IEC 61158-6-23

Edition 3.0 2023-03

INTERNATIONAL STANDARD

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

IEC 61158-6-23:2023-03(en)

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

Tel.: +41 22 919 02 11
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 corrigendum or an amendment might have been published.

IEC publications search - webstore.iec.ch/advsearchform
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 It also gives information on projects, replaced and withdrawn publications.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 once a month by email.

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: sales@iec.ch.

IEC Products & Services Portal - products.iec.ch
Discover our powerful search engine and read freely all the publications previews. With a subscription you will always have access to up to date content tailored to your needs.

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 300 terminological entries in English and French, with equivalent terms in 19 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC 61158-6-23

Edition 3.0 2023-03

INTERNATIONAL STANDARD

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

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-8322-6641-0

Warning! Make sure that you obtained this publication from an authorized distributor.

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.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

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 the IEC 61784-1 series and the IEC 61784-2 series.

IEC 61158-6-23 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation. It is an International Standard.

This third edition cancels and replaces the second edition published in 2019. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

a) addition of the transmission extended mode and related attribute (see 3.2.28, 4.1.9, 4.4, 5.2.9.2, and 5.3);
b) update of Table 4, Table 5, Table 16 and Table 48.

The text of this International Standard is based on the following documents:

Draft	Report on voting
65C/1204/FDIS	65C/1245/RVD

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

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/publications.

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 document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

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

INTRODUCTION

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

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

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent. IEC takes no position concerning the evidence, validity, and scope of this patent right.

The holder of this patent right has assured IEC that s/he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with IEC. Information may be obtained from the patent database available at http://patents.iec.ch.

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

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 part of IEC 61158 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 document 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.

The purpose of this document 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 document 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 document to provide access to the FAL to control certain aspects of its operation.

1.2 Specifications

The principal objective of this document 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 the IEC 61158-6 series.

1.3 Conformance

This document 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 document. Instead, conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. 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 series and IEC 61784-2 series 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 61158-1:2023, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-5-23:2023, *Industrial communication networks – Fieldbus specifications – Part 5-23: Application layer service definition – Type 23 elements*

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

ISO/IEC 8822, *Information technology – Open Systems Interconnection – Presentation service definition*

ISO/IEC 8824-1, *Information technology – Abstract Syntax Notation One (ASN.1) – Part 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*

IEEE Std 802.1AS, *Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*

IEEE Std 1588, *Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*

IETF RFC 768, J. Postel, *User Datagram Protocol*, August 1980, available at https://www.rfc-editor.org/info/rfc768 [viewed 2022-02-18]

IETF RFC 791, J. Postel, *Internet Protocol*, September 1981, available at https://www.rfc-editor.org/info/rfc791 [viewed 2022-02-18]

3 Terms, definitions, symbols, abbreviated terms and conventions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at https://www.electropedia.org/
- ISO Online browsing platform: available at https://www.iso.org/obp

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) 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

3.1.3 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 Additional Type 23 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 of a network

3.2.23
synchronization manager
single node in a master station role 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
single node in a master station role per network that performs token passing management

3.2.28
transmission extended mode
operation mode to extend the number of link bit and link word.

3.2.29
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 document.

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 document 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"
||	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,
    availableFuncs           AvailableFunc,
    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..1 482),
    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 ..1 466)),
    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
AvailableFuncs::=Unsigned16
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
```

4.5 FALPDU type T abstract syntax

4.5.1 Basic abstract syntax

The definitions of FALPDU are shown below.

```
FAL-PDU::= CHOICE {
    cyclic-PDU                         Cyclic-PDU,
    acyclic-PDU                        Acyclic-PDU,
    ptp-PDU                            PTP-PDU,
    ip-PDU                             IP-PDU
}

Cyclic-PDU::= CHOICE {
    cyclicM-PDU                        CyclicM-PDU,
    cyclicS-PDU                        CyclicS-PDU,
    cyclicMs-PDU                       CyclicMs-PDU,
    cyclicSs-PDU                       CyclicSs-PDU
}

Acyclic-PDU::= CHOICE {
    acyclicPriority-PDU                AcyclicPriority-PDU,
    acyclicDetection-PDU               AcyclicDetection-PDU,
    acyclicDetectionAck-PDU            AcyclicDetectionAck-PDU,
    acyclicTestData-PDU                AcyclicTestData-PDU,
    acyclicTestDataAck-PDU             AcyclicTestDataAck-PDU,
    acyclicData-PDU                    AcyclicData-PDU
}
```

```
PTP-PDU::= CHOICE {
        ptpSync-PDU                             PtpSyn-PDU,
        ptpPdelayReq-PDU                        PtpPdelayReq-PDU,
        ptpPdelayResp-PDU                       PtpPdelayResp-PDU,
        ptpFollowUp -PDU                        PtpFollowUp-PDU,
        ptpPdelayRespFollowUp-PDU               PtpPdelayRespFollowUp-PDU,
        ptpAnnounce -PDU                        PtpAnnounce-PDU
        ptpdelayReq-PDU                         PtpdelayReq-PDU,
        ptpdelayResp-PDU                        PtpdelayResp-PDU,
        ptpdelayRespFollowUp-PDU                PtpdelayRespFollowUp-PDU,
}

IP-PDU::= SEQUENCE {
        ipHeader                                IPHeader,
        updHeader                               UDPHeader,
        udpData                                 CHOICE {
                slmp-networkConfigMain-PDU              Slmp-networkConfigMain-PDU,
                slmp-networkConfigTslt-PDU              Slmp-networkConfigTslt-PDU,
                slmp-notification-PDU                   Slmp-notification-PDU,
                slmp-masterConfig-PDU                   Slmp-masterConfig-PDU,
                slmp-slaveConfig -PDU                   Slmp-slaveConfig-PDU,
                slmp-cyclicConfigMain-PDU               Slmp-cyclicConfigMain-PDU,
                slmp-cyclicConfigTrnSubPayload-PDU      Slmp-cyclicConfigTrnSubPayload-PDU,
                slmp-cyclicConfigRcvSubPayload-PDU      Slmp-cyclicConfigRcvSubPayload-PDU,
                slmp-cyclicConfigRcvSrcInfo-PDU         Slmp-cyclicConfigRcvSrcInfo-PDU,
        }
}
```

The header used in each PDU is shown below.

```
CyclicMSHeader::= SEQUENCE {
        fType                                   FrameType,
        cyclicNo                                CyclicNo,
        varField                                CHOICE {
                vField[0]                               SEQUENCE {
                        sa                                      SA
                },
                vField[1]                               SEQUENCE {
                        da                                      DA
                },
        },
        reserved                                OCTET STRING (SIZE (2))
}
```

```
CyclicMsSsHeader::= SEQUENCE {
        frameType                                           FrameType,
        cyclicNo                                            CyclicNo,
        varField                                            CHOICE {
                vField[0]                                   SEQUENCE {
                        sa                                  SA
                },
                vField[1]                                   SEQUENCE {
                        da                                  DA
                },
        },
        reserved                                            OCTET STRING (SIZE (2)),
        hec                                                 HEC
}

AcyclicPriorityHeader::= SEQUENCE {
        frameType                                           FrameType,
        reserved                                            OCTET STRING (SIZE (9)),
        hec                                                 HEC
}

AcyclicDetectionHeader::= SEQUENCE {
        frameType                                           FrameType,
        reserved                                            OCTET STRING (SIZE (1))
}

AcyclicTestDataHeader::= SEQUENCE {
        frameType                                           FrameType,
        dataType                                            DataType,
        persPriority                                        PersPriority,
        nodeType                                            NodeType,
        srcNodeNumber                                       NodeNumber,
        protocolVerType                                     ProtocolVerType,
        reserved                                            OCTET STRING (SIZE (1)),
        hec                                                 HEC
}

AcyclicDataHeader::= SEQUENCE {
        frameType                                           FrameType,
        reserved                                            OCTET STRING (SIZE (1)),
        da                                                  DA,
        reserved1                                           OCTET STRING (SIZE (2)),
```

4.5.2 CyclicM-PDU

```
CyclicM-PDU::= SEQUENCE {
    cyclicMSHeader              CyclicMSHeader,
    subPayloadHeader            SEQUENCE {
        da                      OCTET STRING (SIZE (2)),
        commInfo                Unsigned16,
        txAsynInfo              Unsigned8,
    },
    subPayloadData              SEQUENCE {
        seqNo                   Unsigned16,
        diagnosisData           Unsigned32,
        reserved1               OCTET STRING (SIZE (1)),
        memoryAddress           Unsigned32,
        applicationData         OCTET STRING(SIZE(1..1 454))
    }
}
```


4.5.3 CyclicS-PDU

```
CyclicS-PDU::= SEQUENCE {
    cyclicMSHeader              CyclicMSHeader,
    subPayloadHeader            SEQUENCE {
        da                      OCTET STRING (SIZE (2)),
        commInfo                Unsigned16,
        txAsynInfo              Unsigned8,
    },
    subPayloadData              SEQUENCE {
        seqNo                   Unsigned16,
        diagnosisData           Unsigned32,
        reserved1               OCTET STRING (SIZE (1)),
        memoryAddress           Unsigned32,
        applicationData         OCTET STRING(SIZE(1..1 454))
    }
}
```

4.5.4 CyclicMs-PDU

```
CyclicMs-PDU::= SEQUENCE {
    cyclicMsSsHeader            CyclicMsSsHeader,
    subPayloadBlock             SEQUENCE OF SubPayloadSet1,
    eos                         EOS
}
```

```
SubPayloadSet1::= SEQUENCE {
        subPayloadHeader                SEQUENCE {
                da                      OCTET STRING (SIZE (2)),
                commInfo                Unsigned16,
                txAsynInfo              Unsigned8,
        },
        subPayloadData                  SEQUENCE {
                seqNo                   Unsigned16,
                diagnosisData           Unsigned32,
                reserved1               OCTET STRING (SIZE (1)),
                memoryAddress           Unsigned32,
                applicationData         OCTET STRING(SIZE(1..1 454)),
        }
}
```

4.5.5 CyclicSs-PDU

```
CyclicSs-PDU::= SEQUENCE {
        cyclicMsSsHeader                CyclicMsSsHeader,
        subPayloadBlock                 SEQUENCE OF SubPayloadSet2,
        eos                             EOS
}

SubPayloadSet2::= SEQUENCE {
        subPayloadHeader                SEQUENCE {
                sa                      OCTET STRING (SIZE (2)),
                commInfo                Unsigned16,
                txAsynInfo              Unsigned8,
        },
        subPayloadCheckData             SEQUENCE {
                seqNo                   Unsigned16,
                diagnosisData           Unsigned32,
                reserved1               OCTET STRING (SIZE (1)),
                memoryAddress           Unsigned32,
                applicationData         OCTET STRING(SIZE(1..1 454)),
                sdCRC                   Unsigned32
        }
}
```

4.5.6 AcyclicPriority-PDU

```
AcyclicPriority-PDU::= SEQUENCE {
    acyclicPriorityHeader          AcyclicPriorityHeader,
    acyclicPriorityData            SEQUENCE {
        srcMAC                     OCTET STRING (SIZE (6)),
        mngPriority                Unsigned8,
        reserved1                  OCTET STRING (SIZE (1)),
        mngMAC                     OCTET STRING (SIZE (6)),
        hopCount                   Unsigned16,
        reqRes                     Unsigned8,
        reserved2                  OCTET STRING (SIZE (11))
    },
    dcs                            Unsigned32
}
```

4.5.7 AcyclicDetection-PDU

```
AcyclicDetection-PDU::= SEQUENCE {
    acyclicDetectionHeader         AcyclicDetectionHeader,
    acyclicDetectionData           SEQUENCE {
        reserved1                  OCTET STRING (SIZE (2)),
        protocolVer                Unsigned8,
        reserved2                  OCTET STRING (SIZE (1)),
        mngMAC                     OCTET STRING (SIZE (6)),
        previousNodeMAC            OCTET STRING (SIZE (6)),
        previousNodePort           Unsigned8,
        optionFlag                 Unsigned8,
        hopCount                   Unsigned16,
        ipAdd                      OCTET STRING (SIZE (25)),
        sendInfo                   Unsigned8,
        reserved3                  OCTET STRING (SIZE (10))
    }
}
```

4.5.8 AcyclicDetectionAck-PDU

```
AcyclicDetectionAck-PDU::= SEQUENCE {
    acyclicDetectionHeader              AcyclicDetectionHeader,
    acyclicDetectionAckData             SEQUENCE {
        nodeType                        Unsigned8,
        protocolVer                     Unsigned8,
        ipAddressFourthOctet            Unsigned16,
        srcMAC                          OCTET STRING (SIZE (6)),
        reserved1                       OCTET STRING (SIZE (2)),
        previousNodeMAC                 OCTET STRING (SIZE (6)),
        previousNodePort                Unsigned8
        detectRcvPort                   Unsigned8,
        myPort                          Unsigned8,
        reserved2                       OCTET STRING (SIZE (3)),
        myPortLinkStatus                Unsigned96,
        myPortFilterStatus              Unsigned96,
        currentManager                  OCTET STRING (SIZE (6)),
        reserved3                       OCTET STRING (SIZE (2)),
        ipAdd                           OCTET STRING (SIZE (45)),
        performance                     Unsigned8,
        reserved4                       OCTET STRING (SIZE (2)),
        gmPriority                      Unsigned48,
        synctype                        Unsigned8,
        reserved5                       OCTET STRING (SIZE (1)),
        pdelayResTime                   Unsigned8,
        delaySetTime                    Unsigned8,
        announceRelayTime               Unsigned8,
        reserved6                       OCTET STRING (SIZE (1)),
        deviceVer                       OCTET STRING (SIZE (2)),
        vendorCode                      Unsigned16,
        modelCode                       OCTET STRING (SIZE (4)),
        expansionmodelCode              OCTET STRING (SIZE (2)),
        deviceType                      Unsigned16,
        memoryAddress                   Unsigned224,
        cyclicSize                      Unsigned112,
        reserved7                       OCTET STRING (SIZE (2)),
        function                        Unsigned8,
        optionInfo                      Unsigned8,
        stationMode                     Unsigned16,
        blank                           OCTET STRING (SIZE (4))
    }
}
```

4.5.9 AcyclicTestDataHeader

```
AcyclicTestDataHeader::= SEQUENCE {
    acyclicTestDataHeader           AcyclicTestDataHeader,
    acyclicTestDataData             SEQUENCE {
        macAddr                     MACAddress,
        srcPort                     PortNumber,
        reserved1                   OCTET STRING (SIZE (1)),
        send Inf                    SendInfo,
        vField[0]                   SEQUENCE {
            reserved2               OCTET STRING (SIZE (1)),
            hopCount                HopCount,
            ipv4Address             Ipv4Address,
            ipv4Subnet              Ipv4Subnet,
            reserved3               OCTET STRING (SIZE (8)) ,
        },
        vField[1]                   SEQUENCE {
            ipv6Subnet              Ipv6Subnet,
            hopCount                HopCount,
            ipv6Address             Ipv6Address,
        },
        dcs                         DCS
    }
}
```

4.5.10 AcyclicTestDataHeader

```
AcyclicTestDataHeader::= SEQUENCE {
    acyclicTestDataHeader          AcyclicTestDataHeader,
    acyclicTestDataAckData         SEQUENCE {
        macAddr                    MACAddress,
        srcPort                    PortNumber,
        rcvPort                    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
    }
}
```

4.5.11 AcyclicData-PDU

```
AcyclicData-PDU::= SEQUENCE {
    acyclicDataHeader              AcyclicDataHeader,
    data                           SEQUENCE OF OctetString
}
```


4.5.12 PtpSync-PDU

Same as Sync as defined in IEEE Std 802.1AS and IEEE Std 1588.

4.5.13 PtpPdelayReq-PDU

Same as Pdelay_Req as defined in IEEE Std 802.1AS and IEEE Std 1588.

4.5.14 PtpPdelayResp-PDU

Same as Pdelay_Resp as defined in IEEE Std 802.1AS and IEEE Std 1588.

4.5.15 PtpFollowUp-PDU

Same as Follow_Up as defined in IEEE Std 802.1AS and IEEE Std 1588.

4.5.16 PtpPdelayRespFollowUp-PDU

Same as Pdelay_Resp_Follow_Up as defined in IEEE Std 802.1AS and IEEE Std 1588.

4.5.17 PtpAnnounce-PDU

Same as Announce as defined in IEEE Std 802.1AS and IEEE Std 1588.

4.5.18 SlmpIPAddressSet-PDU

```
SlmpIPAddressSet-PDU::= CHOICE {
    reqIPAddressSet          ReqIPAddressSet,
    resIPAddressSet          ResIPAddressSet
}
```

NOTE Refer to the SLMP Specification for ResIPAddressSet and ResIPAddressSet g.

4.5.19 SlmpNetworkConfigMain-PDU

```
SlmpNetworkConfigMain-PDU::= CHOICE {
    reqNetworkConfigMain          ReqNetworkConfigMain,
    resNetworkConfigMain          ResNetworkConfigMain
}
```

NOTE Refer to the SLMP Specification for ReqNetworkConfigMain and ResNetworkConfigMain.

4.5.20 SlmpNetworkConfigTslt-PDU

```
SlmpNetworkConfigTslt-PDU::= CHOICE {
    reqNetworkConfigTslt          ReqNetworkConfigTslt,
    resNetworkConfigTslt          ResNetworkConfigTslt
}
```

NOTE Refer to the SLMP Specification for ReqNetworkConfigTslt and ResNetworkConfigTslt.

4.5.21 SlmpNotification-PDU

```
SlmpNotification-PDU::= CHOICE {
    reqNotification          ReqNotification,
    resNotification          ResNotification
}
```

NOTE Refer to the SLMP Specification for ReqNotification and ResNotification.

4.5.22 SlmpMasterConfig-PDU

SlmpMasterConfig-PDU::= CHOICE {

reqMasterConfig	ReqMasterConfig,
resMasterConfig	ResMasterConfig

}

NOTE Refer to the SLMP Specification for ReqMasterConfig and ResMasterConfig.

4.5.23 SlmpSlaveConfig-PDU

SlmpSlaveConfig-PDU::= CHOICE {

reqSlaveConfig	ReqSlaveConfig,
resSlaveConfig	ResSlaveConfig

}

NOTE Refer to the SLMP Specification for ReqSlaveConfig and ResSlaveConfig.

4.5.24 SlmpCyclicConfigMain-PDU

SlmpCyclicConfigMain-PDU::= CHOICE {

reqCyclicConfigMain	ReqCyclicConfigMain,
resCyclicConfigMain	ResCyclicConfigMain

}

NOTE Refer to the SLMP Specification for ReqCyclicConfigMain and ResCyclicConfigMain.

4.5.25 SlmpCyclicConfigTrnSubPayload-PDU

SlmpCyclicConfigTrnSubPayload-PDU::= CHOICE {

reqCyclicConfigTrnSubPayload	ReqCyclicConfigTrnSubPayload,
resCyclicConfigTrnSubPayload	ResCyclicConfigTrnSubPayload

}

NOTE Refer to the SLMP Specification for ReqCyclicConfigTrnSubPayload and ResCyclicConfigTrnSubPayload.

4.5.26 SlmpCyclicConfigRcvSubPayload-PDU

SlmpCyclicConfigRcvSubPayload-PDU::= CHOICE {

reqCyclicConfigRcvSubPayload	ReqCyclicConfigRcvSubPayload,
resCyclicConfigRcvSubPayload	ResCyclicConfigRcvSubPayload

}

NOTE Refer to the SLMP Specification for ReqCyclicConfigRcvSubPayload and ResCyclicConfigRcvSubPayload.

4.5.27 SlmpCyclicConfigRcvSrcInfo-PDU

