

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

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



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2023 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester. If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'IEC ou du Comité national de l'IEC du pays du demandeur. Si vous avez des questions sur le copyright de l'IEC ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de l'IEC de votre pays de résidence.

IEC Secretariat
3, rue de Varembe
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 It also gives information on projects, replaced and withdrawn publications.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.

A propos de l'IEC

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

A propos des publications IEC

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

Recherche de publications IEC -

webstore.iec.ch/advsearchform

La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études, ...). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et une fois par mois par email.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: sales@iec.ch.

IEC Products & Services Portal - products.iec.ch

Découvrez notre puissant moteur de recherche et consultez gratuitement tous les aperçus des publications. Avec un abonnement, vous aurez toujours accès à un contenu à jour adapté à vos besoins.

Electropedia - www.electropedia.org

Le premier dictionnaire d'électrotechnologie en ligne au monde, avec plus de 22 300 articles terminologiques en anglais et en français, ainsi que les termes équivalents dans 19 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.



IEC 61158-6-23

Edition 3.0 2023-03

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

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

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

ICS 25.040.40, 35.100.70, 35.110

ISBN 978-2-8322-7815-4

Warning! Make sure that you obtained this publication from an authorized distributor. Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.

CONTENTS

FOREWORD.....	14
INTRODUCTION.....	16
1 Scope.....	17
1.1 General.....	17
1.2 Specifications	18
1.3 Conformance	18
2 Normative references	18
3 Terms, definitions, symbols, abbreviated terms and conventions	19
3.1 Referenced terms and definitions.....	19
3.1.1 ISO/IEC 7498-1 terms.....	19
3.1.2 ISO/IEC 8822 terms.....	19
3.1.3 IEC 61158-1 terms.....	19
3.2 Additional Type 23 terms and definitions.....	19
3.3 Symbols and abbreviated terms	22
3.4 Conventions.....	23
3.4.1 General concept	23
3.4.2 Convention for the encoding of reserved bits and octets	23
3.4.3 Conventions for abstract syntax description	23
3.4.4 Conventions for bit description in octets	23
3.4.5 Conventions for state machine descriptions	24
4 FAL syntax description	25
4.1 FALPDU type C abstract syntax	25
4.1.1 Basic abstract syntax.....	25
4.1.2 Connect-PDU	26
4.1.3 ConnectAck-PDU.....	26
4.1.4 Scan-PDU	26
4.1.5 Collect-PDU.....	27
4.1.6 Select-PDU.....	27
4.1.7 Launch-PDU	27
4.1.8 Token-PDU.....	27
4.1.9 MyStatus-PDU.....	28
4.1.10 Transient1-PDU	28
4.1.11 Dummy-PDU.....	29
4.1.12 Transient2-PDU	29
4.1.13 NTNTest-PDU.....	29
4.1.14 CyclicDataW-PDU.....	30
4.1.15 CyclicDataB-PDU	30
4.1.16 CyclicDataOut1-PDU	30
4.1.17 CyclicDataOut2-PDU	31
4.1.18 CyclicDataIn1-PDU	31
4.1.19 CyclicDataIn2-PDU	31
4.2 FALPDU type F abstract syntax	32
4.2.1 Basic abstract syntax.....	32
4.2.2 Persuasion-PDU	33
4.2.3 TestData-PDU	34
4.2.4 TestDataAck-PDU.....	34
4.2.5 Setup-PDU	35

4.2.6	SetupAck-PDU	35
4.2.7	F-Token-PDU	36
4.2.8	F-MyStatus-PDU	36
4.2.9	Measure-PDU	36
4.2.10	F-Offset-PDU	37
4.2.11	F-Update-PDU	37
4.2.12	F-CyclicData-PDU	37
4.2.13	Transient1-PDU	37
4.2.14	TransientAck-PDU	41
4.2.15	Transient2-PDU	42
4.2.16	ParamCheck-PDU	42
4.2.17	Parameter-PDU	43
4.2.18	Timer-PDU	44
4.3	Data type assignments for type C	45
4.4	Data type assignments for type F	46
4.5	FALPDU type T abstract syntax	47
4.5.1	Basic abstract syntax	47
4.5.2	CyclicM-PDU	50
4.5.3	CyclicS-PDU	50
4.5.4	CyclicMs-PDU	50
4.5.5	CyclicSs-PDU	51
4.5.6	AcyclicPriority-PDU	52
4.5.7	AcyclicDetection-PDU	52
4.5.8	AcyclicDetectionAck-PDU	53
4.5.9	AcyclicTestDataHeader	54
4.5.10	AcyclicTestDataHeader	55
4.5.11	AcyclicData-PDU	55
4.5.12	PtpSync-PDU	55
4.5.13	PtpPdelayReq-PDU	55
4.5.14	PtpPdelayResp-PDU	55
4.5.15	PtpFollowUp-PDU	56
4.5.16	PtpPdelayRespFollowUp-PDU	56
4.5.17	PtpAnnounce-PDU	56
4.5.18	SImpIPAddressSet-PDU	56
4.5.19	SImpNetworkConfigMain-PDU	56
4.5.20	SImpNetworkConfigTslt-PDU	56
4.5.21	SImpNotification-PDU	56
4.5.22	SImpMasterConfig-PDU	57
4.5.23	SImpSlaveConfig-PDU	57
4.5.24	SImpCyclicConfigMain-PDU	57
4.5.25	SImpCyclicConfigTrnSubPayload-PDU	57
4.5.26	SImpCyclicConfigRcvSubPayload-PDU	57
4.5.27	SImpCyclicConfigRcvSrcInfo-PDU	58
4.5.28	SImpLinkDevicePrmWrite-PDU	58
4.5.29	SImpLinkDevicePrmWriteCheckRequest-PDU	58
4.5.30	SImpLinkDevicePrmWriteCheckResponse-PDU	58
4.5.31	SImpNMTStateUpload-PDU	58
4.5.32	SImpNMTStateDownload-PDU	59
4.5.33	SImpReadObject-PDU	59

4.5.34	SImpWriteObject-PDU	59
4.5.35	SImpObjectSubIDReadBlock-PDU	59
4.5.36	SImpObjectSubIDWriteBlock-PDU	59
4.5.37	SImpGetODList-PDU	59
4.5.38	SImpGetObjectDescription-PDU	60
4.5.39	SImpGetEntryDescription-PDU	60
4.5.40	SImpStopOwnStationCyclic-PDU	60
4.5.41	SImpStartOwnStationCyclic-PDU	60
4.5.42	SImpStopOtherStationsCyclic-PDU	60
4.5.43	SImpStartOtherStationsCyclic-PDU	60
4.5.44	SImpAllParameterGet-PDU	61
4.5.45	SImpParameterGet-PDU	61
4.5.46	SImpAllParameterSizeGet-PDU	61
4.5.47	SImpParameterSizeGet-PDU	61
4.5.48	SImpStationSubIDListGet-PDU	61
4.5.49	SImpDeviceIdentificationInfoGet-PDU	61
4.5.50	SImpDataMonitoring-PDU	62
4.5.51	SImpAllParameterSet-PDU	62
4.5.52	SImpParameterSet-PDU	62
4.5.53	SImpParameterVersionCheck-PDU	62
4.5.54	SImpDeviceIdentificationInfoCompare-PDU	62
4.5.55	SImpNodeSearch-PDU	62
4.5.56	SImpIPAddressSet-PDU	63
4.5.57	SImpDeviceInfoCompare-PDU	63
4.5.58	SImpParameterGet-PDU	63
4.5.59	SImpParameterSet-PDU	63
4.5.60	SImpParameterSetStart-PDU	63
4.5.61	SImpParameterSetEnd-PDU	63
4.5.62	SImpVerifyCheckCode-PDU	64
4.5.63	SImpOutputMapFileNameGet-PDU	64
4.5.64	SImpNewFile-PDU	64
4.5.65	SImpParameterSetCancel-PDU	64
4.5.66	SImpOpenFile-PDU	64
4.5.67	SImpCloseFile-PDU	64
4.5.68	SImpReadFile-PDU	65
4.5.69	SImpWriteFile-PDU	65
4.5.70	SImpStatusRead-PDU	65
4.5.71	SImpCommunicationSettingGet-PDU	65
4.5.72	SImpGetDeviceInfo-PDU	65
4.5.73	SImpGetBackupListFileName-PDU	65
4.5.74	SImpStartBackup-PDU	66
4.5.75	SImpEndBackup-PDU	66
4.5.76	SImpCheckRestoreVersion-PDU	66
4.5.77	SImpStartRestore-PDU	66
4.5.78	SImpEndRestore-PDU	66
4.5.79	SImpStatusRead2-PDU	66
4.5.80	SImpReqSearchNode-PDU	67
4.5.81	SImpGetSearchNodeState-PDU	67
4.5.82	SImpGetNodeList-PDU	67

