	ACTIVE
31	ACTIVE	Get Node Info Detail.req => AC Send.req	ACTIVE
32	ACTIVE	AC Send.ind(Data) / Command == Get Node Info Detail && Request type == Client Request => Get Node Info Detail.ind	ACTIVE
33	ACTIVE	Get Node Info Detail.rsp => AC Send.rsp	ACTIVE
34	ACTIVE	AC Send.ind(Data) / Command == Get Node Info Detail && Request type == Server Response => Get Node Info Detail.cnf	ACTIVE
35	ACTIVE	AC Data.req => AC Send.req	ACTIVE
36	ACTIVE	AC Data ND.req => AC Send ND.req	ACTIVE
37	ACTIVE	AC Send.ind(Data) / Command == AC Data && Request type == Client Request => AC Data.ind	ACTIVE
38	ACTIVE	AC Send.ind(Data) / Command == AC Send && Request type == Server Response => AC Data.cnf	ACTIVE
39	ACTIVE	AC Data.rsp => AC Send.req	ACTIVE
40	ACTIVE	AC Data ND.rsp => AC Send ND.req	ACTIVE

7.3.2.3 Gestion

La description détaillée du diagramme d'états de gestion (Management) est indiquée dans le Tableau 56.

Tableau 56 – Table d'états de gestion (Management)

N	État actuel	Événement /Condition => Action	État suivant
1	ACTIVE	Get Attribute.req => Get Attribute.cnf	ACTIVE
2	ACTIVE	Set Attribute.req => Set Attribute.cnf	ACTIVE

7.3.2.4 Synchronization

La description détaillée du diagramme d'états de Synchronization (synchronisation) est indiquée dans le Tableau 57.

Tableau 57 – Table d'états de Synchronization (synchronisation)

N	État actuel	Événement /Condition => Action	État suivant
1	ACTIVE	Syncronous Trigger Internal.ind => Syncronous Trigger.ind	ACTIVE

7.3.2.5 Mesure

La description détaillée du diagramme d'états de Measurement (mesure) est indiquée dans le Tableau 58.

Tableau 58 – Table d'états de Measurement (mesure)

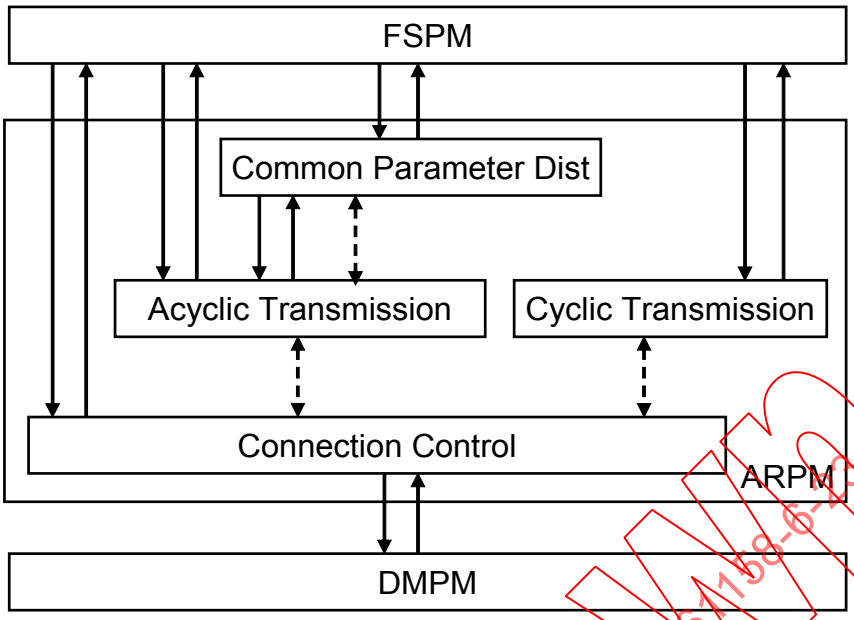
N	État actuel	Événement /Condition => Action	État suivant
1	ACTIVE	Start Measure.req => Start Measure Internal.req	ACTIVE
2	ACTIVE	Start Measure Internal.cnf(DATA) => Start Measure.cnf(DATA)	ACTIVE
3	ACTIVE	Get Offset.req => Get Offset Internal.req	ACTIVE
4	ACTIVE	Get Offset Internal.cnf(offset) => Get Offset.cnf(offset)	ACTIVE