```
SlmpCyclicConfigRcvSrcInfo-PDU::= CHOICE {
    reqCyclicConfigRcvSrcInfo        ReqCyclicConfigRcvSrcInfo,
    resCyclicConfigRcvSrcInfo        ResCyclicConfigRcvSrcInfo
}
```

NOTE Refer to the SLMP Specification for ReqCyclicConfigRcvSrcInfo and ResCyclicConfigRcvSrcInfo.

4.5.28 SlmpLinkDevicePrmWrite-PDU

```
SlmpLinkDevicePrmWrite-PDU::= CHOICE {
    reqLinkDevicePrmWrite        ReqLinkDevicePrmWrite,
    resLinkDevicePrmWrite        ResLinkDevicePrmWrite
}
```

NOTE Refer to the SLMP Specification for ReqLinkDevicePrmWrite and ResLinkDevicePrmWrite.

4.5.29 SlmpLinkDevicePrmWriteCheckRequest-PDU

```
SlmpLinkDevicePrmWriteCheckRequest-PDU::= CHOICE {
    reqLinkDevicePrmWriteCheckRequest        ReqLinkDevicePrmWriteCheckReq,
    resLinkDevicePrmWriteCheckRequest        ResLinkDevicePrmWriteCheckReq
}
```

NOTE Refer to the SLMP Specification for ReqLinkDevicePrmWriteCheckReq and ResLinkDevicePrmWriteCheckReq.

4.5.30 SlmpLinkDevicePrmWriteCheckResponse-PDU

```
SlmpLinkDevicePrmWriteCheckResponse-PDU::= CHOICE {
    reqLinkDevicePrmWritecheckResp        ReqLinkDevicePrmWriteCheckResp,
    resLinkDevicePrmWritecheckResp        ResLinkDevicePrmWriteCheckResp
}                                         }
```

NOTE Refer to the SLMP Specification for ReqLinkDevicePrmWritecheckResp and ResLinkDevicePrmWritecheckResp.

4.5.31 SlmpNMTStateUpload-PDU

```
SlmpNMTStateUpload-PDU::= CHOICE {
    reqNMTStateUpload        ReqNMTStateUpload,
    resNMTStateUpload        ResNMTStateUpload
}
```

NOTE Refer to the SLMP Specification for ReqNMTStateUpload and ResNMTStateUpload.

4.5.32 SlmpNMTStateDownload-PDU

```
SlmpNMTStateDownload-PDU::= CHOICE {
    reqNMTStateDownload          ReqNMTStateDownload,
    resNMTStateDownload          ResNMTStateDownload
}
```

NOTE Refer to the SLMP Specification for ReqNMTStateDownload and ResNMTStateDownload.

4.5.33 SlmpReadObject-PDU

```
SlmpReadObject-PDU::= CHOICE {
    reqReadObject                ReqReadObject,
    resReadObject                ResReadObject
}
```

NOTE Refer to the SLMP Specification for ReqReadObject and ResReadObject.

4.5.34 SlmpWriteObject-PDU

```
SlmpWriteObject-PDU::= CHOICE {
    reqWriteObject               ReqWriteObject,
    resWriteObject               ResWriteObject
}
```

NOTE Refer to the SLMP Specification for ReqWriteObject and ResWriteObject.

4.5.35 SlmpObjectSubIDReadBlock-PDU

```
SlmpObjectSubIDReadBlock-PDU::= CHOICE {
    reqObjectSubIDReadBlock      ReqObjectSubIDReadBlock,
    resObjectSubIDReadBlock      ResObjectSubIDReadBlock
}
```

NOTE Refer to the SLMP Specification for ReqObjectSubIDReadBlock and ResObjectSubIDReadBlock.

4.5.36 SlmpObjectSubIDWriteBlock-PDU

```
SlmpObjectSubIDWriteBlock-PDU::= CHOICE {
    reqObjectSubIDWriteBlock     ReqObjectSubIDWriteBlock,
    resObjectSubIDWriteBlock     ResObjectSubIDWriteBlock
}
```

NOTE Refer to SLMP Specification for ReqObjectSubIDWriteBlock and ResObjectSubIDWriteBlock.

4.5.37 SlmpGetODList-PDU

```
SlmpGetODList-PDU::= CHOICE {
    reqGetODList                 ReqGetODList,
    resGetODList                 ResGetODList
}
```

NOTE Refer to the SLMP Specification for ReqGetODList and ResGetODList.

4.5.38 SlmpGetObjectDescription-PDU

```
SlmpGetObjectDescription-PDU::= CHOICE {
    reqGetObjectDescription        ReqGetObjectDescription,
    resGetObjectDescription        ResGetObjectDescription
}
```

NOTE Refer to the SLMP Specification for ReqGetObjectDescription and ResGetObjectDescription.

4.5.39 SlmpGetEntryDescription-PDU

```
SlmpGetEntryDescription-PDU::= CHOICE {
    reqGetEntryDescription        ReqGetEntryDescription,
    resGetEntryDescription        ResGetEntryDescription
}
```

NOTE Refer to the SLMP Specification for ReqGetEntryDescription and ResGetEntryDescription.

4.5.40 SlmpStopOwnStationCyclic-PDU

```
SlmpStopOwnStationCyclic-PDU::= CHOICE {
    reqStopOwnStationCyclic        ReqStopOwnStationCyclic,
    resStopOwnStationCyclic        ResStopOwnStationCyclic
}
```

NOTE Refer to the SLMP Specification for ReqStopOwnStationCyclic and ResStopOwnStationCyclic.

4.5.41 SlmpStartOwnStationCyclic-PDU

```
SlmpStartOwnStationCyclic-PDU::= CHOICE {
    reqStartOwnStationCyclic        ReqStartOwnStationCyclic,
    resStartOwnStationCyclic        ResStartOwnStationCyclic
}
```

NOTE Refer to the SLMP Specification for ReqStartOwnStationCyclic and ResStartOwnStationCyclic.

4.5.42 SlmpStopOtherStationsCyclic-PDU

```
SlmpStopOtherStationsCyclic-PDU::= CHOICE {
    reqStopOtherStationsCyclic        ReqStopOtherStationsCyclic,
    resStopOtherStationsCyclic        ResStopOtherStationsCyclic
}
```

NOTE Refer to the SLMP Specification for ReqStopOtherStationsCyclic and ResStopOtherStationsCyclic.

4.5.43 SlmpStartOtherStationsCyclic-PDU

```
SlmpStartOtherStationsCyclic-PDU::= CHOICE {
    reqStartOtherStationsCyclic        ReqStartOtherStationsCyclic,
    resStartOtherStationsCyclic        ResStartOtherStationsCyclic
}
```

NOTE Refer to the SLMP Specification for ReqStartOtherStationsCyclic and ResStartOtherStationsCyclic.

4.5.44 SlmpAllParameterGet-PDU

```
SlmpAllParameterGet-PDU::= CHOICE {
    reqAllParameterGet                ReqAllParameterGet,
    resAllParameterGet                ResAllParameterGet
}
```

NOTE Refer to the SLMP Specification for ReqAllParameterGet and ResAllParameterGet.

4.5.45 SlmpParameterGet-PDU

```
SlmpParameterGet-PDU::= CHOICE {
    reqParameterGet                   ReqParameterGet,
    resParameterGet                   ResParameterGet
}
```

NOTE Refer to the SLMP Specification for ReqParameterGet and ResParameterGet.

4.5.46 SlmpAllParameterSizeGet-PDU

```
SlmpAllParameterSizeGet-PDU::= CHOICE {
    reqAllParameterSizeGet            ReqAllParameterSizeGet,
    resAllParameterSizeGet            ResAllParameterSizeGet
}
```

NOTE Refer to the SLMP Specification for ReqAllParameterSizeGet and ResAllParameterSizeGet.

4.5.47 SlmpParameterSizeGet-PDU

```
SlmpParameterSizeGet-PDU::= CHOICE {
    reqParameterSizeGet               ReqParameterSizeGet,
    resParameterSizeGet               ResParameterSizeGet
}
```

NOTE Refer to the SLMP Specification for ReqParameterSizeGet and ResParameterSizeGet.

4.5.48 SlmpStationSubIDListGet-PDU

```
SlmpStationSubIDListGet-PDU::= CHOICE {
    reqStationSubIDListGet            ReqStationSubIDListGet,
    resStationSubIDListGet            ResStationSubIDListGet
}
```

NOTE Refer to the SLMP Specification for ReqStationSubIDListGet and ResStationSubIDListGet.

4.5.49 SlmpDeviceIdentificationInfoGet-PDU

```
SlmpDeviceIdentificationInfoGet-PDU::= CHOICE {
    reqDeviceIdentificationInfoGet    ReqDeviceIdentificationInfoGet,
    resDeviceIdentificationInfoGet    ResDeviceIdentificationInfoGet
}
```

NOTE Refer to the SLMP Specification for ReqDeviceIdentificationInfoGet and ResDeviceIdentificationInfoGet.

4.5.50 SlmpDataMonitoring-PDU

```
SlmpDataMonitoring-PDU::= CHOICE {
        reqDataMonitoring                    ReqDataMonitoring,
        resDataMonitoring                    ResDataMonitoring
}
```

NOTE Refer to the SLMP Specification for ReqDataMonitoring and ResDataMonitoring.

4.5.51 SlmpAllParameterSet-PDU

```
SlmpAllParameterSet-PDU::= CHOICE {
        reqAllParameterSet                   ReqAllParameterSet,
        resAllParameterSet                   ResAllParameterSet
}
```

NOTE Refer to the SLMP Specification for ReqAllParameterSet and ResAllParameterSet.

4.5.52 SlmpParameterSet-PDU

```
SlmpParameterSet-PDU::= CHOICE {
        reqParameterSet                      ReqParameterSet,
        resParameterSet                      ResParameterSet
}
```

NOTE Refer to the SLMP Specification for ReqParameterSet and ResParameterSet.

4.5.53 SlmpParameterVersionCheck-PDU

```
SlmpParameterVersionCheck-PDU::= CHOICE {
        reqParameterVersionCheck             ReqParameterVersionCheck,
        resParameterVersionCheck             ResParameterVersionCheck
}
```

NOTE Refer to the SLMP Specification for ReqParameterVersionCheck and ResParameterVersionCheck.

4.5.54 SlmpDeviceIdentificationInfoCompare-PDU

```
SlmpDeviceIdentificationInfoCompare-PDU::= CHOICE {
        reqDeviceIdentificationInfoCompare   ReqDeviceIdentificationInfoCompare,
        resDeviceIdentificationInfoCompare   ResDeviceIdentificationInfoCompare
}
```

NOTE Refer to the SLMP Specification for ReqDeviceIdentificationInfoCompare and ResDeviceIdentificationInfoCompare.

4.5.55 SlmpNodeSearch-PDU

```
SlmpNodeSearch-PDU::= CHOICE {
        reqNodeSearch                        ReqNodeSearch,
        resNodeSearch                        ResNodeSearch
}
```

NOTE Refer to the SLMP Specification for ReqNodeSearch and ResNodeSearch.

4.5.56 SlmpIPAddressSet-PDU

```
SlmpIPAddressSet-PDU::= CHOICE {
    reqIPAddressSet                    ReqIPAddressSet,
    resIPAddressSet                    ResIPAddressSet
}
```

NOTE Refer to the SLMP Specification for ReqIPAddressSet and ResIPAddressSet.

4.5.57 SlmpDeviceInfoCompare-PDU

```
SlmpDeviceInfoCompare-PDU::= CHOICE {
    reqDeviceInfoCompare               ReqDeviceInfoCompare,
    resDeviceInfoCompare               ResDeviceInfoCompare
}
```

NOTE Refer to the SLMP Specification for ReqDeviceInfoCompare and ResDeviceInfoCompare.

4.5.58 SlmpParameterGet-PDU

```
SlmpParameterGet-PDU::= CHOICE {
    reqParameterGet                    ReqParameterGet,
    resParameterGet                    ResParameterGet
}
```

NOTE Refer to the SLMP Specification for ReqParameterGet and ResParameterGet.

4.5.59 SlmpParameterSet-PDU

```
SlmpParameterSet-PDU::= CHOICE {
    reqParameterSet                    ReqParameterSet,
    resParameterSet                    ResParameterSet
}
```

NOTE Refer to the SLMP Specification for ReqParameterSet and ResParameterSet.

4.5.60 SlmpParameterSetStart-PDU

```
SlmpParameterSetStart-PDU::= CHOICE {
    reqParameterSetStart               ReqParameterSetStart,
    resParameterSetStart               Res SlmpParameterSetStart
}
```

NOTE Refer to the SLMP Specification for ReqParameterSetStart and ResParameterSetStart.

4.5.61 SlmpParameterSetEnd-PDU

```
SlmpParameterSetEnd-PDU::= CHOICE {
    reqParameterSetEnd                 ReqParameterSetEnd,
    resParameterSetEnd                 ResParameterSetEnd
}
```

NOTE Refer to the SLMP Specification for ReqParameterSetEnd and ResParameterSetEnd.

4.5.62 SlmpVerifyCheckCode-PDU

```
SlmpVerifyCheckCode-PDU::= CHOICE {
    reqVerifyCheckCode                    ReqVerifyCheckCode,
    resVerifyCheckCode                    ResVerifyCheckCode
}
```

NOTE Refer to the SLMP Specification for ReqVerifyCheckCode and ResVerifyCheckCode.

4.5.63 SlmpOutputMapFileNameGet-PDU

```
SlmpOutputMapFileNameGet-PDU::= CHOICE {
    reqOutputMapFileNameGet               ReqOutputMapFileNameGet,
    resOutputMapFileNameGet               ResOutputMapFileNameGet
}
```

NOTE Refer to the SLMP Specification for ReqOutputMapFileNameGet and ResOutputMapFileNameGet.

4.5.64 SlmpNewFile-PDU

```
SlmpNewFile-PDU::= CHOICE {
    reqNewFile                            ReqNewFile,
    resNewFile                            ResNewFile
}
```

NOTE Refer to the SLMP Specification for ReqNewFile and ResNewFile.

4.5.65 SlmpParameterSetCancel-PDU

```
SlmpParameterSetCancel-PDU::= CHOICE {
    reqParameterSetCancel                 ReqParameterSetCancel,
    resParameterSetCancel                 ResParameterSetCancel
}
```

NOTE efer to the SLMP Specification for ReqParameterSetCancel and ResParameterSetCancel.

4.5.66 SlmpOpenFile-PDU

```
SlmpOpenFile-PDU::= CHOICE {
    reqOpenFile                           ReqOpenFile,
    resOpenFile                           ResOpenFile
}
```

NOTE Refer to the SLMP Specification for ReqOpenFile and ResOpenFile.

4.5.67 SlmpCloseFile-PDU

```
SlmpCloseFile-PDU::= CHOICE {
    reqCloseFile                          ReqCloseFile,
    resCloseFile                          ResCloseFile
}
```

NOTE Refer to the SLMP Specification for ReqCloseFile and ResCloseFile.

4.5.68 SlmpReadFile-PDU

```
SlmpReadFile-PDU::= CHOICE {
    reqReadFile                     ReqReadFile,
    resReadFile                     ResReadFile
}
```

NOTE Refer to the SLMP Specification for ReqReadFile and ResReadFile.

4.5.69 SlmpWriteFile-PDU

```
SlmpWriteFile-PDU::= CHOICE {
    reqWriteFile                    ReqWriteFile,
    resWriteFile                    ResWriteFile
}
```

NOTE Refer to the SLMP Specification for ReqWriteFile and ResWriteFile.

4.5.70 SlmpStatusRead-PDU

```
SlmpStatusRead-PDU::= CHOICE {
    reqStatusRead                   ReqStatusRead,
    resStatusRead                   ResStatusRead
}
```

NOTE Refer to the SLMP Specification for ReqStatusRead and ResStatusRead.

4.5.71 SlmpCommunicationSettingGet-PDU

```
SlmpCommunicationSettingGet-PDU::= CHOICE {
    reqCommunicationSettingGet      ReqCommunicationSettingGet,
    resCommunicationSettingGet      ResCommunicationSettingGet
}
```

NOTE Refer to the SLMP Specification for ReqCommunicationSettingGet and ResCommunicationSettingGet.

4.5.72 SlmpGetDeviceInfo-PDU

```
SlmpGetDeviceInfo-PDU::= CHOICE {
    reqGetDeviceInfo                ReqGetDeviceInfo,
    resGetDeviceInfo                ResGetDeviceInfo
}
```

NOTE Refer to the SLMP Specification for ReqGetDeviceInfo and ResGetDeviceInfo.

4.5.73 SlmpGetBackupListFileName-PDU

```
SlmpGetBackupListFileName-PDU::= CHOICE {
    reqGetBackupListFileName        ReqGetBackupListFileName,
    resGetBackupListFileName        ResGetBackupListFileName
}
```

NOTE Refer to the SLMP Specification for ReqGetBackupListFileName and ResGetBackupListFileName.

4.5.74 SlmpStartBackup-PDU

```
SlmpStartBackup-PDU::= CHOICE {
        reqStartBackup                ReqStartBackup,
        resStartBackup                ResStartBackup
}
```

NOTE Refer to the SLMP Specification for ReqStartBackup and ResStartBackup.

4.5.75 SlmpEndBackup-PDU

```
SlmpEndBackup-PDU::= CHOICE {
        reqEndBackup                  ReqEndBackup,
        resEndBackup                  ResEndBackup
}
```

NOTE Refer to the SLMP Specification for ReqEndBackup and ResEndBackup.

4.5.76 SlmpCheckRestoreVersion-PDU

```
SlmpCheckRestoreVersion-PDU::= CHOICE {
        reqCheckRestoreVersion        ReqCheckRestoreVersion,
        resCheckRestoreVersion        ResCheckRestoreVersion
}
```

NOTE Refer to the SLMP Specification for ReqCheckRestoreVersion and ResCheckRestoreVersion.

4.5.77 SlmpStartRestore-PDU

```
SlmpStartRestore-PDU::= CHOICE {
        reqStartRestore               ReqStartRestore,
        resStartRestore               ResStartRestore
}
```

NOTE Refer to the SLMP Specification for ReqStartRestore and ResStartRestore.

4.5.78 SlmpEndRestore-PDU

```
SlmpEndRestore-PDU::= CHOICE {
        reqEndRestore                 ReqEndRestore,
        resEndRestore                 ResEndRestore
}
```

NOTE Refer to the SLMP Specification for ReqEndRestore and ResEndRestore.

4.5.79 SlmpStatusRead2-PDU

```
SlmpStatusRead2-PDU::= CHOICE {
        reqStatusRead2                ReqStatusRead2,
        resStatusRead2                ResStatusRead2
}
```

NOTE Refer to the SLMP Specification for ReqStatusRead2 and ResStatusRead2.

4.5.80 SlmpReqSearchNode-PDU

```
SlmpReqSearchNode-PDU::= CHOICE {
    reqReqSearchNode                    ReqReqSearchNode,
    resReqSearchNode                    ResReqSearchNode
}
```

NOTE Refer to the SLMP Specification for ReqReqSearchNode and ResReqSearchNode.

4.5.81 SlmpGetSearchNodeState-PDU