4.5.83	SImpReqSetIPAddress-PDU	67
4.5.84	SImpSearchPrmControlStation-PDU	67
4.5.85	SImpRequestRestore-PDU	68
4.5.86	SImpCheckPrmDelivery-PDU	68
4.5.87	SImpRsvStationConfigTemporaryRelease-PDU	68
4.5.88	SImpRsvStationConfig-PDU	68
4.5.89	SImpGetEventNum-PDU	68
4.5.90	SImpGetEventHistory-PDU	69
4.5.91	SImpClearEventHistory-PDU	69
4.5.92	SImpClockOffsetDataSend-PDU	69
4.5.93	SImpSetWatchdogCounterInfo-PDU	69
4.5.94	SImpWatchdogCounterOffsetConfig-PDU	69
4.5.95	SImpRemoteReset-PDU	69
4.5.96	SImpGetCommunicationSet-PDU	70
4.5.97	SImpGetStationSubIDList-PDU	70
4.5.98	SImpGetDeviceInfo-PDU	70
4.5.99	SImpStartBackup-PDU	70
4.5.100	SImpEndBackup-PDU	70
4.5.101	SImpRequestBackup-PDU	70
4.5.102	SImpGetBackupPrm-PDU	71
4.5.103	SImpCheckRestore-PDU	71
4.5.104	SImpStartRestore-PDU	71
4.5.105	SImpEndRestore-PDU	71
4.5.106	SImpSetBackupPrm-PDU	71
4.5.107	SImpLinkupSpeed-PDU	71
4.5.108	SImpNodeIndication-PDU	72
4.6	Data type assignments for type T	72
5	FAL transfer syntax	72
5.1	Encoding rules	72
5.1.1	Unsigned encoding	72
5.1.2	Octet string encoding	72
5.1.3	SEQUENCE encoding	72
5.1.4	LOctetString encoding	72
5.2	FALPDU type C elements encoding	73
5.2.1	FALARHeader	73
5.2.2	Connect-PDU	75
5.2.3	ConnectAck-PDU	76
5.2.4	Scan-PDU	76
5.2.5	Collect-PDU	77
5.2.6	Select-PDU	79
5.2.7	Launch-PDU	80
5.2.8	Token-PDU	80
5.2.9	MyStatus-PDU	80
5.2.10	Transient1-PDU	82
5.2.11	Dummy-PDU	86
5.2.12	Transient2-PDU	87
5.2.13	NTNTest-PDU	98
5.2.14	CyclicDataW-PDU	98
5.2.15	CyclicDataB-PDU	99

5.2.16	CyclicDataOut1-PDU	100
5.2.17	CyclicDataOut2-PDU	100
5.2.18	CyclicDataIn1-PDU	101
5.2.19	CyclicDataIn2-PDU	102
5.3	FALPDU type F elements encoding	103
5.3.1	FALARHeader	103
5.3.2	Persuasion-PDU	108
5.3.3	TestData-PDU	109
5.3.4	TestDataAck-PDU	109
5.3.5	Setup-PDU	111
5.3.6	SetupAck-PDU	113
5.3.7	F-Token-PDU	114
5.3.8	F-Measure-PDU	115
5.3.9	F-Offset-PDU	116
5.3.10	F-Update-PDU	116
5.3.11	F-MyStatus-PDU	116
5.3.12	F-CyclicData-PDU	122
5.3.13	Transient1-PDU	123
5.3.14	TransientAck-PDU	128
5.3.15	Transient2-PDU	129
5.3.16	ParamCheck-PDU	132
5.3.17	Parameter-PDU	133
5.3.18	Timer-PDU	140
5.4	FALPDU type T elements encoding	141
5.4.1	CyclicM-PDU	141
5.4.2	CyclicS-PDU	144
5.4.3	CyclicMs-PDU	146
5.4.4	CyclicSs-PDU	147
5.4.5	AcyclicPriority-PDU	148
5.4.6	AcyclicDetection-PDU	150
5.4.7	AcyclicDetectionAck-PDU	151
5.4.8	AcyclicTestData-PDU	157
5.4.9	AcyclicTestDataAck-PDU	160
5.4.10	AcyclicData-PDU	162
5.4.11	Ptp-PDU	162
5.4.12	IpData-PDU	164
6	Structure of the FAL protocol state machine	164
7	FAL service protocol machine (FSPM)	165
7.1	Overview	165
7.2	FSPM type C	165
7.2.1	Overview	165
7.2.2	FSPM	166
7.3	FSPM type F	169
7.3.1	Overview	169
7.3.2	FSPM	171
7.4	FSPM type T	176
7.4.1	Overview	176
7.4.2	FSPM State Machine	177
8	Application relationship protocol machine (ARPM)	181

8.1	ARPM type C	181
8.1.1	Overview	181
8.1.2	Acyclic transmission	182
8.1.3	Cyclic transmission	183
8.1.4	Connection control	188
8.1.5	Common parameter dist	227
8.2	ARPM type F	232
8.2.1	Overview	232
8.2.2	Acyclic transmission	233
8.2.3	Cyclic transmission	235
8.2.4	Channel control	238
8.2.5	Parameter dist	278
8.2.6	Synchronous trigger	282
8.2.7	Timer	283
8.2.8	Measure transmission	284
8.3	ARPM type T	289
8.3.1	Overview	289
8.3.2	Cyclic Transmission	289
8.3.3	Acyclic Transmission	291
8.3.4	Channel Control	294
8.3.5	TimeSync Control	297
8.3.6	IPTrans Control	299
8.3.7	Handler	300
9	DLL mapping protocol machine (DMPM)	305
9.1	DMPM type C	305
9.2	DMPM type F	306
9.3	DMPM type T	307
	Bibliography	309
	Figure 1 – Bit description in octets	24
	Figure 2 – Structure for memory access information retrieve response	90
	Figure 3 – Attribute definitions	90
	Figure 4 – Access code definitions	91
	Figure 5 – Structure for RUN request	92
	Figure 6 – Structure for RUN response	93
	Figure 7 – Structure for STOP request	93
	Figure 8 – Structure for STOP response	93
	Figure 9 – Structure for batch memory read request	94
	Figure 10 – Structure for batch memory read response	94
	Figure 11 – Structure for random memory read request	95
	Figure 12 – Structure for random memory read response	95
	Figure 13 – Structure for batch memory write request	96
	Figure 14 – Structure for batch memory write response	96
	Figure 15 – Structure for random memory write request	97
	Figure 16 – Structure for random memory write response	97
	Figure 17 – Relationships between protocol machines	165