8 Machine de protocole de relations AR (ARPM)

8.1 Type C d'ARPM

8.1.1 Vue d'ensemble

L'ARPM comprend quatre sous-protocoles. La structure de l'ARPM est indiquée à la Figure 20. La ligne continue représente un problème de service et la ligne en pointillés représente une liaison entre les diagrammes d'états de protocole qui utilisent des paramètres et autres.



Légende

Anglais	Français
FSPM	FSPM
Common Parameter Dist	Distribution des paramètres communs
Acyclic Transmission	Transmission acyclique
Cyclic Transmission	Transmission cyclique
Connection Control	Contrôle de connexion
ARPM	ARPM
DMPM	DMPM

Figure 20 – Structure du type C d'ARPM

8.1.2 Acyclic transmission (Transmission acyclique)

8.1.2.1 Définition de primitive

Le FSPM émet un service AC Send.req pour la transmission acyclique. La distribution des paramètres communs émet un service ACParamSend.req pour la transmission acyclique. La transmission acyclique émet un service ACSend.ind pour le FSPM. La transmission acyclique émet un service ACParamSend.ind pour la distribution des paramètres communs.

8.1.2.2 Diagramme d'états de transmission acyclique

La description détaillée du diagramme d'états de transmission acyclique est indiquée dans le Tableau 59.

Tableau 59 – Table d'états de "Acyclic transmission" (transmission acyclique)

N	État actuel	Événement /Condition => Action	État suivant
1	IDLE	/ ACTicket == TRUE => SendCounter = MaxSend	SENDER
2	SENDER	/ SendCounter != 0 && length(RemainingData) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => Data = CreateTransient1-PDU(RemainingData);	SENDER

N	État actuel	Événement /Condition => Action	État suivant
		OutPort.req(Transient1-PDU(Data)); SendCounter = SendCounter-1	
3	SENDER	/ SendCounter != 0 && length(RemainingData) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateTransient1-PDU(RemainingData); InPort.req(Transient1-PDU); SendCounter = SendCounter-1	SENDER
4	SENDER	AC Send.req(Data) / SendCounter != 0 && (OutLoopState == Through || OutLoopState == Loopback) => Create Transient1-PDU(Data); OutPort.req(Transient1-PDU); SendCounter = SendCounter-1	SENDER
5	SENDER	AC Send.req(Data) / SendCounter != 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateTransient1-PDU(Data); InPort.req(Transient1-PDU); SendCounter = SendCounter-1	SENDER
6	SENDER	/ SendCounter == 0 => ACTicket = FALSE	IDLE
7	IDLE	ACReceived(Transient1-PDU) => ReassembleData(Transient1-PDU)	IDLE
8	SENDER	ACReceived(Transient1-PDU) => ReassembleData(Transient1-PDU)	SENDER
9	IDLE	ACReceived(Transient2-PDU) => AC Param Send.ind(Data)	IDLE
10	SENDER	ACReceived(Transient2-PDU) => AC Param Send.ind(Data)	SENDER
11	IDLE	/ ReceivedDataAvailable == TRUE => AC Send.ind(Data); ReceivedDataAvailable = FALSE	IDLE
12	SENDER	/ ReceivedDataAvailable == TRUE => AC Send.ind(Data); ReceivedDataAvailable = FALSE	SENDER

8.1.2.3 Fonctions

Les fonctions activées lors de la transmission acyclique sont indiquées dans le Tableau 60.