```
SlmpGetSearchNodeState-PDU::= CHOICE {
    reqGetSearchNodeState               ReqGetSearchNodeState,
    resGetSearchNodeState               ResGetSearchNodeState
}
```

NOTE Refer to the SLMP Specification for ReqGetSearchNodeState and ResGetSearchNodeState.

4.5.82 SlmpGetNodeList-PDU

```
SlmpGetNodeList-PDU::= CHOICE {
    reqGetNodeList                      ReqGetNodeList,
    resGetNodeList                      ResGetNodeList
}
```

NOTE Refer to the SLMP Specification for ReqGetNodeList and ResGetNodeList.

4.5.83 SlmpReqSetIPAddress-PDU

```
SlmpReqSetIPAddress-PDU::= CHOICE {
    reqReqSetIPAddress                  ReqReqSetIPAddress,
    resReqSetIPAddress                  ResReqSetIPAddress
}
```

NOTE Refer to the SLMP Specification for ReqReqSetIPAddress and ResReqSetIPAddress.

4.5.84 SlmpSearchPrmControlStation-PDU

```
SlmpSearchPrmControlStation-PDU::= CHOICE {
    reqSearchPrmControlStation          ReqSearchPrmControlStation,
    resSearchPrmControlStation          ResSearchPrmControlStation
}
```

NOTE Refer to the SLMP Specification for ReqSearchPrmControlStation and ResSearchPrmControlStation.

4.5.85 SlmpRequestRestore-PDU

```
SlmpRequestRestore-PDU::= CHOICE {
      reqRequestRestore                ReqRequestRestore,
      resRequestRestore                ResRequestRestore,
      errRequestRestore                ErrRequestRestore
}
```

NOTE Refer to the SLMP Specification for ReqRequestRestore, ResRequestRestore and ErrRequestRestore.

4.5.86 SlmpCheckPrmDelivery-PDU

```
SlmpCheckPrmDelivery-PDU::= CHOICE {
      reqCheckPrmDelivery              ReqCheckPrmDelivery,
      resCheckPrmDelivery              ResCheckPrmDelivery,
      errCheckPrmDelivery              ErrCheckPrmDelivery
}
```

NOTE Refer to the SLMP Specification for ReqCheckPrmDelivery, ResCheckPrmDelivery, and ErrCheckPrmDelivery.

4.5.87 SlmpRsvStationConfigTemporaryRelease-PDU

```
SlmpRsvStationConfigTemporaryRelease-PDU::= CHOICE {
      reqRsvStationConfigTemporaryRelease      ReqRsvStationConfigTemporaryRelease,
      resRsvStationConfigTemporaryRelease      ResRsvStationConfigTemporaryRelease
}
```

NOTE Refer to the SLMP Specification for ReqRsvStationConfigTemporaryRelease and ResRsvStationConfigTemporaryRelease.

4.5.88 SlmpRsvStationConfig-PDU

```
SlmpRsvStationConfig-PDU::= CHOICE {
      reqRsvStationConfig              ReqRsvStationConfig,
      resRsvStationConfig              ResRsvStationConfig
}
```

NOTE Refer to the SLMP Specification for ReqRsvStationConfig and ResRsvStationConfig.

4.5.89 SlmpGetEventNum-PDU

```
SlmpGetEventNum-PDU::= CHOICE {
      reqGetEventNum                   ReqGetEventNum,
      resGetEventNum                   ResGetEventNum
}
```

NOTE Refer to the SLMP Specification for ReqGetEventNum and ResGetEventNum.

4.5.90 SlmpGetEventHistory-PDU

```
SlmpGetEventHistory-PDU::= CHOICE {
    reqGetEventHistory          ReqGetEventHistory,
    resGetEventHistory          ResGetEventHistory
}
```

NOTE Refer to the SLMP Specification for ReqGetEventHistory and ResGetEventHistory.

4.5.91 SlmpClearEventHistory-PDU

```
SlmpClearEventHistory-PDU::= CHOICE {
    reqClearEventHistory        ReqClearEventHistory,
    resClearEventHistory        ResClearEventHistory
}
```

NOTE For ReqClearEventHistory and ResClearEventHistory, refer to the SLMP Specification.

4.5.92 SlmpClockOffsetDataSend-PDU

```
SlmpClockOffsetDataSend-PDU::= CHOICE {
    reqClockOffsetDataSend      ReqClockOffsetDataSend,
    resClockOffsetDataSend      ResClockOffsetDataSend
}
```

NOTE For ReqClockOffsetDataSend and ResClockOffsetDataSend, refer to the SLMP Specification.

4.5.93 SlmpSetWatchdogCounterInfo-PDU

```
SlmpSetWatchdogCounterInfo-PDU::= CHOICE {
    reqSetWatchdogCounterInfo   ReqSetWatchdogCounterInfo,
    resSetWatchdogCounterInfo   ResSetWatchdogCounterInfo
}
```

NOTE Refer to the SLMP Specification for ReqSetWatchdogCounterInfo and ResSetWatchdogCounterInfo.

4.5.94 SlmpWatchdogCounterOffsetConfig-PDU

```
SlmpWatchdogCounterOffsetConfig-PDU::= CHOICE {
    reqWatchdogCounterOffsetConfig   ReqWatchdogCounterOffsetConfig,
    resWatchdogCounterOffsetConfig   ResWatchdogCounterOffsetConfig
}
```

NOTE Refer to the SLMP Specification for ReqWatchdogCounterOffsetConfig and ResWatchdogCounterOffsetConfig.

4.5.95 SlmpRemoteReset-PDU

```
SlmpRemoteReset-PDU::= CHOICE {
    reqRemoteReset              ReqRemoteReset,
    resRemoteReset              ResRemoteReset
}
```

NOTE Refer to the SLMP Specification for ReqRemoteReset and ResRemoteReset.

4.5.96 SlmpGetCommunicationSet-PDU

```
SlmpGetCommunicationSet-PDU::= CHOICE {
    reqGetCommunicationSet        ReqGetCommunicationSet,
    resGetCommunicationSet        ResGetCommunicationSet
}
```

NOTE Refer to the SLMP Specification for ReqGetCommunicationSet and ResGetCommunicationSet.

4.5.97 SlmpGetStationSubIDList-PDU

```
SlmpGetStationSubIDList-PDU::= CHOICE {
    reqGetStationSubIDList        ReqGetStationSubIDList,
    resGetStationSubIDList        ResGetStationSubIDList
}
```

NOTE Refer to the SLMP Specification for ReqGetStationSubIDList and ResGetStationSubIDList.

4.5.98 SlmpGetDeviceInfo-PDU

```
SlmpGetDeviceInfo-PDU::= CHOICE {
    reqGetDeviceInfo        ReqGetDeviceInfo,
    resGetDeviceInfo        ResGetDeviceInfo
}
```

NOTE Refer to the SLMP Specification for ReqGetDeviceInfo and ResGetDeviceInfo.

4.5.99 SlmpStartBackup-PDU

```
SlmpStartBackup-PDU::= CHOICE {
    reqStartBackup        ReqStartBackup,
    resStartBackup        ResStartBackup
}
```

NOTE Refer to the SLMP Specification for ReqStartBackup and ResStartBackup.

4.5.100 SlmpEndBackup-PDU

```
SlmpEndBackup-PDU::= CHOICE {
    reqEndBackup        ReqEndBackup,
    resEndBackup        ResEndBackup
}
```

NOTE Refer to the SLMP Specification for ReqEndBackup and ResEndBackup.

4.5.101 SlmpRequestBackup-PDU

```
SlmpRequestBackup-PDU::= CHOICE {
    reqRequestBackup        ReqRequestBackup,
    resRequestBackup        ResRequestBackup
}
```

NOTE Refer to the SLMP Specification for ReqRequestBackup and ResRequestBackup.

4.5.102 SlmpGetBackupPrm-PDU

```
SlmpGetBackupPrm-PDU::= CHOICE {
    reqGetBackupPrm                     ReqGetBackupPrm,
    resGetBackupPrm                     ResGetBackupPrm
}
```

NOTE Refer to the SLMP Specification for ReqGetBackupPrm and ResGetBackupPrm.

4.5.103 SlmpCheckRestore-PDU

```
SlmpCheckRestore-PDU::= CHOICE {
    reqCheckRestore                     ReqCheckRestore,
    resCheckRestore                     ResCheckRestore
}
```

NOTE Refer to the SLMP Specification for ReqCheck Restore and ResCheck Restore.

4.5.104 SlmpStartRestore-PDU

```
SlmpStartRestore-PDU::= CHOICE {
    reqStartRestore                     ReqStartRestore,
    resStartRestore                     ResStartRestore
}
```

NOTE Refer to the SLMP Specification for ReqStartRestore and ResStartRestore.

4.5.105 SlmpEndRestore-PDU

```
SlmpEndRestore-PDU::= CHOICE {
    reqEndRestore                       ReqEndRestore,
    resEndRestore                       ResEndRestore
}
```

NOTE Refer to the SLMP Specification for ReqEndRestore and ResEndRestore.

4.5.106 SlmpSetBackupPrm-PDU

```
SlmpSetBackupPrm-PDU::= CHOICE {
    reqSetBackupPrm                     ReqSetBackupPrm,
    resSetBackupPrm                     ResSetBackupPrm
}
```

NOTE Refer to the SLMP Specification for ReqSetBackupPrm and ResSetBackupPrm.

4.5.107 SlmpLinkupSpeed-PDU

```
SlmpLinkupSpeed-PDU::= CHOICE {
    reqCommunicationSpeed               ReqCommunicationSpeed,
    resCommunicationSpeed               ResCommunicationSpeed
}
```

NOTE For ReqCommunicationSpeed and ResCommunicationSpeed, refer to the SLMP Specification.

4.5.108 SlmpNodeIndication-PDU

```
SlmpNodeIndication-PDU::= CHOICE {
    reqNodeIndication          ReqNodeIndication,
    resNodeIndication          ResNodeIndication
}
```

NOTE For ReqNodeIndication and ResNodeIndication, refer to the SLMP Specification.

4.6 Data type assignments for type T

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