Figure 18 – Structure of FSPM C	166
Figure 19 – Structure of FSPM F	169
Figure 20 – Structure of FSPM T	176
Figure 21 – Structure of ARPM C	181
Figure 22 – Structure of ARPM F	232
Figure 23 – Structure of ARPM T	289
Figure 24 – Structure of type C DMPM	305
Figure 25 – Structure of type F DMPM	306
Figure 26 – Structure of type T DMPM	307
Table 1 – State machine description elements	24
Table 2 – Description of state machine elements	24
Table 3 – Conventions used in state machines	25
Table 4 – afFType	73
Table 5 – priority	74
Table 6 – portChoice	75
Table 7 – portCheckResult	76
Table 8 – dstPortInfo	76
Table 9 – scanState	76
Table 10 – nodeType	77
Table 11 – loopState	78
Table 12 – Cyclic status	78
Table 13 – Parameter setting mode	78
Table 14 – opState	81
Table 15 – errorState	81
Table 16 – Data type	83
Table 17 – CPW	83
Table 18 – CPWC	84
Table 19 – CPWCR	84
Table 20 – cmParam	84
Table 21 – Details of param area	85
Table 22 – Details of application parameters	85
Table 23 – Details of LB/LW CM area and LB/LW CM additional area	86
Table 24 – Details of LX/LY CM 1 area and LX/LY CM 2 area	86
Table 25 – Destination module flag	88
Table 26 – Command types	89
Table 27 – Access codes of network module memory	91
Table 28 – Access codes of controller memory	92
Table 29 – byteValidity	98
Table 30 – afFType	104
Table 31 – dataType	105
Table 32 – varField	106
Table 33 – nodeType	107

Table 34 – ProtocolVerType.....	108
Table 35 – Link status.....	111
Table 36 – Port enable/disable specification	112
Table 37 – Cyclic transmission parameter hold status.....	118
Table 38 – Detailed application operation status	119
Table 39 – Error detection status	119
Table 40 – Slave-specific event reception status.....	121
Table 41 – dataSupType of dataType (0x07).....	124
Table 42 – FieldSpecificTransient opHeader.....	124
Table 43 – command (dataType: 0x07, dataSubType: 0x0002)	125
Table 44 – subCommand type for each command type	125
Table 45 – Structure of Deliver node information	125
Table 46 – Structure of Deliver node information – message.....	126
Table 47 – Structure of Get statistical information response.....	127
Table 48 – Structure of Acquisition of node details response	128
Table 49 – Execution module specification.....	130
Table 50 – Command type	131
Table 51 – frameType	141
Table 52 – cycleNo	141
Table 53 – sa.....	141
Table 54 – da.....	142
Table 55 – commInfo	142
Table 56 – txAsynInfo	142
Table 57 – seqNo	143
Table 58 – Upper one octet of diagnosisData.....	143
Table 59 – Lower three octets of diagnosisData.....	144
Table 60 – Upper one octet of diagnosisData.....	145
Table 61 – Lower three octets of diagnosisData.....	145
Table 62 – mngPriority.....	149
Table 63 – KindFlag.....	149
Table 64 – previousNodePort.....	151
Table 65 – optionFlag	151
Table 66 – sendInfo	151
Table 67 – nodeType	152
Table 68 – IP address 4th octet	152
Table 69 – detectionRcvPort.....	153
Table 70 – myPort	153
Table 71 – Four bits of myPortLinkStatus.....	153
Table 72 – Four bits of myPortFilterStatus	154
Table 73 – performance	154
Table 74 – gmPriority.....	155
Table 75 – syncType.....	155
Table 76 – cyclicSize	156

Table 77 – function	157
Table 78 – optionInfo	157
Table 79 – Protocol version	158
Table 80 – Protocol type	159
Table 81 – Source information	159
Table 82 – Link status	162
Table 83 – Cyclic data state table	167
Table 84 – Acyclic data state table	167
Table 85 – Management state table	169
Table 86 – Cyclic data state table	172
Table 87 – Acyclic data state table	172
Table 88 – Management state table	175
Table 89 – Synchronization state table	175
Table 90 – Measurement state table	176
Table 91 – Primitives provided by FSPM	177
Table 92 – Cyclic Data state	177
Table 93 – Cyclic Data state table	178
Table 94 – Function used for Cyclic Data	178
Table 95 – Acyclic Data state	178
Table 96 – Acyclic Data state table	178
Table 97 – Management state	179
Table 98 – Management state table	180
Table 99 – TimeSync Data state	180
Table 100 – TimeSync Data state table	180
Table 101 – SLMP Data state	180
Table 102 – SLMP Data state table	181
Table 103 – Acyclic transmission state table	182
Table 104 – Acyclic transmission functions	183
Table 105 – Cyclic transmission state table	183
Table 106 – Cyclic transmission functions	188
Table 107 – Connection control state machine – Initial	189
Table 108 – Connection control state machine – Connect	189
Table 109 – Connection control state machine – Scan	191
Table 110 – Connection control state machine – ScanWait	194
Table 111 – Connection control state machine – Collect	197
Table 112 – Connection control state machine – CollectWait	200
Table 113 – Connection control state machine – Select	203
Table 114 – Connection control state machine – TokenStartWait	206
Table 115 – Connection control state machine – LaunchWait	209
Table 116 – Connection control state machine – TokenReleaseWait	212
Table 117 – Connection control state machine – TokenReleased	215
Table 118 – Connection control state machine – TokenWait	221
Table 119 – Connection control state machine – NTNTestMaster	226

Table 120 – Connection control state machine – NTNTestSlave	226
Table 121 – Function list of connection control	227
Table 122 – Common parameter dist state table	227
Table 123 – Function list of connection control	231
Table 124 – Mapping of internal service and acyclic transmission service	232
Table 125 – Acyclic transmission states	233
Table 126 – Acyclic transmission state table	233
Table 127 – Acyclic transmission functions	235
Table 128 – Acyclic transmission variables	235
Table 129 – Cyclic transmission states	236
Table 130 – Cyclic transmission state table	236
Table 131 – Cyclic transmission functions	238
Table 132 – Cyclic transmission variables	238
Table 133 – Master station channel control states	239
Table 134 – Slave station channel control states	239
Table 135 – Master station state table – MasterDown	239
Table 136 – Master station state table – Listen	240
Table 137 – Master station state table – MasterArbitration	241
Table 138 – Master station state table – PrimaryMasterScatterTD	242
Table 139 – Master station state table – PrimaryMasterSettingUp	245
Table 140 – Master station state table – PrimaryMasterHoldToken	248
Table 141 – Master station state table – PrimaryMasterSolicitToken	251
Table 142 – Master station state table – PrimaryMasterInviting	254
Table 143 – Master station state table – MasterWaitTD	256
Table 144 – Master station state table – MasterWaitSetup	257
Table 145 – Master station state table – MasterSolicitToken (without Transmission path delay measurement)	258
Table 146 – Master station state table – MasterSolicitToken (with Transmission path delay measurement)	260
Table 147 – Master station state table – MasterHoldToken	263
Table 148 – Master station state table – MasterMeasurement (without Transmission path delay measurement function)	266
Table 149 – Master station state table – MasterMeasurement (with Transmission path delay measurement function)	266
Table 150 – Slave station state table – SlaveDown	266
Table 151 – Slave station state table – SlaveWaitTD	267
Table 152 – Slave station state table – SlaveWaitSetup	268
Table 153 – Slave station state table – SlaveSolicitToken (without Transmission path delay measurement)	269
Table 154 – Slave station state table – SlaveSolicitToken (with Transmission path delay measurement)	271
Table 155 – Slave station state table – SlaveHoldToken	273
Table 156 – Master station channel control functions	276
Table 157 – Slave station channel control functions	277
Table 158 – Master station channel control variables	277