Tableau 60 – Fonctions de transmission acyclique

Nom	Description
Length (Longueur)	Demande de taille d'argument
CreateTransient1-PDU	Génération de Transient1-PDU. Si les données de l'argument dépassent 1 464 octets, Transeint1-PDU est généré à l'aide des 1 464 premiers octets. Les données restantes sont considérées comme des RemainingData
ReassembleData	Réassemblage des données divisées reçues; dataId est utilisé pour l'identification des données. Si la taille des données reçues est équivalente à wholeDataSize, il est considéré que le réassemblage est terminé et que ReceivedDataAvailable=TRUE.

8.1.3 Cyclic transmission (Transmission cyclique)

8.1.3.1 Définition de primitive

Le FSPM émet un service CT Update.req pour la transmission cyclique. La transmission cyclique émet un service CT Update.ind pour le FSPM.

8.1.3.2 Diagramme d'états de Cyclic transmission (transmission cyclique)

La description détaillée du diagramme d'états de Cyclic transmission (transmission cyclique) est indiquée dans le Tableau 61.

NOTE L'ordre d'envoi n'est pas spécifié pour BitCM, WordCM, OutCM1, OutCM2, InCM1 ni pour InCM2.

Tableau 61 – Table d'états de Cyclic transmission (transmission cyclique)

N	État actuel	Événement /Condition => Action	État suivant

1	IDLE	/ CTicket == TRUE
		=>
		seqno = 0;
		UpdateWaiting = FALSE;
		if length(BitCM) = 0 then BitCMSent = TRUE;
		if length(WordCM) = 0 then WordCMSent = TRUE;
		if length(OutCM1) = 0 then OutCM1Sent = TRUE;
		if length(OutCM2) = 0 then OutCM2Sent = TRUE;
		if length(InCM1) = 0 then InCM1Sent = TRUE;
		if length(InCM2) = 0 then InCM2Sent = TRUE;
			SENDER

2	Any (Any state)	CT Update.req(Data Type, Offset Address, Size, Data) => Update(Data Type, Offset Address, Size, Data)	Any (no change)
3	Any (Any state)	CT Update.req(Data Type, Offset Address, Size, Data) => Update(Data Type, Offset Address, Size, Data)	Any (no change)
4	SENDER	/ UpdateWaiting == TRUE => Update(Data Type, Offset Address, Size, Data); UpdateWaiting = FALSE	SENDER
5	SENDER	/ BitCMSent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER
6	SENDER	/ BitCMSent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER
7	BitCMSENDER	/ length(RemainingBitCM) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER
8	BitCMSENDER	/ length(RemainingBitCM) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER
9	BitCMSENDER	/ length(RemainingBitCM) == 0 =>	SENDER

N	État actuel	Événement /Condition => Action	État suivant
		BitCMSent = TRUE	
10	BitCMSENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	BitCMSENDER
11	SENDER	/ WordCMSent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataW-PDU(seqno, WordCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	WordCMSENDER
12	SENDER	/ WordCMSent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataW-PDU(seqno, WordCM); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	WordCMSENDER
13	WordCMSENDER	/ length(RemainingWordCM) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataW-PDU(seqno, WordCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	WordCMSENDER
14	WordCMSENDER	/ length(RemainingWordCM) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataW-PDU(seqno, WordCM); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	WordCMSENDER
15	WordCMSENDER	/ length(RemainingWordCM) == 0 => WordCMSent = TRUE	SENDER
16	WordCMSENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	WordCMSENDER
17	SENDER	/ OutCM1Sent != TRUE&& Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER
18	SENDER	/ OutCM1Sent != TRUE&& Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER
19	OutCM1SENDER	/ length(RemainingOutCM1) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER
20	OutCM1SENDER	/ length(RemainingOutCM1) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER
21	OutCM1SENDER	/ length(RemainingOutCM1) == 0	SENDER