```
DCS::= Unsigned32
EOS::= Unsigned16
FrameType::= Unsigned8
CyclicNo::= Unsigned8
SA::= OCTET STRING (SIZE (2))
DA::= OCTET STRING (SIZE (2))
HEC::= Unsigned32
OctetString::= OCTET STRING
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))
ApplicationData::= LOctetString
LOctetString::= LOCTET STRING
```

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 a variable length and is encoded as little endian, where the lowest, or least significant, octet is ordered as the first octet, and follows 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
0x02	Scan-PDU
0x03	Collect-PDU
0x04	Select-PDU
0x05	Launch-PDU
0x06	Token-PDU
0x07..0x09	Reserved
0x10	Persuasion-PDU
0x11	TestData-PDU
0x12	TestDataACk-PDU
0x13	Setup-PDU
0x14	SetupAck-PDU
0x15	Token-M-PDU
0x16..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
0x04	Transient2-PDU, NTNTest-PDU
0x05	Reserved
0x06	Transient1-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
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 availableFuncs

This field represents the transmission point extended mode availability.

- 0x01: the transmission extended mode is available.
- 0x00: the transmission extended mode is not available.

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
0x0000	Reserved
0x0001	Parameter delivery
0x0002..0xFFFF	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 1 466 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

<table>
<tr><th>Field name</th><th>Size (octets)</th><th colspan="2">Description</th></tr>
<tr><td>Parameter Name</td><td>8</td><td colspan="2">Specified by the user</td></tr>
<tr><td>Total Size</td><td>2</td><td colspan="2">Data size from start block to end block in octet</td></tr>
<tr><td>Sum Check Enabled</td><td>1</td><td colspan="2">0: Perform sum check
1: Not perform sum check (default)</td></tr>
<tr><td>Reserved</td><td>1</td><td colspan="2">Reserved</td></tr>
<tr><td>Sum Check Value</td><td>4</td><td colspan="2">Check sum value from start block to end block</td></tr>
<tr><td>Create Time</td><td>12</td><td colspan="2">Creation year/month/day/hour/minute/second</td></tr>
<tr><td></td><td></td><td>Size (octets)</td><td>Description</td></tr>
<tr><td></td><td></td><td>1</td><td>Last two digits of A.D.</td></tr>
<tr><td></td><td></td><td>1</td><td>First two digits of A.D.</td></tr>
<tr><td></td><td></td><td>2</td><td>Month</td></tr>
<tr><td></td><td></td><td>2</td><td>Day</td></tr>
<tr><td></td><td></td><td>2</td><td>Hour</td></tr>
<tr><td></td><td></td><td>2</td><td>Minute</td></tr>
<tr><td></td><td></td><td>2</td><td>Second</td></tr>
<tr><td>Begin Marker</td><td>4</td><td colspan="2">0x5047532d</td></tr>
<tr><td>Param Area</td><td>0..5 924</td><td colspan="2">Refer to Table 21.</td></tr>
<tr><td>End Marker</td><td>4</td><td colspan="2">0x5047452d</td></tr>
</table>

Table 21 – Details of param area

Field name	Size (octets)	Description
Parameter Availability	2	Bit 0: LB/LW Common Memory Area Bit 1: LB/LW Common Memory Additional Area Bit 8: LX/LY Common Memory Area Bit 9: 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 Bit 0 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 Bit 1 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 Bit 8 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 Bit 9 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
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 × 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 × 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 of 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.

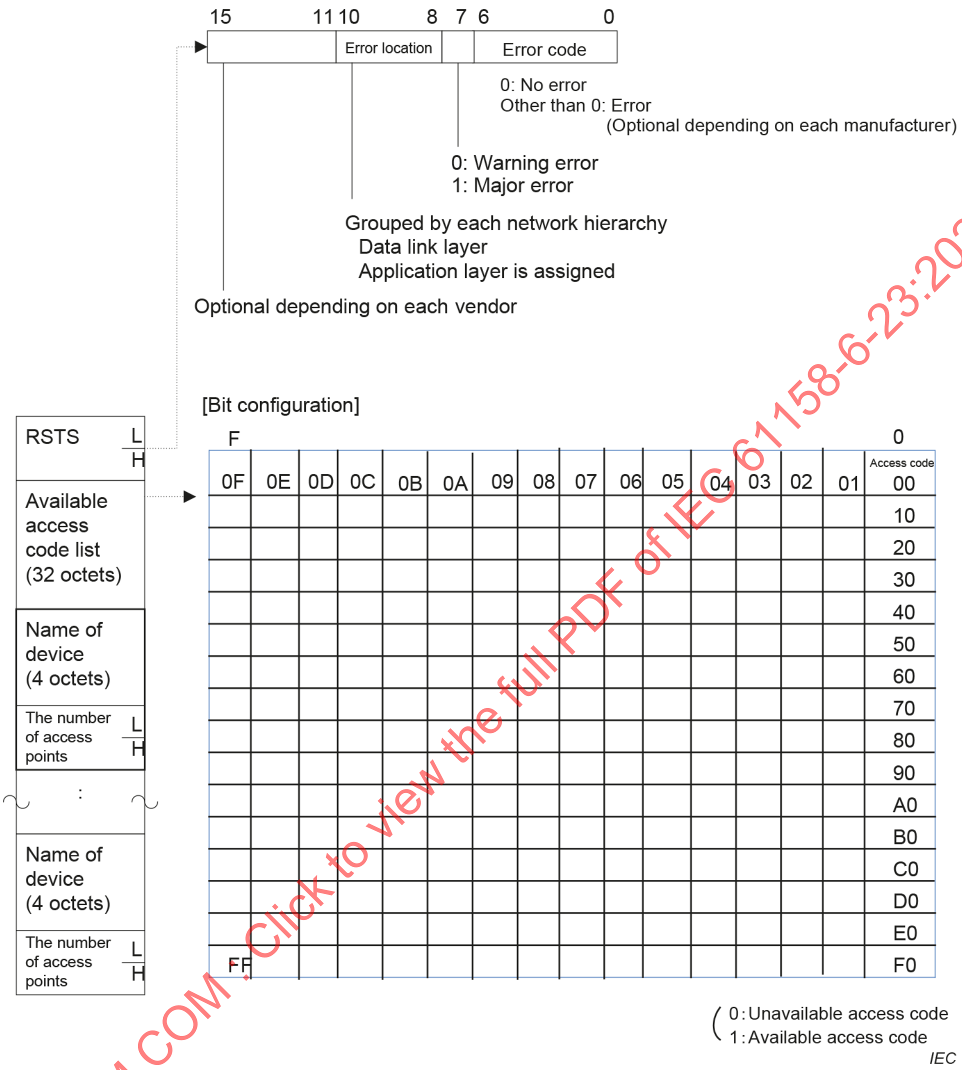

Figure 2 – Structure for memory access information retrieve response

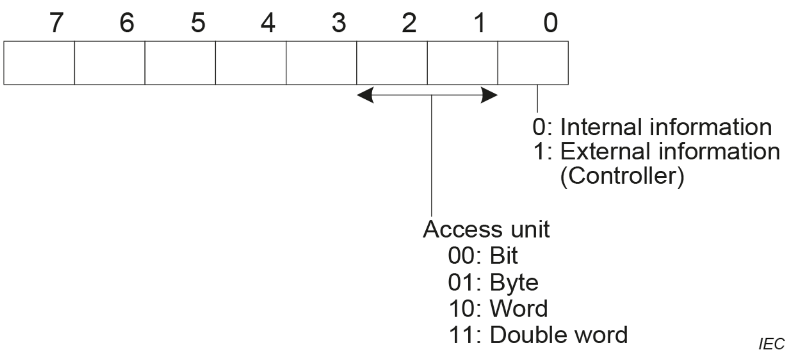

Figure 3 – Attribute definitions

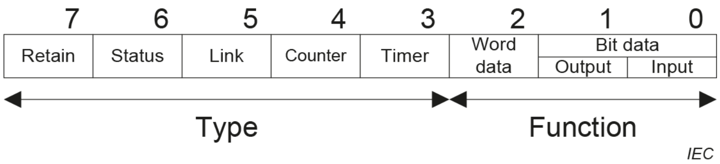

Figure 4 – Access code definitions

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	
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.

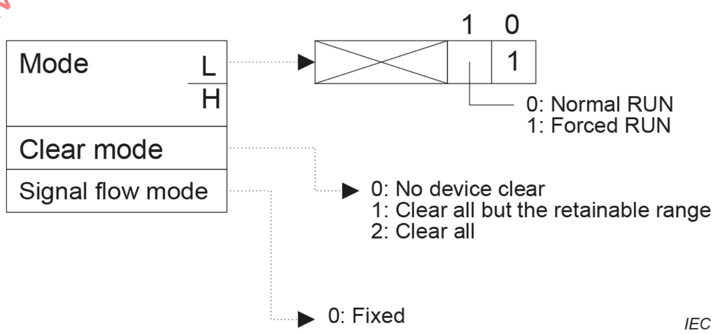

Figure 5 – Structure for RUN request

The structure when responding is shown in Figure 6.

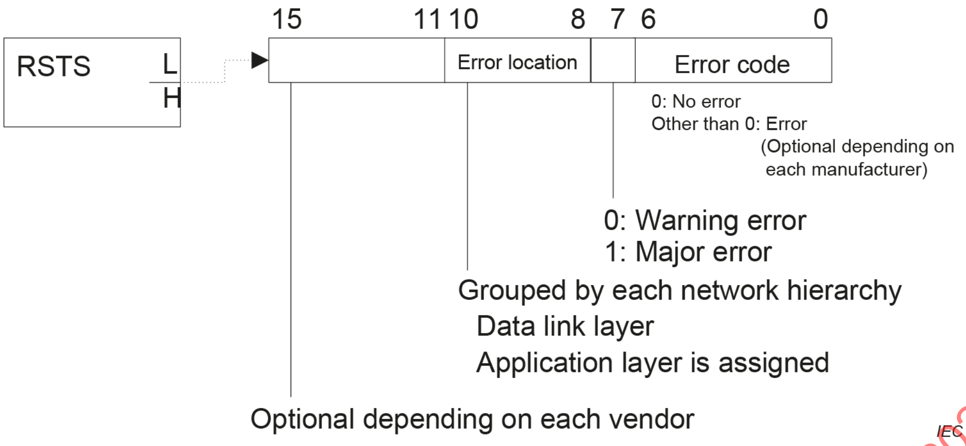

Figure 6 – Structure for RUN response

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.

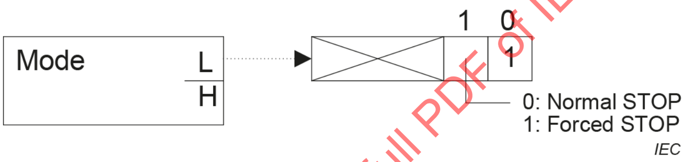

Figure 7 – Structure for STOP request

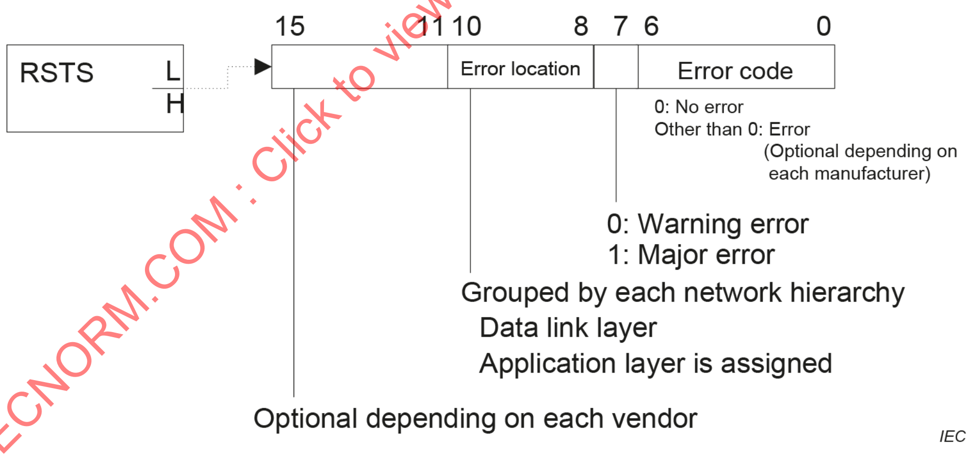

Figure 8 – Structure for STOP response

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.

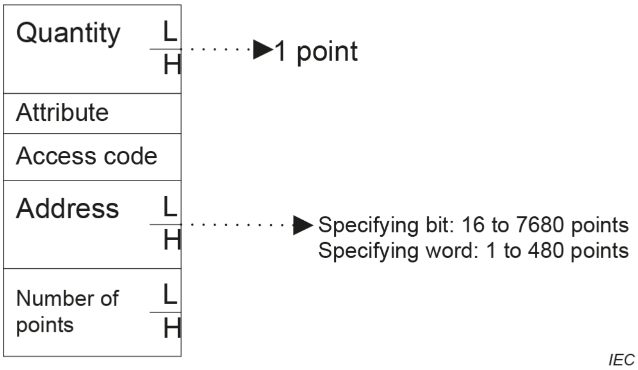

Figure 9 – Structure for batch memory read request

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.

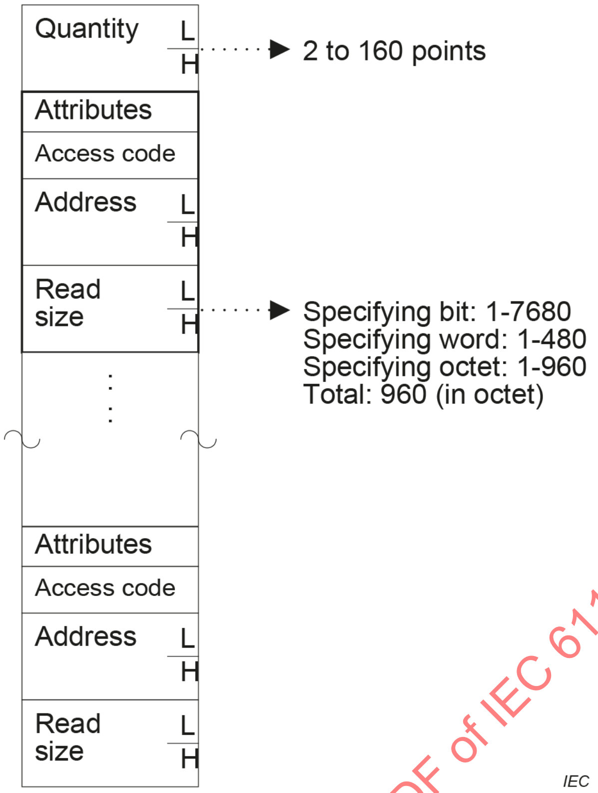

Figure 11 – Structure for random memory read request

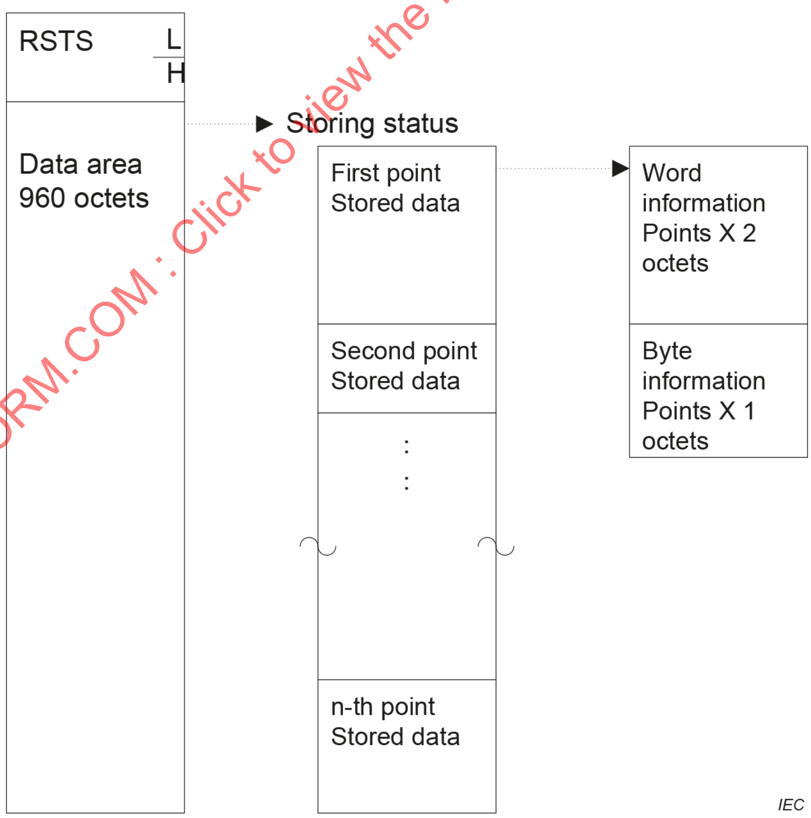

Figure 12 – Structure for random memory read response

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.

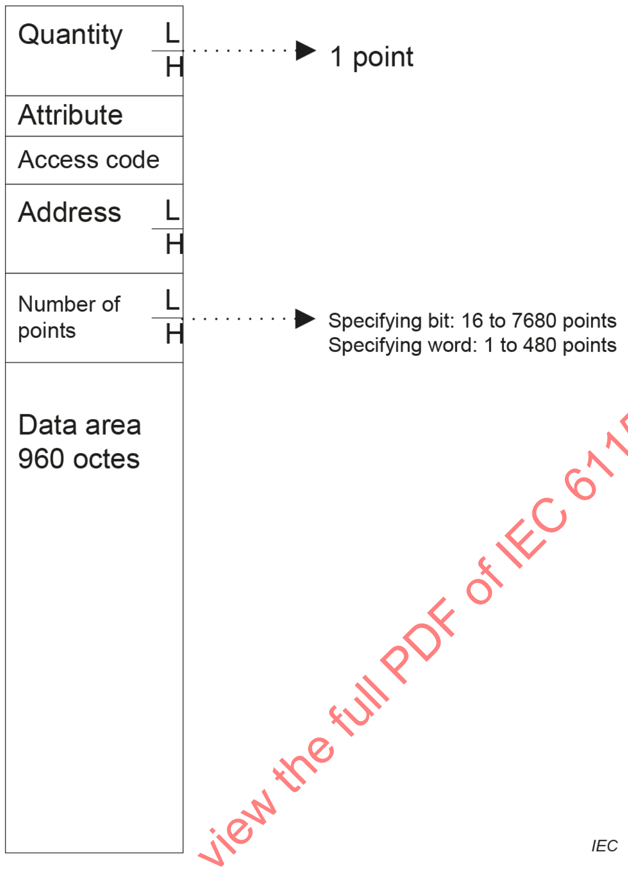

Figure 13 – Structure for batch memory write request

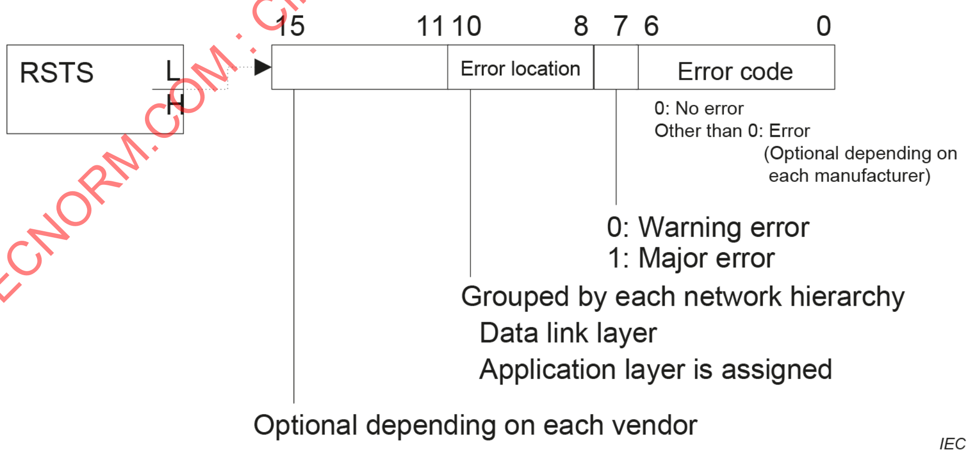

Figure 14 – Structure for batch memory write response

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

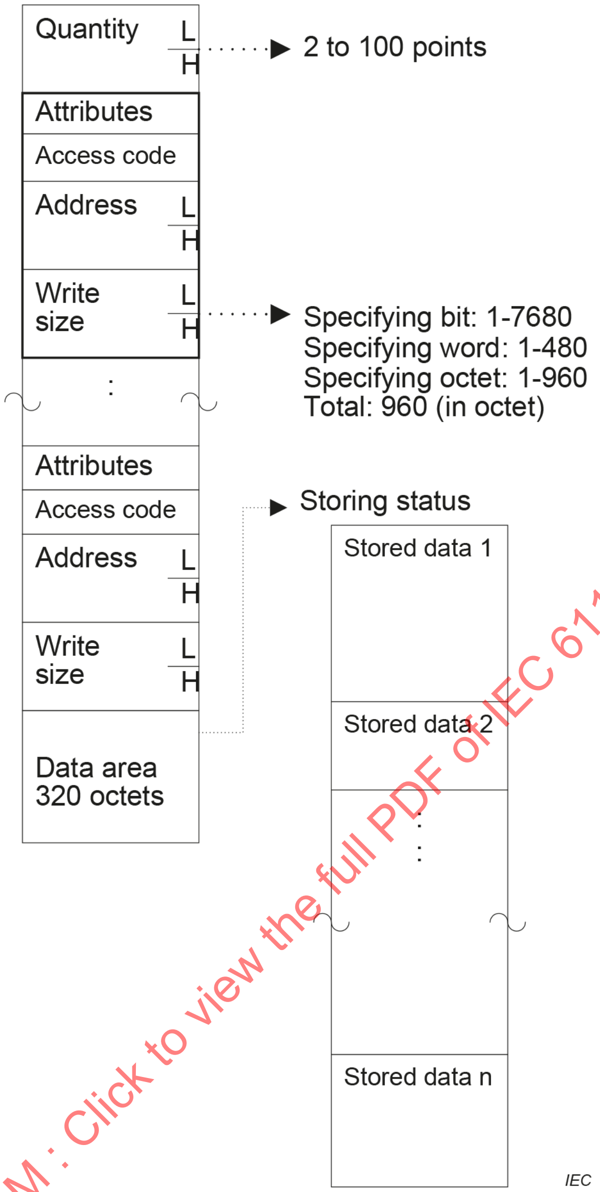

Figure 15 – Structure for random memory write request

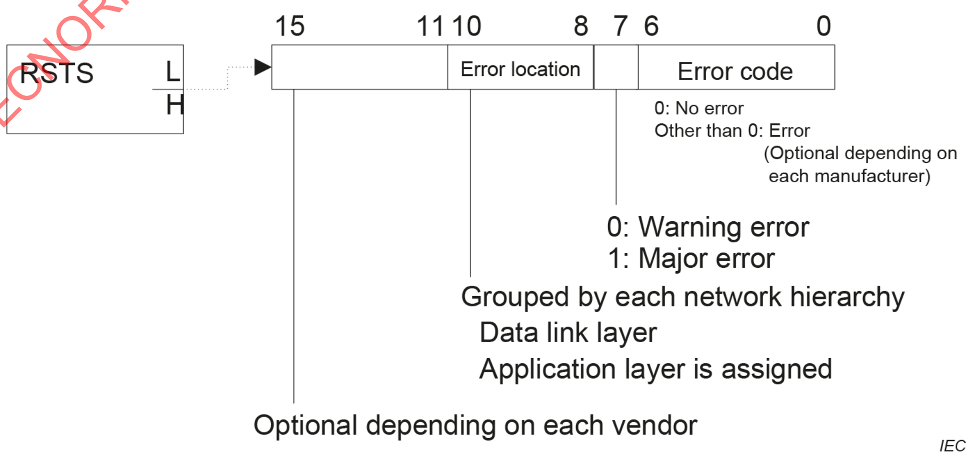

Figure 16 – Structure for random memory write response

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
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 to 0x29	0x00..0x02	Reserved for future use
	0x03	Cyclic transmission setting
	0x04..0xFF	Reserved for future use
0x82 to 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 timer-PDU	reserved1
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 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 falArHeader

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 to 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 is able to 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 transmisions permitted 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 5.3.9.3.

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
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 1 466 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 Acquisition 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.
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.
contAvailableFuncs	contains availability for the transmission extended mode • 0x00: the transmission extended mode is not available. • 0x01: the transmission extended mode is available.

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 to 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 it is possible that the start 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 it is possible that the start 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 it is possible that the start 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 it is possible that the start 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 400 µs.

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.

5.4 FALPDU type T elements encoding

5.4.1 CyclicM-PDU

5.4.1.1 cyclicMSHeader

5.4.1.1.1 frameType

This field represents the frame type. The value according to Table 51 is used.

Table 51 – frameType

Value	Description
0x00 to 0xBF	Used in CC-Link IE Controller Network and CC-Link IE Field Network.
0xC0	acyclicPriority-PDU
0xC1	acyclicDetection-PDU
0xC2	acyclicDetectionAck-PDU
0xC3	acyclicData-PDU
0xC4	cyclicMs-PDU
0xC5	cyclicSs-PDU
0xC6	cyclicM-PDU
0xC7	cyclicS-PDU
0xC8 to 0xFF	For future expansion

5.4.1.1.2 cycleNo

This field represents the cycle number of the cyclic transmission. The value according to Table 52 is used.

Table 52 – cycleNo

Bit	Value	Description
Bit 7	0	Checks the cycle number, and if the cycle number is not found within the received frame, performs discard processing of the frame
	1	Performs reception processing regardless of the cycle number
Bit 6 to 0	0 to 63	Represents a number identifying a frame to be received across cycles

5.4.1.1.3 sa

This field represents the third octet and the fourth octet of the source IP address. The value according to Table 53 is used.

Table 53 – sa

Bit	Value	Description
Bit 15 to 8	0 to 254	3rd octet of the source IP address
	255	For future expansion
Bit 7 to 0	0	For future expansion
	1 to 254	4th octet of the source IP address
	255	For future expansion

5.4.1.1.4 reserved

For future expansion. Each octet shall be 0x00.

5.4.1.2 subPayloadHeader

5.4.1.2.1 da

This field represents the third octet and the fourth octet of the destination IP address. The value according to Table 54 is used.

Table 54 – da

Bit	Value	Description
Bit 15 to 8	0 to 254	3rd octet of the destination IP address
	255	Indicates that this sub payload is transmitted to multiple stations at the time of multicast transmission. Set 255 when the cyclic data is shared with the local station.
Bit 7 to 0	0	For future expansion
	1 to 254	4th octet of the destination IP address
	255	Indicates that this sub payload is transmitted to multiple stations at the time of multicast transmission. Set 255 when the cyclic data is shared with the local station.

5.4.1.2.2 commInfo

This field represents communication-related information (controlFlag, timingErr, reserved, length). The value according to Table 55 is used.

Table 55 – commInfo

Name	Bit	Value	Description
controlFlag	Bit 15	0	Cyclic transmission disabled
		1	Cyclic transmission enabled
timingErr	Bit 14	0	Communication timing error not detected
		1	Communication timing error detected
Reserved	Bit 13 to 11	-	For future extension, 0 is used for each bit.
length	Bit 10 to 0	0 to 2047	The octet length of Application Data of the sub payload

5.4.1.2.3 txAsynInfo

This field represents the mediation type transient transmission information. The value according to Table 56 is used.

Table 56 – txAsynInfo

Bit	Value	Description
Bit 7	0	There is a transmission request
	1	There is no transmission request
Bit 6 to 0	0x00	Transient transmission is not permitted
	0x01 to 0x7F	Transient transmission is permitted

5.4.1.3 subPayloadData

5.4.1.3.1 seqNo

This field represents a sequential number. The value according to Table 57 is used.

Table 57 – seqNo

Bit	Value	Description
Bit 15	0	Represents division in-process frame
	1	Represents division final frame
Bit 14 to 0	0 to 32767	Represents a sequential number When sending by splitting into multiple frames or sub payload, serial numbers are used Also, it starts from 0 for each cycle and increments by 1 every 1 sub payload

5.4.1.3.2 diagnosisData

This field represents diagnostic information. The upper one octet represents the application error state (firmware), and the lower three octets represent the application error state (hardware). The values according to Table 58 and Table 59 are used.

Table 58 – Upper one octet of diagnosisData

Bit	Value	Description
Bit 7	0	Represents that there is no WDT error state error
	1	Represents that there is a WDT error state error
Bit 6	0	Represents that there is no N/W error check (disconnection notification of inter-slave communication) disconnection
	1	Represents that there is a N/W error check (disconnection notification of inter-slave communication) disconnection
Bit 5	-	For future expansion. 0 is used for the value.
Bit 4	0	Not invalid (Cyclic control data is valid.)
	1	Invalid (Cyclic control data is invalid.)
Bit 3	0	Not instructed (Cyclic control data is valid.)
	1	Instructed (Cyclic control data is invalid.)
Bit 2	0	Represents that there is no network part error state error
	1	Represents that there is a network part error state error
Bit 1	0	Represents that there is no application error state error
	1	Represents that there is an application error state error
Bit 0	0	Represents that the application operation is in stop state
	1	Represents that the application operation is in operating state

Table 59 – Lower three octets of diagnosisData

Bit	Value	Description
Bit 23 to 8	0 to 65535	Represents EMG GROUP
Bit 7 to 4	0 to 15	Represents GOF GROUP
Bit 3	-	For future expansion. 0 is used for the value.
Bit 2	0	Represents that ALM has not occurred
	1	Represents that ALM has occurred
Bit 1	0	Represents that there is no gate off request of the GOF power sub unit
	1	Represents that there is a gate off request of the GOF power sub unit
Bit 0	0	Represents that there is no EMG system emergency stop
	1	Represents that there is an EMG system emergency stop

5.4.1.3.3 reserved1

For future expansion. Each octet shall be 0x00.

5.4.1.3.4 memoryAddress

This field represents an address on the cyclic memory space of the destination station. When receiving the sub payload, the receiving station stores the data at the address specified in this area of the receiving RAM.

5.4.1.3.5 applicationData

This field represents control data.

5.4.2 CyclicS-PDU

5.4.2.1 cyclicMSHeader

5.4.2.1.1 frameType

Refer to 5.4.1.

5.4.2.1.2 cycleNo

Refer to 5.4.1.

5.4.2.1.3 da

Refer to 5.4.1.

5.4.2.1.4 reserved

For future expansion. Each octet shall be 0x00.

5.4.2.2 subPayloadHeader

5.4.2.2.1 sa

Refer to 5.4.1.

5.4.2.2.2 commInfo

Refer to 5.4.1.

5.4.2.2.3 txAsynInfo

Refer to 5.4.1.

5.4.2.3 subPayloadData

5.4.2.3.1 seqNo

Refer to 5.4.1.

5.4.2.3.2 diagnosisData

This field represents diagnostic information. The upper one octet represents the application error state (firmware), and the lower three octets represent the application error state (hardware). The values according to Table 60 and Table 61 are used.

Table 60 – Upper one octet of diagnosisData

Bit	Value	Description
Bit 7	0	Represents that there is no WDT error state error.
	1	Represents that there is a WDT error state error.
Bit 6	0	Represents that there is no N/W error check (disconnection notification of inter-slave communication) disconnection.
	1	Represents that there is a N/W error check (disconnection notification of inter-slave communication) disconnection.
Bit 5	-	For future expansion. Use 0 for the value.
Bit 4	0	Not invalid (Cyclic control data is valid.)
	1	Invalid (Cyclic control data is invalid.)
Bit 3	-	For future expansion. Use 0 for the value.
Bit 2	0	Represents that there is no network part error state error.
	1	Represents that there is a network part error state error.
Bit 1	0	Represents that there is no application error state error.
	1	Represents that there is an application error state error.
Bit 0	0	Represents that the application operation is in stop state.
	1	Represents that the application operation is in operating state.

Table 61 – Lower three octets of diagnosisData

Bit	Value	Description
Bit 23 to 8	0 to 65535	Represents EMG GROUP.
Bit 7 to 4	0 to 15	Represents GOF GROUP.
Bit 3	-	For future expansion. Use 0 for the value.
Bit 2	0	Represents that ALM has not occurred.
	1	Represents that ALM has occurred.
Bit 1	0	Represents that there is no gate off request of the GOF power sub unit.
	1	Represents that there is a gate off request of the GOF power sub unit.
Bit 0	0	Represents that there is no EMG system emergency stop.
	1	Represents that there is an EMG system emergency stop.

5.4.2.3.3 reserved1

For future expansion. Each octet shall be 0x00.

5.4.2.3.4 memoryAddress

Refer to 5.4.1.

5.4.2.3.5 applicationData

Refer to 5.4.1.

5.4.3 CyclicMs-PDU

5.4.3.1 cyclicMsSsHeader

5.4.3.1.1 frameType

Refer to 5.4.1.

5.4.3.1.2 cycleNo

Refer to 5.4.1.

5.4.3.1.3 sa

Refer to 5.4.1.

5.4.3.1.4 reserved

For future expansion. Each octet shall be 0x00.

5.4.3.1.5 hec

This field represents HEC. HEC is an error detection code targeting from the DestAddr of the DLPDU to the data immediately before this field (however, the VLAN tag is not included in the calculation).

The definition of DLPDU is as follows.

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

Data types used in DLPDU syntax are shown as follows.

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

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

For Cyclic Ms-PDU, each octet of this field shall be 0x00.

5.4.3.2 subPayloadHeader