Table 159 – Slave station channel control variables	278
Table 160 – Master station channel control timers	278
Table 161 – Slave station channel control timers	278
Table 162 – Master station parameter dist states	279
Table 163 – Slave station parameter dist states	279
Table 164 – Master station parameter dist state table	279
Table 165 – Slave station parameter dist state table	280
Table 166 – Master station parameter dist functions	281
Table 167 – Slave station parameter dist functions	282
Table 168 – Master station synchronous trigger states.....	282
Table 169 – Slave station synchronous trigger states.....	282
Table 170 – Master station synchronous trigger state table.....	282
Table 171 – Slave station synchronous trigger state table.....	283
Table 172 – Synchronous trigger functions	283
Table 173 – Timer states – Best effort type.....	283
Table 174 – Timer states – Fixed cycle type	283
Table 175 – Timer state table – Best effort type	283
Table 176 – Timer state table – Fixed cycle type	284
Table 177 – Timer variables.....	284
Table 178 – Fixed cycle timer	284
Table 179 – Master station measure transmission states	285
Table 180 – Slave station measure transmission states	285
Table 181 – Master station measure transmission state table	285
Table 182 – Slave station measure transmission state table	287
Table 183 – Master station measure transmission functions.....	287
Table 184 – Slave station measure transmission functions.....	288
Table 185 – Master station measure transmission variables.....	289
Table 186 – Primitives provided by Cyclic Transmission	289
Table 187 – Cyclic Transmission state	290
Table 188 – Cyclic Transmission state table	290
Table 189 – Functions used for Cyclic Transmission.....	290
Table 190 – Variables used in Cyclic Transmission.....	291
Table 191 – Primitives provided by AcyclicTransmission.....	291
Table 192 – Acyclic Transmission state	291
Table 193 – Acyclic Transmission state table.....	291
Table 194 – Functions used for Acyclic Transmission	294
Table 195 – Variables used in Acyclic Transmission	294
Table 196 – Primitives provided by Channel Control	295
Table 197 – Channel Control state.....	295
Table 198 – Channel Control state machine.....	295
Table 199 – Functions used for Channel Control.....	297
Table 200 – Variables used in Channel Control.....	297
Table 201 – Primitives provided by TimeSync Control.....	297

Table 202 – TimeSync Control state	298
Table 203 – TimeSync Control state table.....	298
Table 204 – Functions used for TimeSync Control	298
Table 205 – Variables used in TimeSync Control	299
Table 206 – Primitives provided by IPTrans Control	299
Table 207 – IPTrans Control state in master station.....	299
Table 208 – IPTrans Control state table	299
Table 209 – Functions used for IPTrans Control	300
Table 210 – Variables used in IPTrans Control	300
Table 211 – Handler state	300
Table 212 – DOWN state machine	301
Table 213 – INIT state machine	301
Table 214 – RUN state machine	303
Table 215 – Functions used for Handler.....	304
Table 216 – Variables used for Handler	304
Table 217 – Mapping of type C DMPM service and DL service.....	305
Table 218 – Destination address for each type C PDU	306
Table 219 – Mapping of type F DMPM service and DL service	307
Table 220 – Primitives provided by DMPM	307
Table 221 – Cyclic Data state	307
Table 222 – DMPM state table	308

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, slave 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 slave, 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 checked 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,
    rst                  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)),
    cmRxBBothEndsValidity  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
}

```

<pre> PTP-PDU ::= CHOICE { ptpSync-PDU ptpPdelayReq-PDU ptpPdelayResp-PDU ptpFollowUp -PDU ptpPdelayRespFollowUp-PDU ptpAnnounce -PDU ptpdelayReq-PDU ptpdelayResp-PDU ptpdelayRespFollowUp-PDU } </pre>	<pre> PtpSyn-PDU, PtpPdelayReq-PDU, PtpPdelayResp-PDU, PtpFollowUp-PDU, PtpPdelayRespFollowUp-PDU, PtpAnnounce-PDU PtpdelayReq-PDU, PtpdelayResp-PDU, PtpdelayRespFollowUp-PDU, </pre>
<pre> IP-PDU ::= SEQUENCE { ipHeader updHeader udpData slmp-networkConfigMain-PDU slmp-networkConfigTslt-PDU slmp-notification-PDU slmp-masterConfig-PDU slmp-slaveConfig -PDU slmp-cyclicConfigMain-PDU slmp-cyclicConfigTrnSubPayload-PDU slmp-cyclicConfigRcvSubPayload-PDU slmp-cyclicConfigRcvSrcInfo-PDU } } </pre>	<pre> IPHeader, UDPHeader, CHOICE { Slmp-networkConfigMain-PDU, Slmp-networkConfigTslt-PDU, Slmp-notification-PDU, Slmp-masterConfig-PDU, Slmp-slaveConfig-PDU, Slmp-cyclicConfigMain-PDU, Slmp-cyclicConfigTrnSubPayload-PDU, Slmp-cyclicConfigRcvSubPayload-PDU, Slmp-cyclicConfigRcvSrcInfo-PDU, } </pre>

The header used in each PDU is shown below.

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

CyclicMsSsHeader ::= SEQUENCE { frameType cyclicNo varField vField[0] sa }, vField[1] da }, }, reserved hec }	FrameType, CyclicNo, CHOICE { SEQUENCE { SA }, SEQUENCE { DA }, OCTET STRING (SIZE (2)), HEC }
AcyclicPriorityHeader ::= SEQUENCE { frameType reserved hec }	FrameType, OCTET STRING (SIZE (9)), HEC
AcyclicDetectionHeader ::= SEQUENCE { frameType reserved }	FrameType, OCTET STRING (SIZE (1))
AcyclicTestDataHeader ::= SEQUENCE { frameType dataType persPriority nodeType srcNodeNumber protocolVerType reserved hec }	FrameType, DataType, PersPriority, NodeType, NodeNumber, ProtocolVerType, OCTET STRING (SIZE (1)), HEC
AcyclicDataHeader ::= SEQUENCE { frameType reserved da reserved1 }	FrameType, OCTET STRING (SIZE (1)), DA, OCTET STRING (SIZE (2)),

4.5.2 CyclicM-PDU

```
CyclicM-PDU ::= SEQUENCE {
    cyclicMSHeader
    subPayloadHeader
        da
        commInfo
        txAsynInfo
    },
    subPayloadData
        seqNo
        diagnosisData
        reserved1
        memoryAddress
        applicationData
    }
}
```

```
CyclicMSHeader,
SEQUENCE {
    OCTET STRING (SIZE (2)),
    Unsigned16,
    Unsigned8,

    SEQUENCE {
        Unsigned16,
        Unsigned32,
        OCTET STRING (SIZE (1)),
        Unsigned32,
        OCTET STRING(SIZE(1..1 454))
    }
}
```

4.5.3 CyclicS-PDU

```
CyclicS-PDU ::= SEQUENCE {
    cyclicMSHeader
    subPayloadHeader
        da
        commInfo
        txAsynInfo
    },
    subPayloadData
        seqNo
        diagnosisData
        reserved1
        memoryAddress
        applicationData
    }
}
```

```
CyclicMSHeader,
SEQUENCE {
    OCTET STRING (SIZE (2)),
    Unsigned16,
    Unsigned8,

    SEQUENCE {
        Unsigned16,
        Unsigned32,
        OCTET STRING (SIZE (1)),
        Unsigned32,
        OCTET STRING(SIZE(1..1 454))
    }
}
```

4.5.4 CyclicMs-PDU

```
CyclicMs-PDU ::= SEQUENCE {
    cyclicMsSsHeader
    subPayloadBlock
    eos
}
```

```
CyclicMsSsHeader,
SEQUENCE OF SubPayloadSet1,
EOS
```


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

4.5.5 CyclicSs-PDU

<pre> CyclicSs-PDU::= SEQUENCE { cyclicMsSsHeader subPayloadBlock eos } </pre>	<pre> CyclicMsSsHeader, SEQUENCE OF SubPayloadSet2, EOS </pre>

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

4.5.6 AcyclicPriority-PDU

```

AcyclicPriority-PDU ::= SEQUENCE {
    acyclicPriorityHeader
    acyclicPriorityData
        srcMAC
        mngPriority
        reserved1
        mngMAC
        hopCount
        reqRes
        reserved2
    },
    dcs
}

```