N	État actuel	Événement /Condition => Action	État suivant
		=> OutCM1Sent = TRUE	
22	OutCM1SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	OutCM1SENDER
23	SENDER	/ OutCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataOut2-PDU(seqno, OutCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM2SENDER
24	SENDER	/ OutCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataOut2-PDU(seqno, OutCM2); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM2SENDER
25	OutCM2SENDER	/ length(RemainingOutCM2) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataOut2-PDU(seqno, OutCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM2SENDER
26	OutCM2SENDER	/ length(RemainingOutCM2) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataOut2-PDU(seqno, OutCM2); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM2SENDER
27	OutCM2SENDER	/ length(RemainingOutCM2) == 0 => OutCM2Sent = TRUE	SENDER
28	OutCM2SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	OutCM2SENDER
29	SENDER	/ InCM1Sent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER
30	SENDER	/ InCM1Sent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER
31	InCM1SENDER	/ length(RemainingInCM1) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER
32	InCM1SENDER	/ length(RemainingInCM1) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER

N	État actuel	Événement /Condition => Action	État suivant
33	InCM1SENDER	/ length(RemainingInCM1) == 0 => InCM1Sent = TRUE	SENDER
34	InCM1SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	InCM1SENDER
35	SENDER	/ InCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataIn2-PDU(seqno, InCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM2SENDER
36	SENDER	/ InCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataIn2-PDU(seqno, InCM2); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM2SENDER
37	InCM2SENDER	/ length(RemainingInCM2) > 0 && (OutLoopState == Through || OutLoopState == Loopback) => CreateCyclicDataIn2-PDU(seqno, InCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM2SENDER
38	InCM2SENDER	/ length(RemainingInCM2) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataIn2-PDU(seqno, InCM2); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM2SENDER
39	InCM2SENDER	/ length(RemainingInCM2) == 0 => InCM2Sent = TRUE	SENDER
40	InCM2SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	InCM2SENDER
41	Any (n'importe quel état)	CReceived(CyclicDataB-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedBitCM = RetrieveBitCM(CyclicDataB-PDU); CT Update.ind(BitCM, Offset Address, Size,ReceivedBitCM) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (pas de changement)
42	Any (n'importe quel état)	CReceived(CyclicDataW-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedWordCM = RetrieveWordCM(CyclicDataW-PDU); CT Update.ind(WordCM, Offset Address, Size, ReceivedWordCM) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (pas de changement)
43	Any (n'importe quel état)	CReceived(CyclicDataOut1-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedOutCM1 = RetrieveOutCM1(CyclicDataOut1-PDU); CT Update.ind(OutCM1, Offset Address, Size, ReceivedOutCM1) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (pas de changement)
44	Any (n'importe	CReceived(CyclicDataOut2-PDU)	Any (pas de

N	État actuel	Événement /Condition => Action	État suivant
	quel état)	=> Si l'ordre du numéro de séquence est respecté ReceivedOutCM2 = RetrieveOutCM2(CyclicDataOut2-PDU); CT Update.ind(OutCM2, Offset Address, Size, ReceivedOutCM2) Si l'ordre du numéro de séquence n'est pas respecté, rejet	changement)
45	Any (n'importe quel état)	CReceived(CyclicDataIn1-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedInCM1 = RetrieveInCM1(CyclicDataIn1-PDU); CT Update.ind(InCM1, Offset Address, Size, ReceivedInCM1) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (pas de changement)
46	Any (n'importe quel état)	CReceived(CyclicDataIn2-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedInCM2 = RetrieveInCM2(CyclicDataIn2-PDU); CT Update.ind(InCM2, Offset Address, Size, ReceivedInCM2) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (pas de changement)
47	SENDER	/ BitCMSent == TRUE && WordCMSent == TRUE && OutCM1Sent == TRUE && OutCM2Sent == TRUE && InCM1Sent == TRUE && InCM2Sent == TRUE => CTicket = FALSE	IDLE

8.1.3.3 Fonctions

Les fonctions activées pour Cyclic transmission (transmission cyclique) sont indiquées dans le Tableau 62.