5.4.3.2.1 da

Refer to 5.4.1.

5.4.3.2.2 commInfo

Refer to 5.4.1.

5.4.3.2.3 txAsynInfo

Refer to 5.4.1.

5.4.3.3 subPayloadData

5.4.3.3.1 seqNo

Refer to 5.4.1.

5.4.3.3.2 diagnosisData

Refer to 5.4.1.

5.4.3.3.3 reserved1

For future expansion. Each octet shall be 0x00.

5.4.3.3.4 memoryAddress

Refer to 5.4.1.

5.4.3.3.5 applicationData

Refer to 5.4.1.

5.4.4 CyclicSs-PDU

5.4.4.1 cyclicMsSsHeader

5.4.4.1.1 frameType

Refer to 5.4.1.

5.4.4.1.2 cycleNo

Refer to 5.4.1.

5.4.4.1.3 da

Refer to 5.4.1.

5.4.4.1.4 reserved

For future expansion. Each octet shall be 0x00.

5.4.4.1.5 hec

Refer to 5.4.3.

5.4.4.2 subPayloadHeader

5.4.4.2.1 sa

Refer to 5.4.1.

5.4.4.2.2 commInfo

Refer to 5.4.1.

5.4.4.2.3 txAsynInfo

Refer to 5.4.1.

5.4.4.3 subPayloadCheckData

5.4.4.3.1 seqNo

Refer to 5.4.1.

5.4.4.3.2 diagnosisData

Refer to 5.4.2.

5.4.4.3.3 reserved1

For future expansion. Each octet shall be 0x00.

5.4.4.3.4 memoryAddress

Refer to 5.4.1.

5.4.4.3.5 applicationData

Refer to 5.4.1.

5.4.5 AcyclicPriority-PDU

5.4.5.1 acyclicHeader

5.4.5.1.1 frameType

Refer to 5.4.1.

5.4.5.1.2 reserved

For future expansion. 0x00 is used for each octet.

5.4.5.1.3 hec

Refer to 5.4.3.

5.4.5.2 acyclicPriorityData

5.4.5.2.1 srcMAC

This field represents the MAC address of the host station.

5.4.5.2.2 mngPriority

This field represents priority information when determining the management master station. The value according to Table 62 is used.

Table 62 – mngPriority

Value	Description
0x00	Setting prohibited
0x01 to 0x7F	Represent priority
0x80 to 0xFE	Setting prohibited
0xFF	Represent priority

5.4.5.2.3 reserved1

For future expansion. 0x00 is used for each octet.

5.4.5.2.4 mngMAC

This field represents the MAC address of management master station.

5.4.5.2.5 hopCount

This field represents the number of hops from the management master station. A value between 0x0000 and 0xFFFF is used.

5.4.5.2.6 KindFlag

This field represents the type and phase of this frame. The value according to Table 63 is used.

Table 63 – KindFlag

Bit	Value	Description
Bit 7 to 3	-	For future expansion. 0 is used for each bit.
Bit 2	0	Indicates the single configuration.
	1	Indicates other than the single configuration.
Bit 1	0	Indicates the initialization.
	1	Indicates the control communication.
Bit 0	0	Indicates the request frame.
	1	Indicates the response frame.

5.4.5.2.7 reserved2

For future expansion. 0x00 is used for each octet.

5.4.5.3 dcs

This field represents DCS. The DCS is an error detection code targeting from the DestAddr of the DLPDU to the field before the DCS. For the definition of DLPDU, refer to 5.4.3.1.5.

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

5.4.6 AcyclicDetection-PDU

5.4.6.1 acyclicDetectionHeader

5.4.6.1.1 frameType

Refer to 5.4.1.

5.4.6.1.2 reserved

For future expansion. 0x00 is used for each octet.

5.4.6.2 acyclicDetectionData

5.4.6.2.1 reserved1

For future expansion. 0x00 is used for each octet.

5.4.6.2.2 reserved2

For future expansion. 0x00 is used for each octet.

5.4.6.2.3 protocolVer

This field represents the protocol version. A value between 0x00 and 0xFF is used.

5.4.6.2.4 reserved3

For future expansion. 0x00 is used for each octet.

5.4.6.2.5 mngMAC

This field represents the MAC address of management master station.

5.4.6.2.6 previousNodeMAC

This field represents the MAC address of the station that sent or relayed most recently. When receiving, it is the MAC address of another station, and at transmission (relay), it is rewritten to the MAC address of the host station.

5.4.6.2.7 previousNodePort

This field represents the transmission port number of the station that transmitted or relayed most recently. When receiving, it is the port number of another station, and at transmission (relay), it is rewritten to the port number of the host station. The value according to Table 64 is used.

Table 64 – previousNodePort

Value	Description
0x00	Do not use
0x01 to 0x18	Represents a port number from Port1 to Port24
0x19 to 0xFF	For future expansion

5.4.6.2.8 optionFlag

This field represents an option flag. The value according to Table 65 is used.

Table 65 – optionFlag

Bit	Value	Description
Bit 7 to 1	-	For future expansion. 0 is used for each bit.
Bit 0	0	Determines the necessity of DetectionAck according to the state of the receiving station
	1	Always sends DetectionAck regardless of the state of the receiving station

5.4.6.2.9 hopCount

Refer to 5.4.5.

5.4.6.2.10 IP Address

Indicates the length of the IPv4 address, IPv4 subnet mask, IPv6 address, and IPv6 subnet prefix.

5.4.6.2.11 Send Info

This field represents the source information of this frame. The value according to Table 66 is used.

Table 66 – sendInfo

Bit	Value	Description
Bit 7 to 1	-	For future expansion. 0 is used for each bit.
Bit 1	0	Indicates the initialization phase.
	1	Indicates the control communication.
Bit 0	0	Indicates the stack.
	1	Indicates the peripheral device and Plag&Play.

5.4.6.2.12 Reserved4

For future expansion. 0x00 is used for each octet.

5.4.7 AcyclicDetectionAck-PDU

5.4.7.1 acyclicDetectionHeader

Refer to 5.4.6.

5.4.7.2 acyclicDetectionAckData

5.4.7.2.1 nodeType

This field represents the host station type. Each bit is defined in Table 67.

Table 67 – nodeType

Bit	Value	Description
Bit 7	0	Station with cyclic and transient function
	1	Station with transient function
Bit 6 to 2	-	For future expansion. 0 is used for each bit.
Bit 1 to 0	0	Master station
	1	Slave station
	2	Switch
	3	Use prohibited
	4	Sub-master station
	5 to F	For future expansion

5.4.7.2.2 protocolVer

Refer to 5.4.6.

5.4.7.2.3 ipAddressFourthOctet

This field represents the IP address 4th octet value held by the station. The value according to Table 68 is used.

Table 68 – IP address 4th octet

Value	Description
0x00	No IP address 4th octet setting / invalid IP address 4th octet setting
0x01 to 0xFE	IP address 4th octet setting value
0xFF	For future expansion

5.4.7.2.4 srcMAC

This field represents the host station MAC address.

5.4.7.2.5 reserved1

For future expansion. 0x00 is used for each octet.

5.4.7.2.6 previousNodeMAC

Refer to 5.4.6.

5.4.7.2.7 previousNodePort

Refer to 5.4.6.

5.4.7.2.8 detectionRcvPort

This field represents the reception port number of the station that received the AcyclicDetection-PDU. The value according to Table 69 is used.

Table 69 – detectionRcvPort

Value	Description
0x00	Do not use
0x01 to 0x18	Represents a port number from Port1 to Port24
0x19 to 0xFF	For future expansion

5.4.7.2.9 myPort

This field represents the number of ports owned by the host station. The value according to Table 70 is used.

Table 70 – myPort

Value	Description
0x00	Do not use
0x01 to 0x18	Represents a port number from Port1 to Port24
0x19 to 0xFF	For future expansion

5.4.7.2.10 reserved2

For future expansion. 0x00 is used for each octet.

5.4.7.2.11 myPortLinkStatus

This field represents the link state of all the ports owned by the host station. Four bits are used for each port (Port1 to Port24 from the lower port). 0x0 is used for a port which does not exist. Each bit has the meaning given in Table 71.

Table 71 – Four bits of myPortLinkStatus

Bit	Value	Description
Bit 3	0	Indicates full duplex
	1	Indicates half duplex
Bit 2 to 0	0	Indicates link disconnection
	1	Indicates link up at 10 Mbps
	2	Indicates link up at 100 Mbps
	3	Indicates link up at 1 Gbps
	4	For future expansion. 0 is used for the value.
	5	For future expansion. 0 is used for the value.

5.4.7.2.12 myPortFilterStatus

This field represents the filtering state of all the ports owned by the host station. Four bits are used for each port (Port1 to Port24 from the lower port). 0x0 is used for a port which does not exist. Each bit has the meaning given in Table 72.

Table 72 – Four bits of myPortFilterStatus

Bit	Value	Description
Bit 3	-	For future expansion. 0 is used for the value.
Bit 2	0	Indicates a non-loop port
	1	Indicates a loop port
Bit 1 to 0	0	Indicates no filter
	1	Indicates prohibition of broad/multicast frame relay
	2	Indicates prohibition of relay other than the CC-Link IE field network frame
	3	Indicates prohibition of filter setting

5.4.7.2.13 currentManager

This field represents the MAC address of the current management master station.

5.4.7.2.14 reserved3

For future expansion. 0x00 is used for each octet.

5.4.7.2.15 ipAdd

This field indicates an IP address.

5.4.7.2.16 performance

This field represents the performance of the host station. Each bit has the meaning given in Table 73.

Table 73 – performance

Bit	Value	Description
Bit 7 to 4	-	For future extension, 0 is used for the value
Bit 3	0	Indicates that there is no IP address duplication.
	1	Indicates that there is IP address duplication.
Bit 2	0	Indicates that there is no time synchronization error.
	1	Indicates that there is a time synchronization error.
Bit 1	-	For future extension, 0 is used for the value
Bit 0	0	Indicates that processing is impossible with effective throughput equivalent to the link up speed
	1	Indicates that processing is possible with effective throughput equivalent to the link up speed

5.4.7.2.17 reserved4

For future expansion. 0x00 is used for each octet.

5.4.7.2.18 gmPriority

This field represents the priority information of the host station. Each bit has the meaning given in Table 74. For stations other than the grandmaster, all octets shall be 0xFF.

Table 74 – gmPriority

Bit	Value	Description
Bit 47 to 40	0 to 15	Do not use
	16 to 255	Represents Priority1 and the lower the value, the higher the priority
Bit 39 to 8	-	Represents the clock quality (the same value as grandmasterClockQuality that is included in the time synchronization Announce is used)
Bit 7 to 0	0 to 15	Do not use
	16 to 255	Represents Priority2 and the user defines the value

5.4.7.2.19 syncType

This field represents the time synchronization type of the host station. Each bit has the meaning given in Table 75.

Table 75 – syncType

Bit	Value	Description
Bit 7 to 2	-	For future extension, 0 is used for the value
Bit 1	0	Indicates that IEEE Std 802.1AS is not supported.
	1	Indicates that IEEE Std 802.1AS is supported.
Bit 0	0	Indicates that IEEE Std 1588 is not supported.
	1	Indicates that IEEE Std 1588 is supported.

5.4.7.2.20 Reserved5

For future expansion. 0x00 is used for each octet.

5.4.7.2.21 (P)delay Res Time

This field represents the time from when (P)delay_Req is received to when PDelay_Resp_Follow_Up response is sent between nearby stations.

5.4.7.2.22 Delay Set Time

This field represents the time from when propagation delay time is calculated and PortRole is set to MasterPort.

5.4.7.2.23 Announce Relay Time

This field represents the time from when Announce is received from the nearby station and PortRole is set to when MasterPort is relayed to Announce.

5.4.7.2.24 Reserved6

For future expansion. 0x00 is used for each octet.

5.4.7.2.25 deviceVer

This field represents the device version. The value is arbitrary.

5.4.7.2.26 vendorCode

This field represents the vendor code. The value managed by CLPA is used.

5.4.7.2.27 modelCode

This field represents the model name code. The value is arbitrary.

5.4.7.2.28 expansionmodelCode

This field represents the expansion model code. The value is arbitrary. Set 0x0000 when the extended model code is not used.

5.4.7.2.29 deviceType

This field represents the model type. The value managed by CLPA is used.

5.4.7.2.30 memoryAddress

Refer to 5.4.2.

5.4.7.2.31 cyclicSize

This field represents the number of cyclic points. Each bit has the meaning given in Table 76.

Table 76 – cyclicSize

Bit	Value	Description
Bit 96 to 111	0x0000 to 0xFFFF	Transmission from the slave to the master (The bit device size is stored in byte length.)
Bit 80 to 95	0x0000 to 0xFFFF	Transmission from the slave to the master (The word device size is stored in byte length.)
Bit 64 to 79	0x0000 to 0xFFFF	Transmission from the slave to the master (The safety device size is stored in byte length.)
Bit 48 to 63	0x0000 to 0xFFFF	Transmission from the master to the slave (The bit device size is stored in byte length.)
Bit 32 to 47	0x0000 to 0xFFFF	Transmission from the master to the slave (The word device size is stored in byte length.)
Bit 16 to 31	0x0000 to 0xFFFF	Transmission from the master to the slave (The safety device size is stored in byte length.)
Bit 0 to 15	0x0000 to 0xFFFF	Number of status notification device points (The word device size is stored in byte length.)

5.4.7.2.32 Reserved7

For future expansion. 0x00 is used for each octet.

5.4.7.2.33 function

This field represents the function type. Each bit has the meaning given in Table 77.

Table 77 – function

Bit	Value	Description
Bit 7 to 5	0	For future expansion
Bit 4	0	Safety communication function not used
	1	Safety communication function used
Bit 3	0	Backup/restoration function not installed
	1	Backup/restoration function installed
Bit 2	0	Watchdog counter not installed
	1	Watchdog counter installed
Bit 1	0	Indicates rejection of overwriting of parameter information such as network settings
	1	Indicates overwriting permission of parameter information such as network settings (always 1 for a slave station)
Bit 0	0	Indicates that it is not a local station
	1	Indicates that it is a local station

5.4.7.2.34 optionInfo

This field represents presence or absence of option information. Each bit has the meaning given in Table 78.

Table 78 – optionInfo

Bit	Value	Description
Bit 7 to 2	0	For future expansion
Bit 1	0	Indicates that there is no option information
	1	Indicates that there is option information
Bit 0	0	Indicates that there is no controller information part
	1	Indicates that there is a controller information part

5.4.7.2.35 stationMode

This field is used to identify COMM_IF of CSP+. A value between 0x0000 and 0xFFFF is used.

5.4.7.2.36 blank

This field represents a blank area. 0x00 is used for each octet.

5.4.8 AcyclicTestData-PDU

5.4.8.1 acyclicTestDataHeader

5.4.8.1.1 frameType

0x11 is used.

5.4.8.1.2 dataType

0x01 is used.

5.4.8.1.3 persPriority

This field represents the priority when a transmission path manager is selected. The value is from 0x0 to 0xFFFFFF.

When the value is 0x0, this field represents that the transmission path manager is not selected. Each bit has the following meaning.

Bit 23 to 22: Reserved. b10 is used.

Bit 21: For future expansion. 0 is used.

Bit 20: Indicates the status. 1: Under token control, 0: Initializing.

Bit 19: Indicates the master specification. 1: Supported by multimaster function, 0: Not supported.

Bit 18 to 15: For future expansion. 0 is used for each bit.

Bit 14 to 0: The number of cyclic data points is used. For a master station, the value of (RX/16+RY/16+RWw+RWr)/8 is used.

For a slave station, 0 is used for each bit.

5.4.8.1.4 nodeType

This field represents the node type. 0x30 is used.

5.4.8.1.5 srcNodeNumber

This field represents the own node number.

5.4.8.1.6 protocolVerType

Each bit has the following meaning.

Bit 7 to 4: Protocol version. The value according to Table 79 is used.

Bit 0 to 3: Protocol type. The value according to Table 80 is used.

Table 79 – Protocol version

Value	Description
0x0	Supported by the CC-Link IE Field or CC-Link IE controller network single master function (Not supported by the multimaster function)
0x1	Supported by the CC-Link IE Field or CC-Link IE controller network multimaster function
0x2	Supported by the CC-Link IE TSN function
0x3	Supported by the CC-Link IE TSN function and CC-Link IE Field network single master function
0x4 to 0xF	For future expansion
NOTE Bit 3 to 0: The value according to the protocol type is used. 0x3 is used.	

Table 80 – Protocol type

Value	Description
0x0	CC-Link IE Controller Network
0x1	CC-Link IE Field Network
0x2	For future expansion
0x3	CC-Link IE TSN network
0x4 to 0xF	For future expansion