```

AcyclicPriorityHeader,
SEQUENCE {
    OCTET STRING (SIZE (6)),
    Unsigned8,
    OCTET STRING (SIZE (1)),
    OCTET STRING (SIZE (6)),
    Unsigned16,
    Unsigned8,
    OCTET STRING (SIZE (11))
}
Unsigned32

```

4.5.7 AcyclicDetection-PDU

```

AcyclicDetection-PDU ::= SEQUENCE {
    acyclicDetectionHeader
    acyclicDetectionData
        reserved1
        protocolVer
        reserved2
        mngMAC
        previousNodeMAC
        previousNodePort
        optionFlag
        hopCount
        ipAdd
        sendInfo
        reserved3
    }
}

```

```

AcyclicDetectionHeader,
SEQUENCE {
    OCTET STRING (SIZE (2)),
    Unsigned8,
    OCTET STRING (SIZE (1)),
    OCTET STRING (SIZE (6)),
    OCTET STRING (SIZE (6)),
    Unsigned8,
    Unsigned8,
    Unsigned16,
    OCTET STRING (SIZE (25)),
    Unsigned8,
    OCTET STRING (SIZE (10))
}

```

4.5.8 AcyclicDetectionAck-PDU

AcyclicDetectionAck-PDU ::= SEQUENCE {

 acyclicDetectionHeader

 acyclicDetectionAckData

 nodeType

 protocolVer

 ipAddressFourthOctet

 srcMAC

 reserved1

 previousNodeMAC

 previousNodePort

 detectRcvPort

 myPort

 reserved2

 myPortLinkStatus

 myPortFilterStatus

 currentManager

 reserved3

 ipAdd

 performance

 reserved4

 gmPriority

 synctype

 reserved5

 pdelayResTime

 delaySetTime

 announceRelayTime

 reserved6

 deviceVer

 vendorCode

 modelCode

 expansionmodelCode

 deviceType

 memoryAddress

 cyclicSize

 reserved7

 function

 optionInfo

 stationMode

 blank

 }

}

AcyclicDetectionHeader,

SEQUENCE {

 Unsigned8,

 Unsigned8,

 Unsigned16,

 OCTET STRING (SIZE (6)),

 OCTET STRING (SIZE (2)),

 OCTET STRING (SIZE (6)),

 Unsigned8

 Unsigned8,

 Unsigned8,

 OCTET STRING (SIZE (3)),

 Unsigned96,

 Unsigned96,

 OCTET STRING (SIZE (6)),

 OCTET STRING (SIZE (2)),

 OCTET STRING (SIZE (45)),

 Unsigned8,

 OCTET STRING (SIZE (2)),

 Unsigned48,

 Unsigned8,

 OCTET STRING (SIZE (1)),

 Unsigned8,

 Unsigned8,

 Unsigned8,

 OCTET STRING (SIZE (1)),

 OCTET STRING (SIZE (2)),

 Unsigned16,

 OCTET STRING (SIZE (4)),

 OCTET STRING (SIZE (2)),

 Unsigned16,

 Unsigned224,

 Unsigned112,

 OCTET STRING (SIZE (2)),

 Unsigned8,

 Unsigned8,

 Unsigned16,

 OCTET STRING (SIZE (4))

4.5.9 AcyclicTestDataHeader

```

AcyclicTestDataHeader ::= SEQUENCE {
    acyclicTestDataHeader
    acyclicTestDataData
        macAddr
        srcPort
        reserved1
        send Inf
        vField[0]
            reserved2
            hopCount
            ipv4Address
            ipv4Subnet
            reserved3
        },
        vField[1]
            ipv6Subnet
            hopCount
            ipv6Address
        },
    dcs
}

```

AcyclicTestDataHeader,
SEQUENCE {
MACAddress,
PortNumber,
OCTET STRING (SIZE (1)),
SendInfo,
SEQUENCE {
OCTET STRING (SIZE (1)),
HopCount,
Ipv4Address,
Ipv4Subnet,
OCTET STRING (SIZE (8)) ,

SEQUENCE {
Ipv6Subnet,
HopCount,
Ipv6Address,

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 SImpIPAddressSet-PDU

```

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

```

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

4.5.19 SImpNetworkConfigMain-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqNetworkConfigMain and ResNetworkConfigMain.

4.5.20 SImpNetworkConfigTsIt-PDU

```

SImpNetworkConfigTsIt-PDU ::= CHOICE {
    reqNetworkConfigTsIt                       ReqNetworkConfigTsIt,
    resNetworkConfigTsIt                       ResNetworkConfigTsIt
}

```

NOTE Refer to the SLMP Specification for ReqNetworkConfigTsIt and ResNetworkConfigTsIt.

4.5.21 SImpNotification-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqNotification and ResNotification.

4.5.22 SImpMasterConfig-PDU

```

SImpMasterConfig-PDU ::= CHOICE {
    reqMasterConfig          ReqMasterConfig,
    resMasterConfig          ResMasterConfig
}

```

NOTE Refer to the SLMP Specification for ReqMasterConfig and ResMasterConfig.

4.5.23 SImpSlaveConfig-PDU

```

SImpSlaveConfig-PDU ::= CHOICE {
    reqSlaveConfig          ReqSlaveConfig,
    resSlaveConfig          ResSlaveConfig
}

```

NOTE Refer to the SLMP Specification for ReqSlaveConfig and ResSlaveConfig.

4.5.24 SImpCyclicConfigMain-PDU

```

SImpCyclicConfigMain-PDU ::= CHOICE {
    reqCyclicConfigMain     ReqCyclicConfigMain,
    resCyclicConfigMain     ResCyclicConfigMain
}

```

NOTE Refer to the SLMP Specification for ReqCyclicConfigMain and ResCyclicConfigMain.

4.5.25 SImpCyclicConfigTrnSubPayload-PDU

```

SImpCyclicConfigTrnSubPayload-PDU ::= CHOICE {
    reqCyclicConfigTrnSubPayload     ReqCyclicConfigTrnSubPayload,
    resCyclicConfigTrnSubPayload     ResCyclicConfigTrnSubPayload
}

```

NOTE Refer to the SLMP Specification for ReqCyclicConfigTrnSubPayload and ResCyclicConfigTrnSubPayload.

4.5.26 SImpCyclicConfigRcvSubPayload-PDU