Tableau 62 – Fonctions de Cyclic transmission (transmission cyclique)

Nom	Description
Length (Longueur)	Demande de taille donnée par argument.
Update	Mise à jour des données de BitCM, WordCM, OutCM1, OutCM2, InCM1 et InCM2 que la transmission cyclique envoie.
CreateCyclicDataB-PDU	Génération de CyclicDataB-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataB-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingBitCM. Si la dernière PDU est générée, la valeur de Bit7 de seqNumber est 1.
CreateCyclicDataW-PDU	Génération de CyclicDataW-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataW-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingWordCM. Si la dernière PDU est générée, la valeur de Bit7 de seqNumber est 1.
CreateCyclicDataOut1-PDU	Génération de CyclicDataOut1-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataOut1-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingOutCM1. Si la dernière PDU est générée, la valeur de Bit7 de seqNumber est 1.
CreateCyclicDataOut2-PDU	Génération de CyclicDataOu2-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataOut2-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingOutCM2. Si la dernière PDU est générée, la valeur de Bit7 de seqNumber est 1.
CreateCyclicDataIn1-PDU	Génération de CyclicDataIn1-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataIn1-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingInCM1. Si la dernière PDU est générée, la valeur de Bit7 de seqNumber est fixée à 1.
CreateCyclicDataIn2-	Génération de CyclicDataIn2-PDU. Si les données de l'argument dépassent

Nom	Description
PDU	1 468 octets, CyclicDataIn2-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingInCM2. Si la dernière PDU est générée, la valeur de Bit7 de seqNumber est 1.
RetrieveBitCM(PDU)	Récupération de l'adresse de décalage, de la taille et des données BitCM de la PDU.
RetrieveWordCM(PDU)	Récupération de l'adresse de décalage, de la taille et des données WordCM de la PDU.
RetrieveOutCM1(PDU)	Récupération de l'adresse de décalage, de la taille et des données OutCM1 de la PDU.
RetrieveOutCM2(PDU)	Récupération de l'adresse de décalage, de la taille et des données OutCM2 de la PDU.
RetrieveInCM1(PDU)	Récupération de l'adresse de décalage, de la taille et des données InCM1 de la PDU.
RetrieveInCM2(PDU)	Récupération de l'adresse de décalage, de la taille et des données InCM2 de la PDU.

8.1.4 Connection control (Contrôle de connexion)

8.1.4.1 Diagramme d'états de contrôle de connexion

Le diagramme d'états de Connection control (contrôle de connexion) est décrit ci-desus du Tableau 63 au Tableau 76.

Tableau 63 – Diagramme d'états de Connection control (contrôle de connexion) – Initial

N	État actuel	Événement /Condition => Action	État suivant
1	Initial	InPort.ind(Port_State) / Port_State == LinkUp => Start ConnectTimer InPortState = Checking	Connect
2	Initial	OutPort.ind(Port_State) / Port_State == LinkUp => Start ConnectTimer OutPortState = Checking	Connect
3	Initial	InPort.ind(Port_State) / Port_State == LinkDown =>	Initial
4	Initial	OutPort.ind(Port_State) / Port_State == LinkDown =>	Initial
5	Initial	/ NTNTest == TRUE =>	NTNTestMaster

Tableau 64 – Diagramme d'états de Connection control (contrôle de connexion) – Connect

N	État actuel	Événement /Condition => Action	État suivant
1	Connect	InPort.ind(Port_State) / Port_State == LinkUp && InPortState == LinkDown && OutPortState == Checking => InPortState = Checking	Connect
2	Connect	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == Checking && OutPortState == LinkDown => OutPortState = Checking	Connect