5.4.8.1.7 reserved

For future expansion. 0x00 is used for each octet.

5.4.8.1.8 hec

Refer to 5.4.3.

5.4.8.2 acyclicTestDataData

5.4.8.2.1 macAddr

This field represents the MAC address of the management master station.

5.4.8.2.2 srcPort

This field represents the source port number.

5.4.8.2.3 reserved1

For future expansion. 0x00 is used for each octet.

5.4.8.2.4 sendInf

This field represents the source information. The value according to Table 81 is used.

Table 81 – Source information

Bit	Value	Description
Bit 3 to 7	0	For future expansion
Bit 2	0	The IP address system is IPv4.
	1	The IP address system is IPv6.
Bit 1	0	The phase is the initialization phase.
	1	The phase is in the control communication.
Bit 0	0	The TestData source is the stack.
	1	The TestData source is the engineering tool.

5.4.8.2.5 reserved2

For future expansion. 0x00 is used for each octet.

5.4.8.2.6 hopCount

Refer to 5.4.5.

5.4.8.2.7 ipv4Address

Refer to 5.4.6.

5.4.8.2.8 ipv4Subnet

Refer to 5.4.6.

5.4.8.2.9 reserved3

For future expansion. 0x00 is used for each octet.

5.4.8.2.10 ipv6Address

Refer to 5.4.6.

5.4.8.2.11 hopCount

Refer to 5.4.5.

5.4.8.2.12 ipv6Subnet

Refer to 5.4.6.

5.4.8.2.13 dcs

Refer to 5.4.5.

5.4.9 AcyclicTestDataAck-PDU

5.4.9.1 acyclicTestDataAckHeader

5.4.9.1.1 frameType

0x12 is used.

5.4.9.1.2 dataType

0x01 is used.

5.4.9.1.3 persPriority

Refer to 5.4.8.

5.4.9.1.4 nodeType

Refer to 5.4.8.

5.4.9.1.5 srcNodeNumber

Refer to 5.4.8.

5.4.9.1.6 protocolVerType

Refer to 5.4.8.

5.4.9.1.7 reserved

For future expansion. 0x00 is used for each octet.

5.4.9.1.8 hec

Refer to 5.4.8.

5.4.9.2 acyclicTestDataAckData

5.4.9.2.1 macAddr

This field represents the source node MAC address of acyclicTestData-PDU.

5.4.9.2.2 srcPort

This field represents the source port number of acyclicTestData-PDU which is included in acyclicTestData-PDU as srcPort.

5.4.9.2.3 rcvPort

This field represents the port number of the own node that received acyclicTestData-PDU.

5.4.9.2.4 reserved1

For future expansion. 0x00 is used for each octet.

5.4.9.2.5 myPorts

This field represents the number of physical communication ports owned by the own node.

5.4.9.2.6 tokenkeepTime

This field represents the maximum value of the time in which the node keeps the token after the token patrol is started.

The time to keep the token is calculated by subtracting the send time of f-transientData-PDU from the time after token-PDU to host station is received until token-PDU to the next node is sent.

The node notifies the transmission path manager of the time to keep the token using this field.

Each bit has the following meaning.

Bit 15: For future expansion. 0 is used.

Bit 14 to 0: Indicates the time setting. The unit is 1 μs, and the value is from 1 to 32 767.

NOTE The start and end points of the time to keep the token are the same as that of token-PDU. When the start point is set to the receive start point of token-PDU, the end point is set the send start point of token-PDU. When the start point is set to the receive complete point of token-PDU, the end point is set the send complete point of token-PDU.

5.4.9.2.7 reserved2

For future expansion. 0x00 is used for each octet.

5.4.9.2.8 myConnectStatus

This field represents the port status. Each four bit is used to show the status of 24 ports at maximum.

Bit 3 to 0 of the 1st octet show the port 1 status, Bit 7 to 4 show the port 2 status. Bit 3 to 0 of the 2nd octet show the port 3 status, Bit 7 to 4 show the port 4 status.

The same shall apply hereafter. The 1st to 12th octets are used to show the status of the 24 ports.

Each bit of an octet has the following meaning.

Bit 7 to 6: For future expansion. 0 is used for each bit.

Bit 5 to 4: The value according to Table 82 is used.

Bit 3 to 2: For future expansion. 0 is used for each bit.

Bit 1 to 0: The value according to Table 82 is used.

Table 82 – Link status

Value	Description
00b	Link disconnection
01b	Link-up
10b	For future expansion
11b	For future expansion

5.4.9.2.9 dcs

Refer to 5.4.5.

5.4.10 AcyclicData-PDU

5.4.10.1 acyclicDataHeader

5.4.10.1.1 frameType

Refer to 5.4.1.

5.4.10.1.2 reserved

For future expansion. 0x00 is used for each octet.

5.4.10.1.3 da

This field represents the IP address 4th octet of the destination or source station.

5.4.10.1.4 reserved1

For future expansion. 0x00 is used for each octet.

5.4.10.2 data

This field represents arbitrary data to be transmitted in transient transmission.

5.4.11 Ptp-PDU

5.4.11.1 Overview

This subclause describes only the fields that use specific values in the CC-Link IE TSN network. The method of using a field that is not particularly mentioned shall be in conformity with each standard.

5.4.11.2 transportpecific

0x1 is used. If PTP MesSAge with a value other than 0x1 is received, the message is ignored.

5.4.11.3 messageType

Other six messageTypes except ptpSignalling are used.

5.4.11.4 versionPtp

0x2 is used.

5.4.11.5 domainNo

0x00 is used.

5.4.11.6 sourcePortIdentity

The ClockIdentity field of the upper 8 octets represents the clock ID (MAC address). The MAC address of the send station itself is used.

The portNumber field of the lower 2 octets represents the port number. 1 to N values are used to indicate the station having N port. For example, value 1 indicates a 1 port station.

5.4.11.7 sequenceId

Other than ptpPdelayResp and ptpPdelayRespFollpwUp, this is a sequence number managed for each port, incrementing by 1 for each transmission. 0 is used for the initial value. The maximum value is 65 535, and when it exceeds the maximum value, it resumes from 0.

ptpPdelayResp and ptpPdelayRespFollpwUp use the sequence number of the received ptpPdelayReq.

5.4.11.8 control

0x05 is used.

5.4.11.9 LogMessageInterval

A value between –127 and 127 is used as a signed value and it is specified with 2^ (logMessageInterval) [sec].

5.4.11.10 tlvType

0x03 is used.

5.4.11.11 lengthField

28 is used.

5.4.11.12 organizationSubType

1 is used.

5.4.11.13 gmTimeBaseIndicator

If there is a change in frequency or phase, 0x0001 is used.

5.4.11.14 lastGmPhaseChange

It is calculated using [time of the previous grandmaster] – [time of the current grandmaster]. It is specified as a signed value in units of 2^ (−16) nanoseconds. For example, in the case of 2,5 nanoseconds, 0x0000_0000_0000_0000_0002 8000 is used.

5.4.11.15 scaledLastGmFreqChange

The value before/after the frequency is set.

5.4.11.16 currentUtcOffset

The offset value of UTC and TAI is set.

5.4.11.17 gmPriority1

This value represents the Priority1 value of the grandmaster recognized by the sending station. It is used by BMCA. The smaller the value, the higher the priority. 0 is prohibited to be used.

5.4.11.18 gmPriority2

This value represents the Priority2 value of the grandmaster recognized by the sending station. Each station user has the arbitrarily defined priority. 0 is prohibited to be used.

5.4.12 IpData-PDU

5.4.12.1 ipHeader

This field represents the IP header. For details, refer to IETF RFC 791.

5.4.12.2 udpHeader

This field represents the UDP header. For details, refer to IETF RFC 768.

5.4.12.3 Slmp-PDU

In the CC-Link IE TSN network, the SLMP frame of the station number extended MT type is used. For details of each PDU, refer to the SLMP Specification.

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.

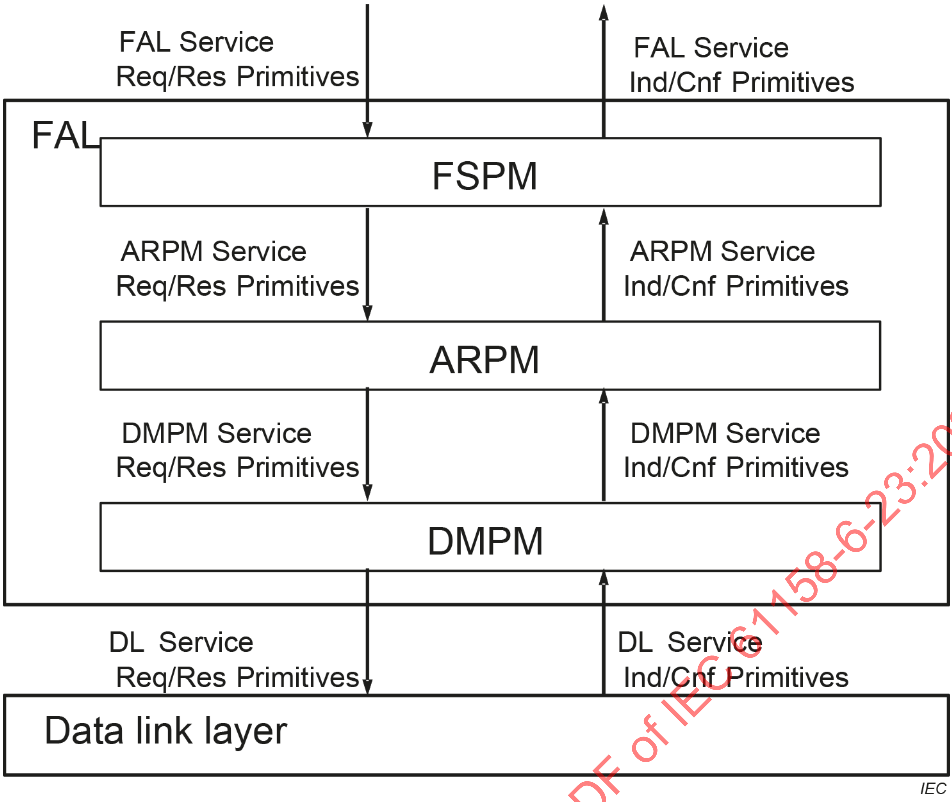

Figure 17 – Relationships between protocol machines

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.

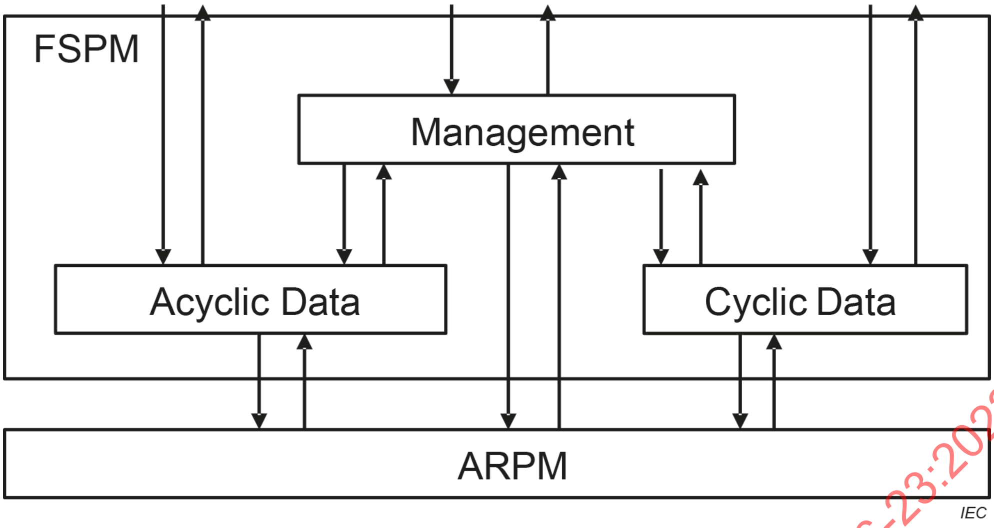

Figure 18 – Structure of FSPM C

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 83.

Table 83 – 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 84.

Table 84 – 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

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

#	Current state	Event/condition => action	Next state
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 Management

Details of Management state machine are shown in Table 85.

Table 85 – 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.

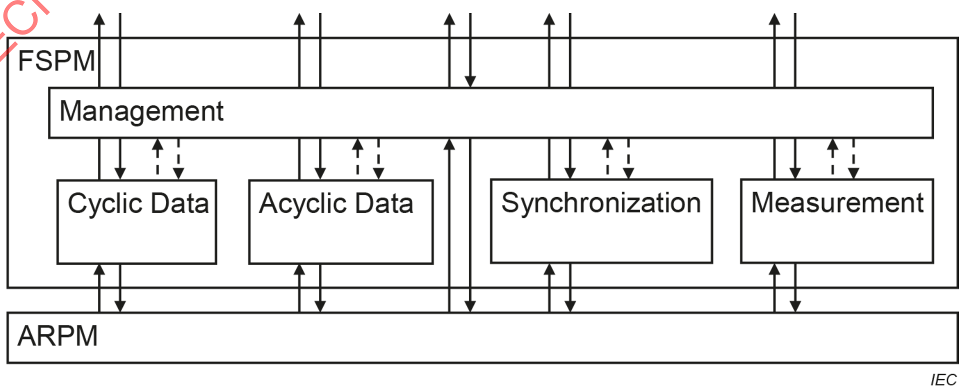

Figure 19 – Structure of FSPM F

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 86.

Table 86 – 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
6	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 87

Table 87 – 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

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

#	Current state	Event /condition => action	Next state
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
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

#	Current state	Event /condition => action	Next state
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 Management

Details of Management state machine are shown in Table 88.

Table 88 – 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 89.

Table 89 – 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 90.

Table 90 – 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

7.4 FSPM type T

7.4.1 Overview

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

FSPM consists of five protocol machines: Cyclic Data, Acyclic Data, Management, TimeSync Data, and SLMP Data.

Relationships between protocol machines are shown in Figure 20. A solid line indicates issuance of a service, and a broken line indicates linkage between protocol machines using variables.

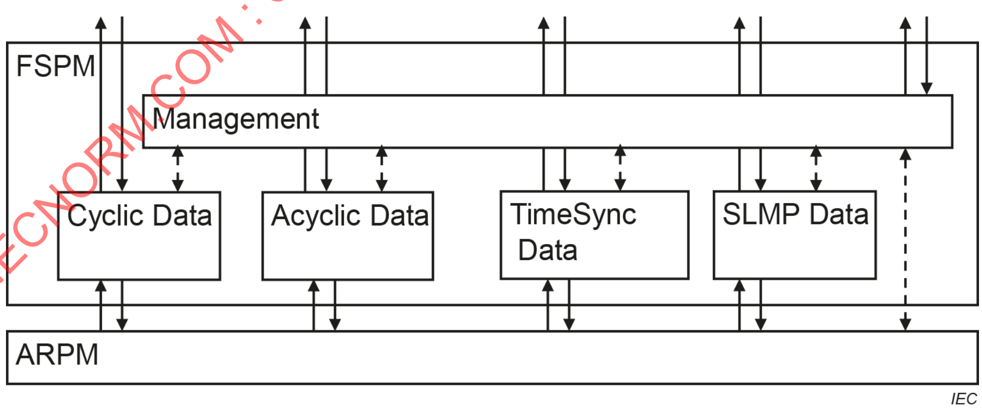

Figure 20 – Structure of FSPM T

Service primitives provided by FSPM are shown in Table 91. For details of each service, refer to the Application Layer Service.

Table 91 – Primitives provided by FSPM

Primitive Name	Publisher
Read.req	FAL user
Read.cnf	FSPM
Write.req	FAL user
Write.cnf	FSPM
Get Attribute.req	FAL user
Get Attribute.cnf	FSPM
Set Attribute.req	FAL user
Set Attribute. cnf	FSPM
Priority.req	FAL user
Priority.ind	FSPM
Detection.req	FAL user
Detection.ind	FSPM
DetectionAck.req	FAL user
DetectionAck.ind	FSPM
Acyclic Data NRSV.req	FAL user
Acyclic Data NRSV.ind	FSPM
Acyclic Data NRSV.rsp	FAL user
Acyclic Data NRSV.cnf	FSPM
Acyclic Data RSV.req	FAL user
Acyclic Data RSV.ind	FSPM
Acyclic Data RSV.rsp	FAL user
Acyclic Data RSV.cnf	FSPM
TimeSyncMng.req	FAL user
TimeSyncMng.ind	FSPM
TimeSyncMng.cnf	FSPM
SLMP Data.req	FAL user
SLMP Data.ind	FSPM
SLMP Data.rsp	FAL user
SLMP Data.cnf	FSPM

7.4.2 FSPM State Machine

7.4.2.1 Cyclic Data

The state of Cyclic Data state machine is shown in Table 92.

Table 92 – Cyclic Data state

Name	Description
ACTIVE	Operating state

Details of Cyclic Data state machine are shown in Table 93.

Table 93 – Cyclic Data state table

#	Current state	Event /Condition => Action	Next state
1	ACTIVE	Read.req(Address, Size) => Read.cnf(Data)	ACTIVE
2	ACTIVE	Write.req(Address, Size, Data) => C_Update.req(fType, Address, Size, Data) Write.cnf()	ACTIVE
3	ACTIVE	C_Update.ind(fType, Address, Size, Data) => MemoryUpdate(Address, Size, Data)	ACTIVE

The function used for Cyclic Data is shown in Table 94.

Table 94 – Function used for Cyclic Data

Name	Contents
MemoryUpdate(Address, Size, Data)	Updates the cyclic memory specified by the argument

7.4.2.2 Acyclic Data

The state of the Acyclic Data state machine is shown in Table 95.

Table 95 – Acyclic Data state

Name	Description
ACTIVE	Operating state

Details of Acyclic Data state machine are shown in Table 96.

Table 96 – Acyclic Data state table

#	Current state	Event /Condition => Action	Next state
1	ACTIVE	Priority.req() => AC_Update.req(fType)	ACTIVE
2	ACTIVE	AC_Update.ind(fType, DA, Data) / fType == 0xC0 => Priority.ind()	ACTIVE
3	ACTIVE	Detection.req() => AC_Update.req(fType)	ACTIVE
4	ACTIVE	AC_Update.ind(fType, DA, Data) / fType == 0xC1 => Detection.ind()	ACTIVE
5	ACTIVE	DetectionAck.req() => AC_Update.req(fType, DA, Data)	ACTIVE

#	Current state	Event /Condition => Action	Next state
6	ACTIVE	AC_Update.ind(fType, DA, Data) / fType == 0xC2 => DetectionAck.ind()	ACTIVE
7	ACTIVE	Acyclic Data NRSV.req(DA, Data) => AC_Update.req(fType, DA, Data)	ACTIVE
8	ACTIVE	AC_Update.ind(fType DA, Data) / fType == 0xC3 && Request Type == Client Request && ACType == NRSV => Acyclic Data NRSV.ind(Data)	ACTIVE
9	ACTIVE	AC_Update.ind(fType, DA, Data) / fType == 0xC3 && Request Type == Server Response && ACType == NRSV => Acyclic Data NRSV.cnf(Data)	ACTIVE
10	ACTIVE	Acyclic Data NRSV.rsp(fType, DA, Data) => AC_Update.req(fType, DA, Data)	ACTIVE
11	ACTIVE	Acyclic Data RSV.req(DA, Data) => AC_Update.req(fType, DA, Data)	ACTIVE
12	ACTIVE	AC_Update.ind(fType DA, Data) / fType == 0xC3 && Request Type == Client Request && ACType == RSV => Acyclic Data RSV.ind(Data)	ACTIVE
13	ACTIVE	AC_Update.ind(fType, DA, Data) / fType == 0xC3 && Request Type == Server Response && ACType == RSV => Acyclic Data RSV.cnf(Data)	ACTIVE
14	ACTIVE	Acyclic Data RSV.rsp(fType, DA, Data) => AC_Update.req(fType, DA, Data)	ACTIVE

7.4.2.3 Management

The state of the Management state machine is shown in Table 97.

Table 97 – Management state

Name	Description
ACTIVE	Operating state

Details of Management state machine are shown in Table 98.

Table 98 – Management state table

#	Current state	Event /Condition => Action	Next state
1	ACTIVE	Get Attribute.req(data) => Get Attribute.cnf(data)	ACTIVE
2	ACTIVE	Set Attribute.req(data) => Set Attribute.cnf(data)	ACTIVE

7.4.2.4 TimeSync Data

The state of the TimeSync Data state machine is shown in Table 99.

Table 99 – TimeSync Data state

Name	Description
ACTIVE	Operating state

Details of TimeSync Data state machine are shown in Table 100.

Table 100 – TimeSync Data state table

#	Current state	Event /Condition => Action	Next state
1	ACTIVE	TimeSyncMng.req(data) => TimeSync.req(data)	ACTIVE
2	ACTIVE	TimeSync.ind(TimeSyncStatus) => TimeSyncMng.ind(TimeSyncStatus)	ACTIVE
3	ACTIVE	TimeSync.cnf(data) => TimeSyncMng.cnf(data)	ACTIVE

7.4.2.5 SLMP Data

The state of the SLMP Data state machine is shown in Table 101.

Table 101 – SLMP Data state

Name	Description
ACTIVE	Operating state

Details of SLMP Data state machine are shown in Table 102.

Table 102 – SLMP Data state table

#	Current state	Event /Condition => Action	Next state
1	ACTIVE	SLMP Data.req(SlmpData) => SLMPSend.req(SlmpData)	ACTIVE
2	ACTIVE	SLMPSend.ind(SlmpData) / slmpMTHeader.fType == 0x6800 => SLMPData.ind(SlmpData)	ACTIVE
3	ACTIVE	SLMPSend.ind(SlmpData) / slmpMTHeader.fType == 0xE800 => SLMPData.cnf(SlmpData)	ACTIVE
4	ACTIVE	SLMPData.rsp(SlmpData) => SLMPSend.req(SlmpData)	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 21. The continuous line represents a service issue, and the dashed line represents a linkage between protocol machines using parameters and others.

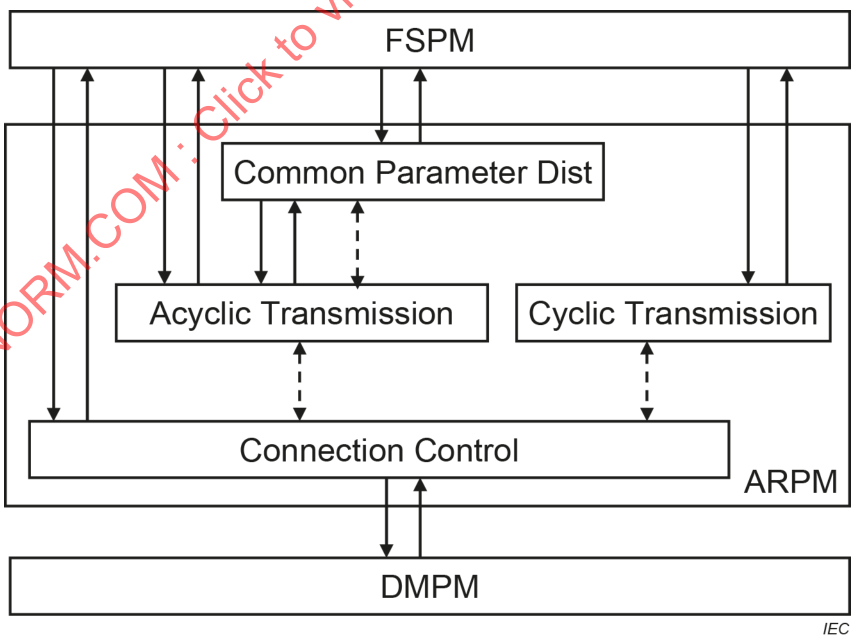

Figure 21 – Structure of ARPM C

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 103

Table 103 – 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 && 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

#	Current state	Event /condition => action	Next state
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 104.

Table 104 – 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 105.

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

Table 105 – 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)

#	Current state	Event /condition => action	Next state
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
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

#	Current state	Event /condition => action	Next state
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 => OutCM1Sent = TRUE	SENDER
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

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

#	Current state	Event /condition => action	Next state
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 ReceivedOutCM2 = RetrieveOutCM2(CyclicDataOut2-PDU); CT Update.ind(OutCM2, Offset Address, Size, ReceivedOutCM2) When not in the order of the sequential number, discarded	Any (no change)