```

SImpCyclicConfigRcvSubPayload-PDU ::= CHOICE {
    reqCyclicConfigRcvSubPayload     ReqCyclicConfigRcvSubPayload,
    resCyclicConfigRcvSubPayload     ResCyclicConfigRcvSubPayload
}

```

NOTE Refer to the SLMP Specification for ReqCyclicConfigRcvSubPayload and ResCyclicConfigRcvSubPayload.

4.5.27 SImpCyclicConfigRcvSrcInfo-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqCyclicConfigRcvSrcInfo and ResCyclicConfigRcvSrcInfo.

4.5.28 SImpLinkDevicePrmWrite-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqLinkDevicePrmWrite and ResLinkDevicePrmWrite.

4.5.29 SImpLinkDevicePrmWriteCheckRequest-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqLinkDevicePrmWriteCheckReq and ResLinkDevicePrmWriteCheckReq.

4.5.30 SImpLinkDevicePrmWriteCheckResponse-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqLinkDevicePrmWritecheckResp and ResLinkDevicePrmWritecheckResp.

4.5.31 SImpNMTStateUpload-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqNMTStateUpload and ResNMTStateUpload.

4.5.32 SImpNMTStateDownload-PDU

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

NOTE Refer to the SLMP Specification for ReqNMTStateDownload and ResNMTStateDownload.

4.5.33 SImpReadObject-PDU

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

NOTE Refer to the SLMP Specification for ReqReadObject and ResReadObject.

4.5.34 SImpWriteObject-PDU

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

NOTE Refer to the SLMP Specification for ReqWriteObject and ResWriteObject.

4.5.35 SImpObjectSubIDReadBlock-PDU

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

NOTE Refer to the SLMP Specification for ReqObjectSubIDReadBlock and ResObjectSubIDReadBlock.

4.5.36 SImpObjectSubIDWriteBlock-PDU

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

NOTE Refer to SLMP Specification for ReqObjectSubIDWriteBlock and ResObjectSubIDWriteBlock.

4.5.37 SImpGetODList-PDU

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

NOTE Refer to the SLMP Specification for ReqGetODList and ResGetODList.

4.5.38 SImpGetObjectDescription-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqGetObjectDescription and ResGetObjectDescription.

4.5.39 SImpGetEntryDescription-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqGetEntryDescription and ResGetEntryDescription.

4.5.40 SImpStopOwnStationCyclic-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqStopOwnStationCyclic and ResStopOwnStationCyclic.

4.5.41 SImpStartOwnStationCyclic-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqStartOwnStationCyclic and ResStartOwnStationCyclic.

4.5.42 SImpStopOtherStationsCyclic-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqStopOtherStationsCyclic and ResStopOtherStationsCyclic.

4.5.43 SImpStartOtherStationsCyclic-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqStartOtherStationsCyclic and ResStartOtherStationsCyclic.

4.5.44 SImpAllParameterGet-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqAllParameterGet and ResAllParameterGet.

4.5.45 SImpParameterGet-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqParameterGet and ResParameterGet.

4.5.46 SImpAllParameterSizeGet-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqAllParameterSizeGet and ResAllParameterSizeGet.

4.5.47 SImpParameterSizeGet-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqParameterSizeGet and ResParameterSizeGet.

4.5.48 SImpStationSubIDListGet-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqStationSubIDListGet and ResStationSubIDListGet.

4.5.49 SImpDeviceIdentificationInfoGet-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqDeviceIdentificationInfoGet and ResDeviceIdentificationInfoGet.

4.5.50 SImpDataMonitoring-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqDataMonitoring and ResDataMonitoring.

4.5.51 SImpAllParameterSet-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqAllParameterSet and ResAllParameterSet.

4.5.52 SImpParameterSet-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqParameterSet and ResParameterSet.

4.5.53 SImpParameterVersionCheck-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqParameterVersionCheck and ResParameterVersionCheck.

4.5.54 SImpDeviceIdentificationInfoCompare-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqDeviceIdentificationInfoCompare and ResDeviceIdentificationInfoCompare.

4.5.55 SImpNodeSearch-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqNodeSearch and ResNodeSearch.

4.5.56 SImpIPAddressSet-PDU

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

NOTE Refer to the SLMP Specification for ReqIPAddressSet and ResIPAddressSet.

4.5.57 SImpDeviceInfoCompare-PDU

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

NOTE Refer to the SLMP Specification for ReqDeviceInfoCompare and ResDeviceInfoCompare.

4.5.58 SImpParameterGet-PDU

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

NOTE Refer to the SLMP Specification for ReqParameterGet and ResParameterGet.

4.5.59 SImpParameterSet-PDU

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

NOTE Refer to the SLMP Specification for ReqParameterSet and ResParameterSet.

4.5.60 SImpParameterSetStart-PDU

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

NOTE Refer to the SLMP Specification for ReqParameterSetStart and ResParameterSetStart.

4.5.61 SImpParameterSetEnd-PDU

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

NOTE Refer to the SLMP Specification for ReqParameterSetEnd and ResParameterSetEnd.

4.5.62 SImpVerifyCheckCode-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqVerifyCheckCode and ResVerifyCheckCode.

4.5.63 SImpOutputMapFileNameGet-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqOutputMapFileNameGet and ResOutputMapFileNameGet.

4.5.64 SImpNewFile-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqNewFile and ResNewFile.

4.5.65 SImpParameterSetCancel-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqParameterSetCancel and ResParameterSetCancel.

4.5.66 SImpOpenFile-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqOpenFile and ResOpenFile.

4.5.67 SImpCloseFile-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqCloseFile and ResCloseFile.

4.5.68 SImpReadFile-PDU

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

NOTE Refer to the SLMP Specification for ReqReadFile and ResReadFile.

4.5.69 SImpWriteFile-PDU

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

NOTE Refer to the SLMP Specification for ReqWriteFile and ResWriteFile.

4.5.70 SImpStatusRead-PDU

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

NOTE Refer to the SLMP Specification for ReqStatusRead and ResStatusRead.

4.5.71 SImpCommunicationSettingGet-PDU

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

NOTE Refer to the SLMP Specification for ReqCommunicationSettingGet and ResCommunicationSettingGet.

4.5.72 SImpGetDeviceInfo-PDU

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

NOTE Refer to the SLMP Specification for ReqGetDeviceInfo and ResGetDeviceInfo.

4.5.73 SImpGetBackupListFileName-PDU

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

NOTE Refer to the SLMP Specification for ReqGetBackupListFileName and ResGetBackupListFileName.

4.5.74 SImpStartBackup-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqStartBackup and ResStartBackup.

4.5.75 SImpEndBackup-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqEndBackup and ResEndBackup.

4.5.76 SImpCheckRestoreVersion-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqCheckRestoreVersion and ResCheckRestoreVersion.

4.5.77 SImpStartRestore-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqStartRestore and ResStartRestore.

4.5.78 SImpEndRestore-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqEndRestore and ResEndRestore.

4.5.79 SImpStatusRead2-PDU

```

SImpStatusRead2-PDU ::= CHOICE {
    reqStatusRead2          ReqStatusRead2,
    resStatusRead2          ResStatusRead2
}

```

NOTE Refer to the SLMP Specification for ReqStatusRead2 and ResStatusRead2.

4.5.80 SImpReqSearchNode-PDU

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

NOTE Refer to the SLMP Specification for ReqReqSearchNode and ResReqSearchNode.

4.5.81 SImpGetSearchNodeState-PDU

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

NOTE Refer to the SLMP Specification for ReqGetSearchNodeState and ResGetSearchNodeState.

4.5.82 SImpGetNodeList-PDU

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

NOTE Refer to the SLMP Specification for ReqGetNodeList and ResGetNodeList.

4.5.83 SImpReqSetIPAddress-PDU

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

NOTE Refer to the SLMP Specification for ReqReqSetIPAddress and ResReqSetIPAddress.

4.5.84 SImpSearchPrmControlStation-PDU

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

NOTE Refer to the SLMP Specification for ReqSearchPrmControlStation and ResSearchPrmControlStation.

4.5.85 SImpRequestRestore-PDU

```

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

```

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

4.5.86 SImpCheckPrmDelivery-PDU

```

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

```

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

4.5.87 SImpRsvStationConfigTemporaryRelease-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqRsvStationConfigTemporaryRelease and ResRsvStationConfigTemporaryRelease.

4.5.88 SImpRsvStationConfig-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqRsvStationConfig and ResRsvStationConfig.

4.5.89 SImpGetEventNum-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqGetEventNum and ResGetEventNum.

4.5.90 SImpGetEventHistory-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqGetEventHistory and ResGetEventHistory.

4.5.91 SImpClearEventHistory-PDU

```

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

```

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

4.5.92 SImpClockOffsetDataSend-PDU

```

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

```

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

4.5.93 SImpSetWatchdogCounterInfo-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqSetWatchdogCounterInfo and ResSetWatchdogCounterInfo.

4.5.94 SImpWatchdogCounterOffsetConfig-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqWatchdogCounterOffsetConfig and ResWatchdogCounterOffsetConfig.

4.5.95 SImpRemoteReset-PDU