N	État actuel	Événement /Condition => Action	État suivant
3	Connect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == NG && OutPortState == Checking => InPortState = LinkDown	Connect
4	Connect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == Checking && OutPortState != OK => InPortState = LinkDown; Stop ConnectTimer.	Initial
5	Connect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == Checking && OutPortState == NG => OutPortState = LinkDown	Connect
6	Connect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == Checking => OutPortState = LinkDown; Stop ConnectTimer	Initial
7	Connect	ConnectTimer times out. / InPortState == Checking && (OutPortState != OK && OutPortState != Checking) => InPort.req(Connect-PDU(PortChoice=In))	Connect
8	Connect	ConnectTimer times out. / (InPortState != OK && InPortState != Checking) && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	Connect
9	Connect	ConnectTimer times out. / InPortState == Checking && OutPortState == Checking => InPort.req(Connect-PDU(PortChoice=In)) OutPort.req(Connect-PDU(PortChoice=Out))	Connect
10	Connect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	Connect
11	Connect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice != Out => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	Connect
12	Connect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Connect
13	Connect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice != In => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	Connect
14	Connect	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && OutPortState == Checking => InPortState = NG	Connect
15	Connect	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == Checking =>	Connect

N	État actuel	Événement /Condition => Action	État suivant
		OutPortState = NG	
16	Connect	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK => InPortState = OK; Start ScanTimer.	Scan
17	Connect	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == Checking && OutPortState != Checking => InPortState = NG; Stop ConnectTimer.	Initial
18	Connect	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK => OutPortState = OK; Start SendScanTimer.	Scan
19	Connect	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState != Checking && OutPortState == Checking => OutPortState = NG; Stop ConnectTimer.	Initial
20	Connect	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState != OK => InPortState = OK; Start ScanTimer.	Scan
21	Connect	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == Checking => OutPortState = OK; Start ScanTimer.	Scan
22	Connect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
23	Connect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Tableau 65 – Diagramme d'états de Connection control (contrôle de connexion) – Scan

N	État actuel	Événement /Condition => Action	État suivant
1	Scan	InPort.ind(Port_State) / Port_State == LinkUp && InPortState == LinkDown && OutPortState == OK && StopConnectFlag != ON => Start ConnectTimer; InPortState = Checking	Scan
2	Scan	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState == LinkDown && StopConnectFlag != ON => Start ConnectTimer; OutPortState = Checking	Scan
3	Scan	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK	Initial

N	État actuel	Événement /Condition => Action	État suivant
		=> InPortState = LinkDown	
4	Scan	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	Scan	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortStart = LinkDown	Initial
6	Scan	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	Scan	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	Scan
8	Scan	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	Scan
9	Scan	ScanTimer times out. / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU(scanState = Through))	Scan
10	Scan	ScanTimer times out. / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU(scanState = InLoopback))	Scan
11	Scan	ScanTimer times out. / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU(scanState = OutLoopback))	Scan
12	Scan	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	Scan
13	Scan	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	Scan
14	Scan	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Scan
15	Scan	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	Scan
16	Scan	InPort.ind(ConnectAck-PDU(PortCheckResult)	Scan

N	État actuel	Événement /Condition => Action	État suivant
		/ PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	
17	Scan	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	Scan
18	Scan	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
19	Scan	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	Scan
20	Scan	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState != OK && Latest(Scan-PDU) == TRUE => InPort.req(Scan-PDU)	Scan
21	Scan	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => OutPort.req(Scan-PDU)	Scan
22	Scan	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	Scan
23	Scan	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState != OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => OutPort.req(Scan-PDU)	Scan
24	Scan	InPort.ind(source_address, Scan-PDU(scanState)) / source_address == my address && InPortState == OK => Start DetectScanTimer.	ScanWait
25	Scan	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
26	Scan	OutPort.ind(source_address, Scan-PDU(scanState)) / source_address == my address && InPortState != OK && OutPortState == OK => Start DetectScanTimer.	ScanWait
27	Scan	OutPort.ind(source_address, Scan-PDU) / source_address == my address && InPortState == OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => InPort.req(Scan-PDU)	Scan
28	Scan	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan

N	État actuel	Événement /Condition => Action	État suivant
29	Scan	InPort.ind(PDU) / PDU != Connect-PDU && PDU != ConnectAck-PDU && PDU != Scan-PDU =>	Scan
30	Scan	OutPort.ind(PDU) / PDU != Connect-PDU && PDU != ConnectAck-PDU && PDU != Scan-PDU =>	Scan
31	Scan	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
32	Scan	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Tableau 66 – Diagramme d'états de Connection control (contrôle de connexion) – ScanWait