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 106.

Table 106 – Cyclic transmission functions

Name	Description
length	Request the size given by argument.
Update	Update the data of BitCM, WordCM, OutCM1, OutCM2, InCM1, and InCM2 thatCyclic Transmission holds to send.
CreateCyclicDataB-PDU	Generate CyclicDataB-PDU. If argument data exceeds 1 468 octets, CyclicDataB-PDU is generated using the first 1 468 octets. The remaining data are considered RemainingBitCM. When the last PDU is generated, the value of Bit 7 of seqNumber is 1.
CreateCyclicDataW-PDU	Generate CyclicDataW-PDU. If argument data exceeds 1 468 octets, CyclicDataW-PDU is generated using the first 1 468 octets. The remaining data are considered RemainingWordCM. When the last PDU is generated, the value of Bit 7 of seqNumber is 1.
CreateCyclicDataOut1-PDU	Generate CyclicDataOut1-PDU. If argument data exceeds 1 468 octets, CyclicDataOut1-PDU is generated using the first 1 468 octets. The remaining data are considered RemainingOutCM1. When the last PDU is generated, the value of Bit 7 of seqNumber is 1.
CreateCyclicDataOut2-PDU	Generate CyclicDataOu2-PDU. If argument data exceeds 1 468 octets, CyclicDataOut2-PDU is generated using the first 1 468 octets. The remaining data are considered RemainingOutCM2. When the last PDU is generated, the value of Bit 7 of seqNumber is 1.
CreateCyclicDataIn1-PDU	Generate CyclicDataIn1-PDU. If argument data exceeds 1 468 octets, CyclicDataIn1-PDU is generated using the first 1 468 octets. The remaining data are considered RemainingInCM1. When the last PDU is generated, the value of Bit 7 of seqNumber is set to 1.
CreateCyclicDataIn2-PDU	Generate CyclicDataIn2-PDU. If argument data exceeds 1 468 octets, CyclicDataIn2-PDU is generated using the first 1 468 octets. The remaining data are considered RemainingInCM2. When the last PDU is generated, the value of Bit 7 of seqNumber is 1.
RetrieveBitCM(PDU)	Retrieve Offset Address, Size, and BitCM data from PDU.
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 107 through Table 120.

Table 107 – 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 108 – 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
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

#	Current state	Event /condition => action	Next state
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; Stop ConnectTimer.	Initial
18	Connect	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK => OutPortState = OK; Start SendScanTimer.	Scan

#	Current state	Event /condition => action	Next state
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 109 – 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 => OutPortStart = LinkDown	Initial

#	Current state	Event /condition => action	Next state
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

#	Current state	Event /condition => action	Next state
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
29	Scan	InPort.ind(PDU) / PDU != Connect-PDU && PDU != ConnectAck-PDU && PDU != Scan-PDU =>	Scan

#	Current state	Event /condition => action	Next state
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

Table 110 – 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

#	Current state	Event /condition => action	Next state
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

<table>
<tr><th>#</th><th>Current state</th><th>Event
/condition
=> action</th><th>Next state</th></tr>
<tr><td>20</td><td>ScanWait</td><td>InPort.ind(source_address, Scan-PDU)
/ source_address != my address &&
InPortState == OK && OutPortState != OK &&
Latest(Scan-PDU) == TRUE
=>
Restart DetectScanTimer;
InPort.req(Scan-PDU)</td><td>ScanWait</td></tr>
<tr><td>21</td><td>ScanWait</td><td>InPort.ind(source_address, Scan-PDU)
/ source_address != my address &&
InPortState == OK &&
Latest(Scan-PDU) != TRUE
=>
Restart DetectScanTimer;</td><td>ScanWait</td></tr>
<tr><td>22</td><td>ScanWait</td><td>OutPort.ind(source_address, Scan-PDU)
/ source_address == my address &&
(InPortState == OK || OutPortState == OK)
=>
Restart DetectScanTimer.</td><td>ScanWait</td></tr>
<tr><td>23</td><td>ScanWait</td><td>OutPort.ind(source_address, Scan-PDU)
/ source_address != my address &&
InPortState != OK && OutPortState == OK &&
Latest(Scan-PDU) == TRUE
=>
Restart DetectScanTimer;
OutPort.req(Scan-PDU)</td><td>ScanWait</td></tr>
<tr><td>24</td><td>ScanWait</td><td>OutPort.ind(source_address, Scan-PDU)
/ source_address != my address &&
InPortState != OK && OutPortState == OK &&
Latest(Scan-PDU) != TRUE
=>
Restart DetectScanTimer;</td><td>ScanWait</td></tr>
<tr><td>25</td><td>ScanWait</td><td>OutPort.ind(source_address, Scan-PDU)
/ source_address != my address &&
InPortState == OK && OutPortState == OK &&
Latest(Scan-PDU) == TRUE
=>
InPort.req(Scan-PDU)</td><td>ScanWait</td></tr>
<tr><td>26</td><td>ScanWait</td><td>OutPort.ind(source_address, Scan-PDU)
/ source_address != my address &&
InPortState == OK && OutPortState == OK
=>
InPort.req(Scan-PDU)</td><td>ScanWait</td></tr>
<tr><td>27</td><td>ScanWait</td><td>InPort.ind(source_address, Scan-PDU)
/ source_address != my address &&
InPortState == Checking && OutPortState == OK
=>
InPortState = OK</td><td>Scan</td></tr>
<tr><td>28</td><td>ScanWait</td><td>OutPort.ind(source_address, Scan-PDU)
/ source_address != my address &&
InPortState == OK && OutPortState == Checking
=>
OutPortState = OK</td><td>Scan</td></tr>
<tr><td>29</td><td>ScanWait</td><td>OutPort.ind(Connect-PDU(PortChoice))
/ PortChoice == NTNTest
=>
OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));</td><td>NTNTestSlave</td></tr>
<tr><td>30</td><td>ScanWait</td><td>InPort.ind(Connect-PDU(PortChoice))
/ PortChoice == NTNTest
=>
InPortState = LinkDown;
OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));</td><td>NTNTestSlave</td></tr>
</table>

Table 111 – 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
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

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

#	Current state	Event /condition => action	Next state
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
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 112 – 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 => InPort.req(Connect-PDU(PortChoice=In));	CollectWait
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

#	Current state	Event /condition => action	Next state
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
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

#	Current state	Event /condition => action	Next state
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 113 – Connection control state machine – Select

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

#	Current state	Event /condition => action	Next state
13	Select	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK || OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	Select
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

<table>
<tr><th>#</th><th>Current state</th><th>Event
/condition
=> action</th><th>Next state</th></tr>
<tr><td>25</td><td>Select</td><td>OutPort.ind(Scan-PDU)
/ InPortState == OK && OutPortState == OK
=>
InPort.req(Scan-PDU)</td><td>Select</td></tr>
<tr><td>26</td><td>Select</td><td>InPort.ind(source_address, Select-PDU)
/ source_address == my address &&
InPortState == OK
=>
Start LaunchTimer.</td><td>TokenStartWait</td></tr>
<tr><td>27</td><td>Select</td><td>InPort.ind(source_address, Select-PDU)
/ source_address < my address &&
InPortState == OK && OutPortState != OK
=>
InPort.req(Select-PDU)</td><td>LaunchWait</td></tr>
<tr><td>28</td><td>Select</td><td>InPort.ind(source_address, Select-PDU)
/ source_address < my address &&
InPortState == OK && OutPortState == OK
=>
OutPort.req(Select-PDU)</td><td>LaunchWait</td></tr>
<tr><td>29</td><td>Select</td><td>InPort.ind(source_address, Select-PDU)
/ source_address > my address
=></td><td>Select</td></tr>
<tr><td>30</td><td>Select</td><td>OutPort.ind(source_address, Select-PDU)
/ source_address == my address &&
InPortState != OK && OutPortState == OK
=>
Start LaunchTimer.</td><td>TokenStartWait</td></tr>
<tr><td>31</td><td>Select</td><td>OutPort.ind(source_address, Select-PDU)
/ source_address < my address &&
InPortState != OK && OutPortState == OK
=>
OutPort.req(Select-PDU)</td><td>LaunchWait</td></tr>
<tr><td>32</td><td>Select</td><td>OutPort.ind(source_address, Select-PDU)
/ source_address > my address &&
InPortState != OK && OutPortState == OK
=></td><td>Select</td></tr>
<tr><td>33</td><td>Select</td><td>OutPort.ind(Select-PDU)
/ InPortState == OK && OutPortState == OK
=>
InPort.req(Select-PDU)</td><td>Select</td></tr>
<tr><td>34</td><td>Select</td><td>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
=></td><td>Select</td></tr>
</table>

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

Table 114 – 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

#	Current state	Event /condition => action	Next state
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 => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	TokenStartWait
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

#	Current state	Event /condition => action	Next state
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) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	TokenStartWait
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 115 – 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 => OutPortState = LinkDown	Scan
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

<table>
<tr><th>#</th><th>Current state</th><th>Event
/condition
=> action</th><th>Next state</th></tr>
<tr><td>12</td><td>LaunchWait</td><td>OutPort.ind(Connect-PDU(PortChoice))
/ PortChoice == Out &&
(InPortState == OK || OutPortState == OK) &&
OutPortState != LinkDown
=>
OutPort.req(ConnectAck-PDU(PortCheckResult=NG));
OutPortState = NG</td><td>LaunchWait</td></tr>
<tr><td>13</td><td>LaunchWait</td><td>InPort.ind(ConnectAck-PDU(PortCheckResult)
/ PortCheckResult == OK &&
InPortState == Checking && OutPortState == OK
=>
InPortState = OK</td><td>Scan</td></tr>
<tr><td>14</td><td>LaunchWait</td><td>InPort.ind(ConnectAck-PDU(PortCheckResult)
/ PortCheckResult == NG &&
InPortState == Checking && OutPortState == OK
=>
InPortState = NG</td><td>LaunchWait</td></tr>
<tr><td>15</td><td>LaunchWait</td><td>OutPort.ind(ConnectAck-PDU(PortCheckResult)
/ PortCheckResult == OK &&
InPortState == OK && OutPortState == Checking
=>
OutPortState = OK</td><td>Scan</td></tr>
<tr><td>16</td><td>LaunchWait</td><td>OutPort.ind(ConnectAck-PDU(PortCheckResult)
/ PortCheckResult == NG &&
InPortState == OK && OutPortState == Checking
=>
OutPortState = NG</td><td>LaunchWait</td></tr>
<tr><td>17</td><td>LaunchWait</td><td>OutPort.ind(Select-PDU)
/ InPortState == OK && OutPortState == OK
=>
InPort.req(Select-PDU)</td><td>LaunchWait</td></tr>
<tr><td>18</td><td>LaunchWait</td><td>InPort.ind(Scan-PDU)
/ InPortState == OK && OutPortState != OK
=>
InPort.req(Scan-PDU)</td><td>Scan</td></tr>
<tr><td>19</td><td>LaunchWait</td><td>InPort.ind(Scan-PDU)
/ InPortState == OK && OutPortState == OK
=>
OutPort.req(Scan-PDU)</td><td>Scan</td></tr>
<tr><td>20</td><td>LaunchWait</td><td>InPort.ind(Scan-PDU)
/ InPortState == Checking && OutPortState == OK
=>
InPortState = OK;
OutPort.req(Scan-PDU)</td><td>Scan</td></tr>
<tr><td>21</td><td>LaunchWait</td><td>OutPort.ind(Scan-PDU)
/ InPortState != OK && OutPortState == OK
=>
OutPort.req(Scan-PDU)</td><td>Scan</td></tr>
<tr><td>22</td><td>LaunchWait</td><td>OutPort.ind(Scan-PDU)
/ InPortState == OK && OutPortState == Checking
=>
OutPortState = OK;
InPort.req(Scan-PDU)</td><td>Scan</td></tr>
<tr><td>23</td><td>LaunchWait</td><td>OutPort.ind(Scan-PDU)
/ InPortState == OK && OutPortState == OK
=>
InPort.req(Scan-PDU)</td><td>LaunchWait</td></tr>
<tr><td>24</td><td>LaunchWait</td><td>InPort.ind(source_address, Select-PDU)
/ source_address == my address &&
=></td><td>LaunchWait</td></tr>
</table>

#	Current state	Event /condition => action	Next state
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) / InPortState == OK && OutPortState == OK => InPort.req(Launch-PDU)	LaunchWait
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 116 – 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 => InPort.req(Connect-PDU(PortChoice=In));	TokenReleaseWait
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

#	Current state	Event /condition => action	Next state
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
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

<table>
<tr><th>#</th><th>Current state</th><th>Event
/condition
=> action</th><th>Next state</th></tr>
<tr><td>25</td><td>TokenReleaseWait</td><td>OutPort.ind(Scan-PDU)
/ InPortState == OK && OutPortState == OK
=>
InPort.req(Scan-PDU)</td><td>TokenReleaseWait</td></tr>
<tr><td>26</td><td>TokenReleaseWait</td><td>InPort.ind(source_address, Select-PDU)
/ source_address < my address &&
InPortState == OK && OutPortState != OK
=>
InPort.req(Select-PDU)</td><td>LaunchWait</td></tr>
<tr><td>27</td><td>TokenReleaseWait</td><td>InPort.ind(source_address, Select-PDU)
/ source_address < my address &&
InPortState == OK && OutPortState == OK
=>
OutPort.req(Select-PDU)</td><td>LaunchWait</td></tr>
<tr><td>28</td><td>TokenReleaseWait</td><td>InPort.ind(source_address, Select-PDU)
/ source_address > my address
=></td><td>Select</td></tr>
<tr><td>29</td><td>TokenReleaseWait</td><td>OutPort.ind(source_address, Select-PDU)
/ source_address < my address &&
InPortState != OK && OutPortState == OK
=>
OutPort.req(Select-PDU)</td><td>LaunchWait</td></tr>
<tr><td>30</td><td>TokenReleaseWait</td><td>OutPort.ind(source_address, Select-PDU)
/ source_address > my address &&
InPortState != OK && OutPortState == OK
=></td><td>Select</td></tr>
<tr><td>31</td><td>TokenReleaseWait</td><td>OutPort.ind(Select-PDU)
/ InPortState == OK && OutPortState == OK
=>
InPort.req(Select-PDU)</td><td>TokenReleaseWait</td></tr>
<tr><td>32</td><td>TokenReleaseWait</td><td>InPort.ind(Launch-PDU)
=></td><td>TokenReleaseWait</td></tr>
<tr><td>33</td><td>TokenReleaseWait</td><td>OutPort.ind(Launch-PDU)
=></td><td>TokenReleaseWait</td></tr>
<tr><td>34</td><td>TokenReleaseWait</td><td>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
=></td><td>TokenReleaseWait</td></tr>
<tr><td>35</td><td>TokenReleaseWait</td><td>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) &&
OutPortState == OK
=></td><td>TokenReleaseWait</td></tr>
<tr><td>36</td><td>TokenReleaseWait</td><td>Control Cyclic.req(Control_Type)
/ Control_Type == Restart
=>
CyclicControl = Running;
Control Cyclic.cnf(State = Running)</td><td>TokenReleaseWait</td></tr>
</table>

#	Current state	Event /condition => action	Next state
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 117 – 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. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	TokenReleased

#	Current state	Event /condition => action	Next state
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

#	Current state	Event /condition => action	Next state
21	TokenReleased	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
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

#	Current state	Event /condition => action	Next state
35	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 => Restart NetworkWatchTimer.	TokenReleased
36	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
37	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
38	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
39	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
40	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

<table>
<tr><th>#</th><th>Current state</th><th>Event
/condition
=> action</th><th>Next state</th></tr>
<tr><td>41</td><td>TokenReleased</td><td>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.</td><td>TokenWait</td></tr>
<tr><td>42</td><td>TokenReleased</td><td>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 ||