```

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

```

NOTE Refer to the SLMP Specification for ReqRemoteReset and ResRemoteReset.

4.5.96 SImpGetCommunicationSet-PDU

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

NOTE Refer to the SLMP Specification for ReqGetCommunicationSet and ResGetCommunicationSet.

4.5.97 SImpGetStationSubIDList-PDU

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

NOTE Refer to the SLMP Specification for ReqGetStationSubIDList and ResGetStationSubIDList.

4.5.98 SImpGetDeviceInfo-PDU

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

NOTE Refer to the SLMP Specification for ReqGetDeviceInfo and ResGetDeviceInfo.

4.5.99 SImpStartBackup-PDU

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

NOTE Refer to the SLMP Specification for ReqStartBackup and ResStartBackup.

4.5.100 SImpEndBackup-PDU

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

NOTE Refer to the SLMP Specification for ReqEndBackup and ResEndBackup.

4.5.101 SImpRequestBackup-PDU

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

NOTE Refer to the SLMP Specification for ReqRequestBackup and ResRequestBackup.

4.5.102 SImpGetBackupPrm-PDU

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

NOTE Refer to the SLMP Specification for ReqGetBackupPrm and ResGetBackupPrm.

4.5.103 SImpCheckRestore-PDU

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

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

4.5.104 SImpStartRestore-PDU

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

NOTE Refer to the SLMP Specification for ReqStartRestore and ResStartRestore.

4.5.105 SImpEndRestore-PDU

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

NOTE Refer to the SLMP Specification for ReqEndRestore and ResEndRestore.

4.5.106 SImpSetBackupPrm-PDU

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

NOTE Refer to the SLMP Specification for ReqSetBackupPrm and ResSetBackupPrm.

4.5.107 SImpLinkupSpeed-PDU

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

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

4.5.108 SImpNodeIndication-PDU

```

SImpNodeIndication-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 commandParamId. 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

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

Table 21 – Details of param area

Field name	Size (octets)	Description
Parameter Availability	2	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.

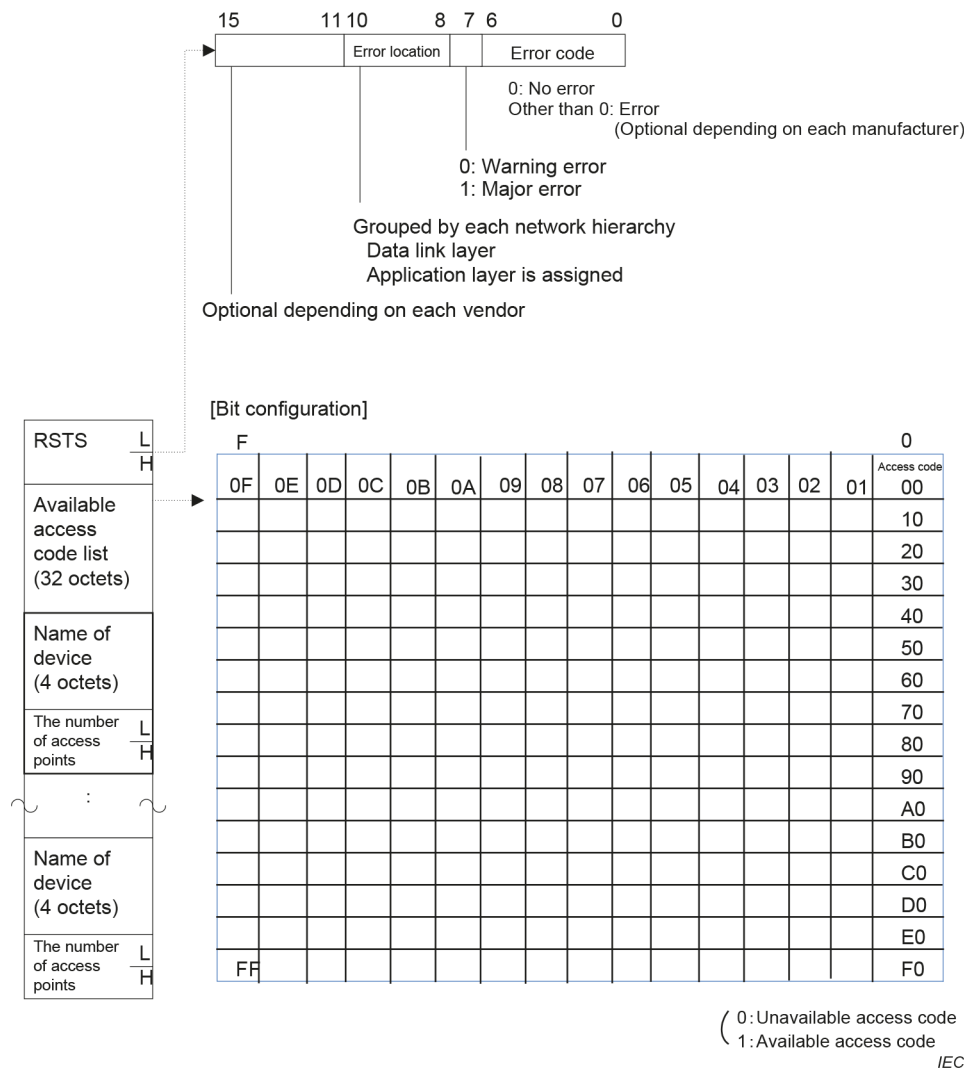


Figure 2 – Structure for memory access information retrieve response

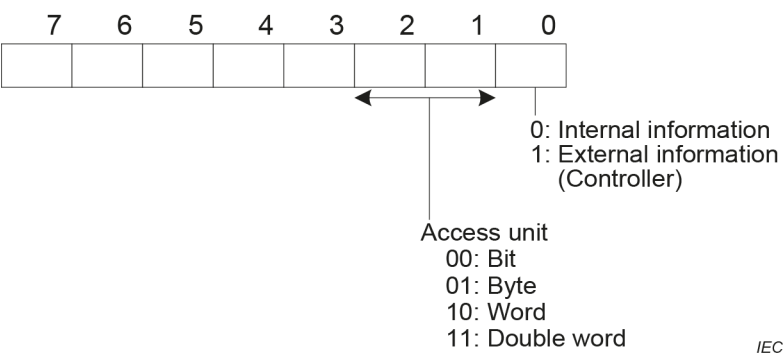


Figure 3 – Attribute definitions

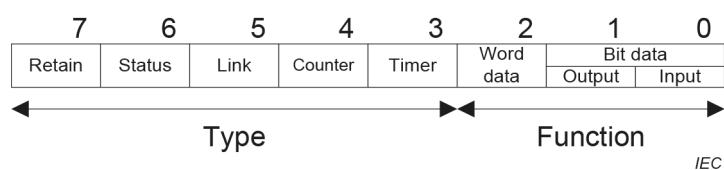


Figure 4 – Access code definitions

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

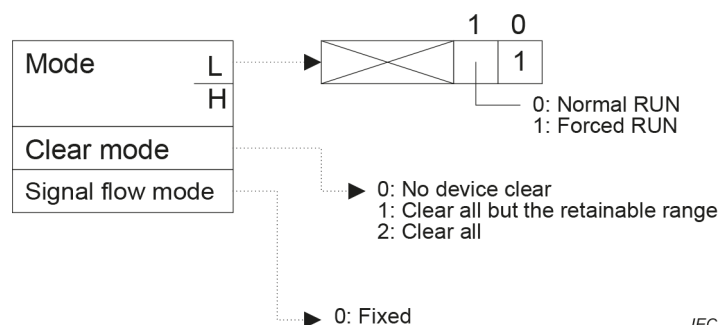
Memory contents		Access code
Buffer	Standard buffer	0x00
Status buffer	Intelligent device node Auto-refresh buffer	0x40
Link buffer	Random access buffer	0x20
Link Device	Link input	0x21
	Link output	0x22
	Link register	0x24
	Link special relay	0x63
	Link special register	0x64
NOTE Specific fields and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

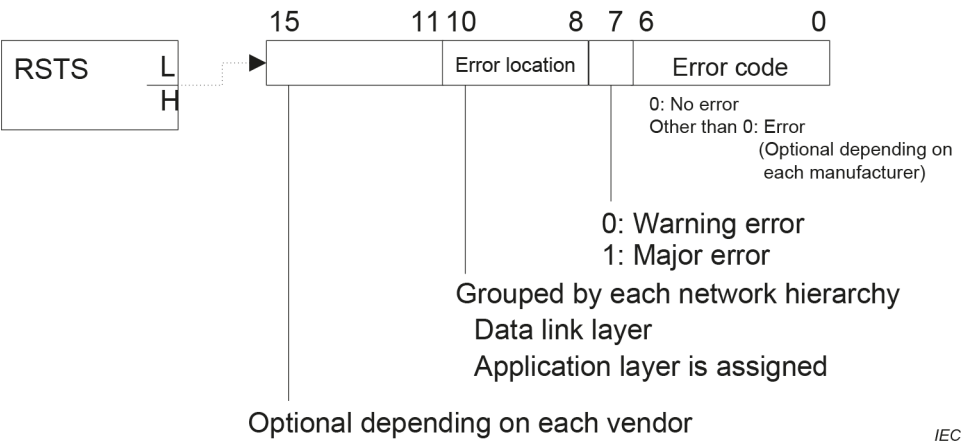


Figure 6 – Structure for RUN response

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.