N	État actuel	Événement /Condition => Action	État suivant
1	ScanWait	InPort.ind(Port_State) / Port_State == LinkUp && InPortState == LinkDown && OutPortState == OK && StopConnectFlag != ON => Start ConnectTimer; InPortState = Checking	ScanWait
2	ScanWait	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState == LinkDown && StopConnectFlag != ON => Start ConnectTimer; OutPortState = Checking	ScanWait
3	ScanWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	ScanWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	ScanWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	ScanWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState !=LinkDown => OutPortState = LinkDown	Scan
7	ScanWait	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	ScanWait
8	ScanWait	ConnectTimer times out. / InPortState == OK && OutPortState == Checking =>	ScanWait

N	État actuel	Événement /Condition => Action	État suivant
		OutPort.req(Connect-PDU(PortChoice=Out))	
9	ScanWait	DetectScanTimer times out. / InPortState == OK || OutPortState == OK => Start CollectTimer.	Collect
10	ScanWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	ScanWait
11	ScanWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	ScanWait
12	ScanWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	ScanWait
13	ScanWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	ScanWait
14	ScanWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
15	ScanWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	ScanWait
16	ScanWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
17	ScanWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	ScanWait
18	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address == my address && (InPortState == OK || InPortState == Checking) => Restart DetectScanTimer.	ScanWait
19	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => Restart DetectScanTimer; OutPort.req(Scan-PDU)	ScanWait
20	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address != my address &&	ScanWait

N	État actuel	Événement /Condition => Action	État suivant
		InPortState == OK && OutPortState != OK && Latest(Scan-PDU) == TRUE => Restart DetectScanTimer; InPort.req(Scan-PDU)	
21	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && Latest(Scan-PDU) != TRUE => Restart DetectScanTimer;	ScanWait
22	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address == my address && (InPortState == OK || OutPortState == OK) => Restart DetectScanTimer.	ScanWait
23	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState != OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => Restart DetectScanTimer; OutPort.req(Scan-PDU)	ScanWait
24	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState != OK && OutPortState == OK && Latest(Scan-PDU) != TRUE => Restart DetectScanTimer.	ScanWait
25	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address == my address && InPortState == OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => InPort.req(Scan-PDU)	ScanWait
26	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	ScanWait
27	ScanWait	InPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
28	ScanWait	OutPort.ind(source_address, Scan-PDU) / source_address != my address && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
29	ScanWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
30	ScanWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Tableau 67 – Diagramme d'états de Connection control (contrôle de connexion) – Collect

N	État actuel	Événement /Condition => Action	État suivant
1	Collect	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	Collect
2	Collect	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	Collect
3	Collect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	Collect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown;	Scan
5	Collect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	Collect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	Collect	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	Collect
8	Collect	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	Collect
9	Collect	CollectTimer times out. / InPortState == OK && OutPortState == OK => OutPort.req(Collect-PDU(InLoopState = Through, OutLoopState = Through))	Collect
10	Collect	CollectTimer times out. / InPortState == OK && OutPortState != OK => InPort.req(Collect-PDU(InLoopState = Loopback, OutLoopState =OutPortState))	Collect
11	Collect	CollectTimer times out. / InPortState != OK && OutPortState == OK => OutPort.req(Collect-PDU(InLoopState = InPortState, OutLoopState = Loopback))	Collect
12	Collect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	Collect
13	Collect	InPort.ind(Connect-PDU(PortChoice))	Collect

N	État actuel	Événement /Condition => Action	État suivant
		/ PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	
14	Collect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Collect
15	Collect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK || OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	Collect
16	Collect	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
17	Collect	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	Collect
18	Collect	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
19	Collect	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	Collect
20	Collect	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
21	Collect	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	Collect	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
23	Collect	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
24	Collect	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
25	Collect	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	Collect