PDU == CyclicDataIn1-PDU ||
PDU == CyclicDataIn2-PDU) &&
OutPortState == OK
=>
Restart NetworkWatchTimer.</td><td>TokenReleased</td></tr>
<tr><td>43</td><td>TokenReleased</td><td>OutPort.ind(source_address, PDU)
/ source_address != my address &&
(PDU == MyStatus-PDU ||
PDU == Dummy-PDU) &&
InPortState != OK && OutPortState == OK
=>
OutPort.req(PDU):
Restart NetworkWatchTimer.</td><td>TokenWait</td></tr>
<tr><td>44</td><td>TokenReleased</td><td>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.</td><td>TokenWait</td></tr>
<tr><td>45</td><td>TokenReleased</td><td>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.</td><td>TokenWait</td></tr>
<tr><td>46</td><td>TokenReleased</td><td>OutPort.ind(source_address, PDU)
/ source_address != my address &&
(PDU == MyStatus-PDU ||
PDU == Dummy-PDU &&
InPortState == OK && OutPortState == OK
=>
InPort.req(PDU)</td><td>TokenReleased</td></tr>
</table>

#	Current state	Event /condition => action	Next state
47	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
48	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
49	TokenReleased	Control Cyclic.req(Control_Type) / Control_Type == Restart => CyclicControl = Running; Control Cyclic.cnf(State = Running)	TokenReleased
50	TokenReleased	Control Cyclic.req(Control_Type) / Control_Type == Stop => CyclicControl = Stop; Control Cyclic.cnf(State = Stop)	TokenReleased
51	TokenReleased	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
52	TokenReleased	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 118 – Connection control state machine – TokenWait

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

#	Current state	Event /condition => action	Next state
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 => InPort.req(Scan-PDU)	TokenWait
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

<table>
<tr><th>#</th><th>Current state</th><th>Event
/condition
=> action</th><th>Next state</th></tr>
<tr><td>26</td><td>TokenWait</td><td>InPort.ind(source_address, Select-PDU)
/ source_address > my address &&
InPortState == OK
=></td><td>Select</td></tr>
<tr><td>27</td><td>TokenWait</td><td>OutPort.ind(source_address, Select-PDU)
/ source_address < my address &&
InPortState != OK && OutPortState == OK
=>
OutPort.req(Select-PDU)</td><td>LaunchWait</td></tr>
<tr><td>28</td><td>TokenWait</td><td>OutPort.ind(source_address, Select-PDU)
/ source_address > my address &&
InPortState != OK && OutPortState == OK
=></td><td>Select</td></tr>
<tr><td>29</td><td>TokenWait</td><td>OutPort.ind(source_address, Select-PDU)
/ InPortState == OK && OutPortState == OK
=>
InPort.req(Select-PDU)</td><td>TokenWait</td></tr>
<tr><td>30</td><td>TokenWait</td><td>InPort.ind(Launch-PDU)
/ InPortState == OK && OutPortState != OK
=>
InPort.req(Launch -PDU)</td><td>TokenWait</td></tr>
<tr><td>31</td><td>TokenWait</td><td>InPort.ind(Launch-PDU)
/ InPortState == OK && OutPortState == OK
=>
OutPort.req(Launch -PDU)</td><td>TokenWait</td></tr>
<tr><td>32</td><td>TokenWait</td><td>OutPort.ind(Launch-PDU)
/ InPortState != OK && OutPortState == OK
=>
OutPort.req(Launch -PDU)</td><td>TokenWait</td></tr>
<tr><td>33</td><td>TokenWait</td><td>OutPort.ind(Launch-PDU)
/ InPortState == OK && OutPortState == OK
=>
InPort.req(Launch -PDU)</td><td>TokenWait</td></tr>
<tr><td>34</td><td>TokenWait</td><td>InPort.ind(Token-PDU)
/ InPortState == OK
=></td><td>TokenReleaseWait</td></tr>
<tr><td>35</td><td>TokenWait</td><td>OutPort.ind(Token-PDU)
/ InPortState != OK && OutPortState == OK
=></td><td>TokenReleaseWait</td></tr>
<tr><td>36</td><td>TokenWait</td><td>OutPort.ind(Token-PDU)
/ InPortState == OK && OutPortState == OK
=>
InPort.req(Token-PDU)</td><td>TokenWait</td></tr>
<tr><td>37</td><td>TokenWait</td><td>InPort.ind(source_address, PDU)
/ (PDU == MyStatus-PDU ||
PDU == Dummy-PDU) &&
InPortState == OK && OutPortState != OK
=>
InPort.req(PDU);
Restart NetworkWatchTimer.</td><td>TokenWait</td></tr>
<tr><td>38</td><td>TokenWait</td><td>InPort.ind(source_address, PDU)
/ (PDU ==Transient1-PDU || PDU ==Transient2-PDU) &&
InPortState == OK && OutPortState != OK
=>
ACReceived(PDU);
InPort.req(PDU);
Restart NetworkWatchTimer.</td><td>TokenWait</td></tr>
</table>

#	Current state	Event /condition => action	Next state
39	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
40	TokenWait	InPort.ind(source_address, PDU) / (PDU == MyStatus-PDU || PDU == Dummy-PDU) && InPortState == OK && OutPortState == OK => OutPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
41	TokenWait	InPort.ind(source_address, PDU) / (PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState == OK && OutPortState == OK => ACReceived(PDU); OutPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
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); OutPort.req(PDU); Restart NetworkWatchTimer	TokenWait
43	TokenWait	OutPort.ind(source_address, PDU) / (PDU == MyStatus-PDU || PDU == Dummy-PDU) && InPortState != OK&& OutPortState == OK => OutPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
44	TokenWait	OutPort.ind(source_address, PDU) / (PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState != OK&& OutPortState == OK => ACReceived(PDU); OutPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
45	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

#	Current state	Event /condition => action	Next state
46	TokenWait	OutPort.ind(source_address, PDU) / (PDU == MyStatus-PDU || PDU == Dummy-PDU) && InPortState == OK && OutPortState == OK => InPort.req(PDU)	TokenWait
47	TokenWait	OutPort.ind(source_address, PDU) / PDU ==(PDU ==Transient1-PDU || PDU ==Transient2-PDU) && InPortState == OK && OutPortState == OK => ACReceived(PDU); InPort.req(PDU)	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); InPort.req(PDU)	TokenWait
49	TokenWait	Control Cyclic.req(Control_Type) / Control_Type == Restart => CyclicControl = Running; Control Cyclic.cnf(State = Running)	TokenWait
50	TokenWait	Control Cyclic.req(Control_Type) / Control_Type == Stop => CyclicControl = Stop; Control Cyclic.cnf(State = Stop)	TokenWait
51	TokenWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
52	TokenWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

Table 119 – 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)) / (PortCheckResut == NTNTestNG || PortCheckResut == NTNTestOK) && Tries != 0 => OutPort.req(NTNTest-PDU); Tries = Tries – 1; NTNTestTimer is restarted.	NTNTestMaster
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 120 – 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 121.

Table 121 – 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 122.

Table 122 – 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
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

#	Current state	Event /condition => action	Next state
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
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

#	Current state	Event /condition => action	Next state
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); CPChanged = TRUE	ReceivedCPChecking
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

#	Current state	Event /condition => action	Next state
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 => CPWCR.req(CheckResult = CPNotReceived)	SlaveInit
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 123.

Table 123 – 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 124.

Table 124 – 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.

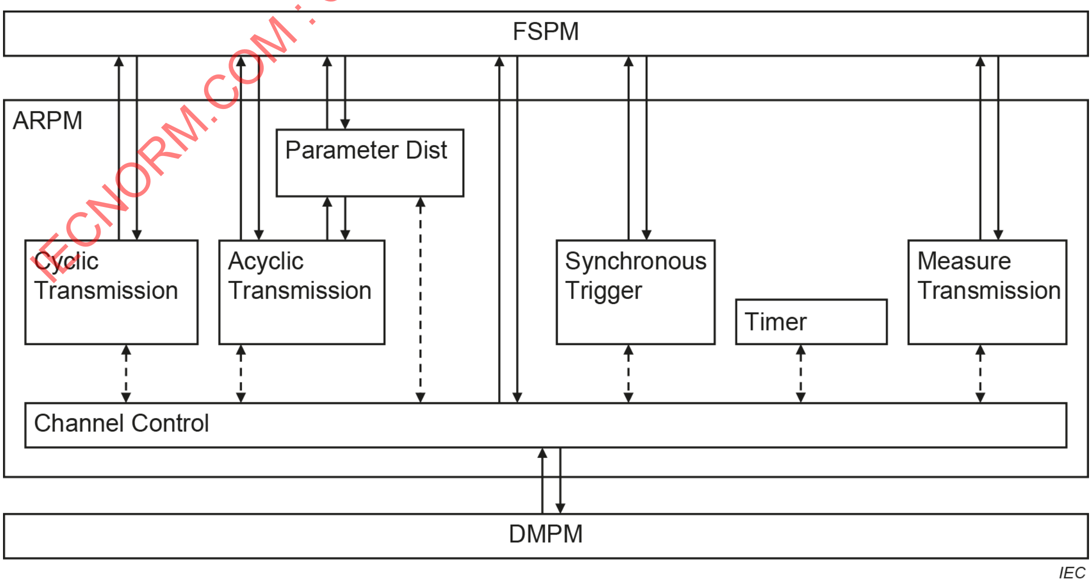

Figure 22 – Structure of ARPM F

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 125 and the details are specified in Table 126.

Table 125 – 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 126 – 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) => ACTicket = FALSE	IDLE
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

#	Current state	Event /condition => action	Next state
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
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 127.

Table 127 – 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 128.

Table 128 – 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 129 and the details are specified in Table 130.

Table 129 – 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 130 – 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

6	RXRYSENDER	/ RXRYSent != TRUE && sending == TRUE && Length(RemainingRXRY) > 0 => pdu = CreateCyclicDataRXRY-PDU(seqno, type, RemainingRXRY); SendCyclic(pdu); seqno = seqno + 1	RXRYSENDER

#	Current state	Event /condition => action	Next state
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	RWrRWwSENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	RWrRWwSENDER
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 131.

Table 131 – 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 1 468 octets from the start when RXRY given as the argument exceeds 1 468 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 1 468 octets from the start when RXRY given as the argument exceeds 1 468 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 132.

Table 132 – 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 132 with the details specified in Table 135 through Table 149.

The states of the channel control state machine for the Slave station are described in Table 134 with the details specified in Table 150 through Table 155.

Table 133 – 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
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 134 – Slave station channel control states

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

Table 135 – Master station state table – MasterDown

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

Table 136 – 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) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	Listen
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

#	Current state	Event /condition => action	Next state
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	PrimaryMasterScatterTD

Table 137 – Master station state table – MasterArbitration

#	Current state	Event /condition => action	Next state
1	MasterArbitratio n	/ TR2 == TRUE => SynchronizationMaster=FALSE TR2 = FALSE; RvLastArbTimer startup; ChannelGroup = NULL; SendArbTimer startup; pdu = CreatePersuasion(); Send.req (port, BROADCAST, pdu) to all ports	MasterArbitratio n
2	MasterArbitratio n	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	MasterArbitratio n
3	MasterArbitratio n	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	MasterArbitratio n	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitratio n
5	MasterArbitratio n	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitratio n
6	MasterArbitratio n	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitratio n

#	Current state	Event /condition => action	Next state
7	MasterArbitratio n	Receive.ind(rport, DA, SA, ftype, pdu) / SA != MyAddrftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitratio n
8	MasterArbitratio n	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	MasterArbitratio n
9	MasterArbitratio n	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	MasterArbitratio n
10	MasterArbitratio n	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitratio n
11	MasterArbitratio n	SendArbTimer time-out => pdu = CreatePersuasion(); Send.req (port, BROADCAST, pdu) to all ports; SendArbTimer restart	MasterArbitratio n
12	MasterArbitratio n	RvLastArbTimer timeout => SendArbTimer stop; RvLastArbTimer stop; TR3 = TRUE	PrimaryMasterS catterTD

Table 138 – 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	MasterArbitrati on

#	Current state	Event /condition => action	Next state
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	MasterArbitrati on
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	MasterArbitrati on
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 ScatterTD	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 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

<table>
<tr><th>#</th><th>Current state</th><th>Event
/condition
=> action</th><th>Next state</th></tr>
<tr><td>13</td><td>PrimaryMaster
ScatterTD</td><td>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)</td><td>PrimaryMaster
ScatterTD</td></tr>
<tr><td>14</td><td>PrimaryMaster
ScatterTD</td><td>Receive.ind(rport, DA, SA, ftype, pdu)
/
DA == MyAddr && SA != MyAddr &&
(ftype == 0x23 || ftype == 0x25 || ftype == 0x29 ||
ftype == 0x2E)
=>
ACReceived(pdu)</td><td>PrimaryMaster
ScatterTD</td></tr>
<tr><td>15</td><td>PrimaryMaster
ScatterTD</td><td>Receive.ind(rport, DA, SA, ftype, pdu)
/
DA != MyAddr && SA != MyAddr
=>
Send.req (sport, DA, pdu) to all ports other than rport</td><td>PrimaryMaster
ScatterTD</td></tr>
<tr><td>16</td><td>PrimaryMaster
ScatterTD</td><td>TDAckTimer time-out
/
Nodes > 0
=>
TDAckTimer stop;
TR4 = TRUE</td><td>PrimaryMaster
SettingUp</td></tr>
<tr><td>17</td><td>PrimaryMaster
ScatterTD</td><td>TDAckTimer time-out
/
Nodes == 0
=>
TDAckTimer stop;
TR3 = TRUE</td><td>PrimaryMaster
ScatterTD</td></tr>
<tr><td>18</td><td>PrimaryMaster
ScatterTD</td><td>/
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</td><td>PrimaryMaster
ScatterTD</td></tr>
<tr><td>19</td><td>PrimaryMaster
ScatterTD</td><td>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</td><td>PrimaryMaster
ScatterTD</td></tr>
<tr><td>20</td><td>PrimaryMaster
ScatterTD</td><td>TDAckTimer time-out
/
Nodes > 0 && RingCheck == TRUE && IsValidPort(port 1) !=
TRUE
=>
TDAckTimer stop;
TR4 = TRUE</td><td>PrimaryMaster
SettingUp</td></tr>
</table>

Table 139 – 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	MasterArbitrati on
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	MasterArbitrati on
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 && SARcvd+1 < SetupNodes => SARcvd++;pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) to all ports	PrimaryMaster SettingUp
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

#	Current state	Event /condition => action	Next state
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; 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	PrimaryMaster SettingUp

<table>
<tr><th>#</th><th>Current state</th><th>Event
/condition
=> action</th><th>Next state</th></tr>
<tr><td>18</td><td>PrimaryMaster
SettingUp</td><td>
/
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</td><td>PrimaryMaster
SettingUp</td></tr>
<tr><td>19</td><td>PrimaryMaster
SettingUp</td><td>
/
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</td><td>PrimaryMaster
SettingUp</td></tr>
<tr><td>20</td><td>PrimaryMaster
SettingUp</td><td>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</td><td>PrimaryMaster
SettingUp</td></tr>
<tr><td>21</td><td>PrimaryMaster
SettingUp</td><td>Receive.ind(rport, DA, SA, ftype, pdu)
/
RingChecked == TRUE && DA == MyAddr &&
SA != MyAddr && ftype == 0x14 &&
SlavePortValidating == TRUE
=>
SlavePortValidating = FALSE;
SAckTimer stop;
TR5 = TRUE</td><td>PrimaryMaster
HoldToken</td></tr>
</table>

Table 140 – 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; TraAvailHopCounter = 0; else TraAvailHopCounter = token-PDU.traLastHopCounter; TraLastHopCounter = 0; TraAllows = Max Transients per round; TokenHopCounter = 0; TraAllowsND = GetTraAllowsND(); SetNextPhase(PrimaryMasterHoldToken,0);	PrimaryMaster HoldToken
2	PrimaryMaster HoldToken	/ PMChangeFlag == ON => TR2 = TRUE	MasterArbitrati on
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	PrimaryMasterI nviting
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

#	Current state	Event /condition => action	Next state

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;
		Enque(pdu);
		sents = 1;
		PH = 4
			PrimaryMaster
			HoldToken

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	MasterArbitrati on
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	MasterArbitrati on
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