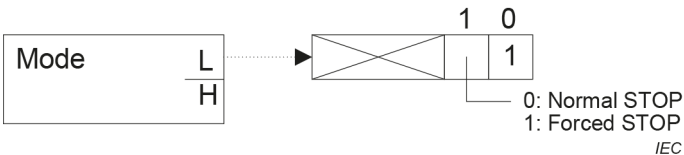


Figure 7 – Structure for STOP request

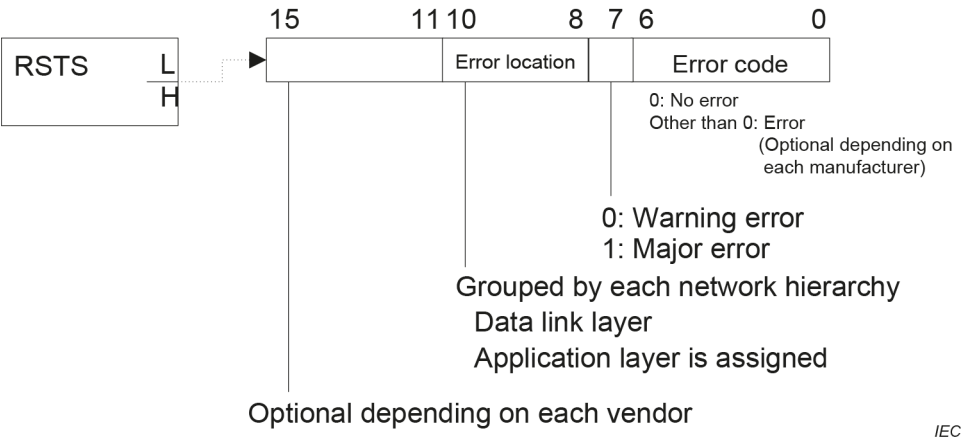


Figure 8 – Structure for STOP response

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.

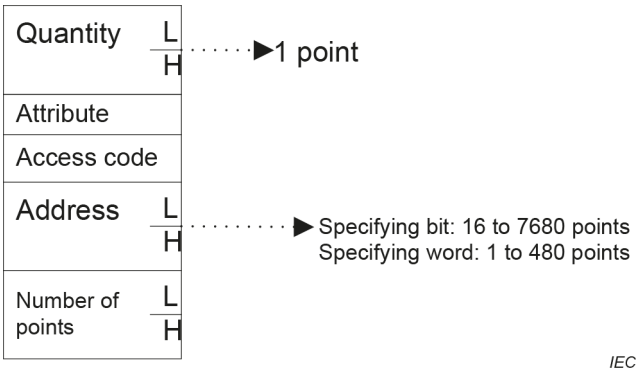


Figure 9 – Structure for batch memory read request

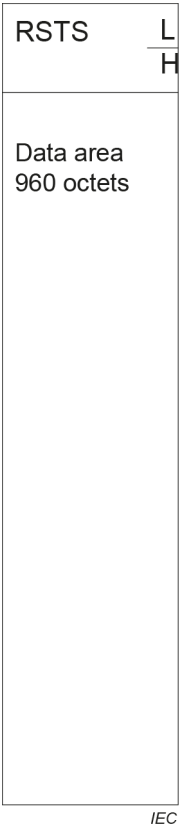


Figure 10 – Structure for batch memory read response

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

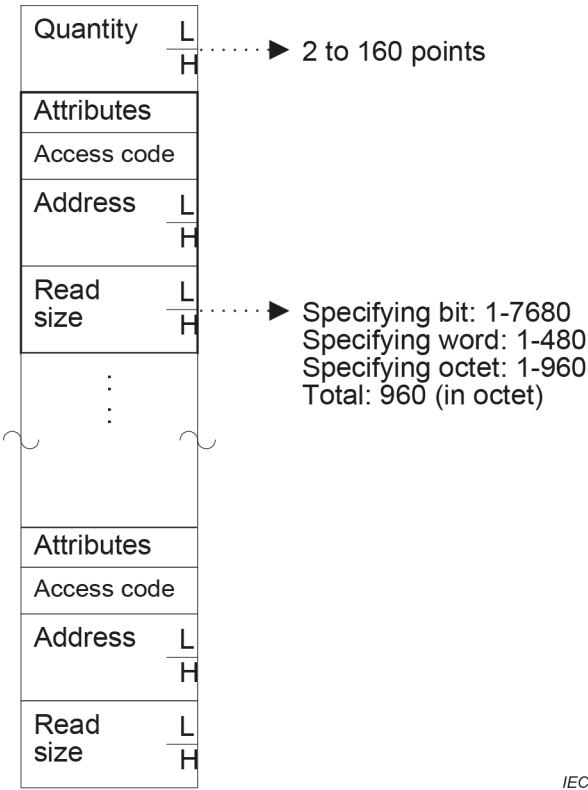


Figure 11 – Structure for random memory read request

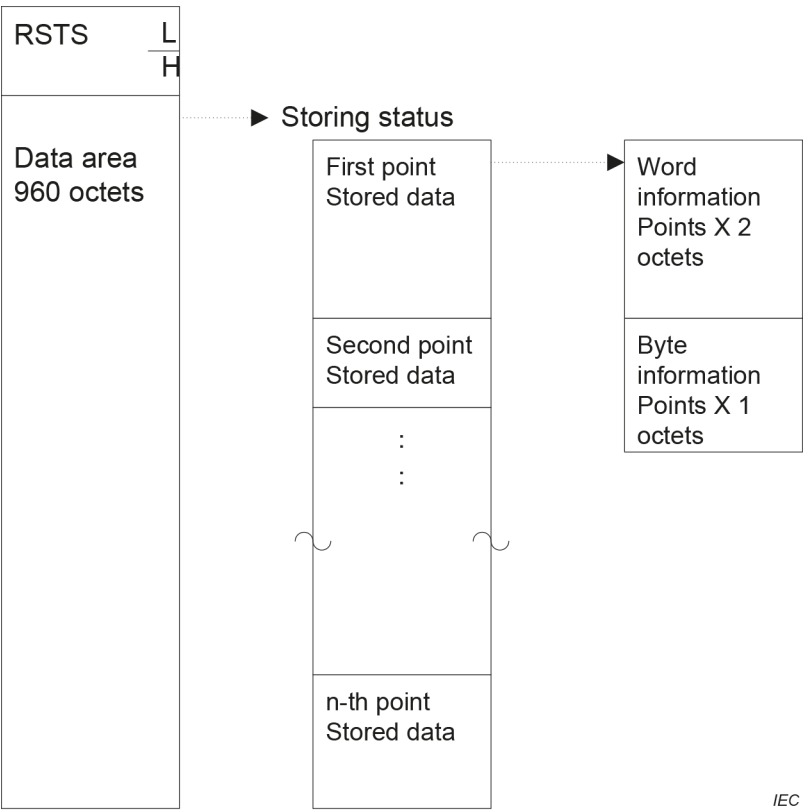


Figure 12 – Structure for random memory read response

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.

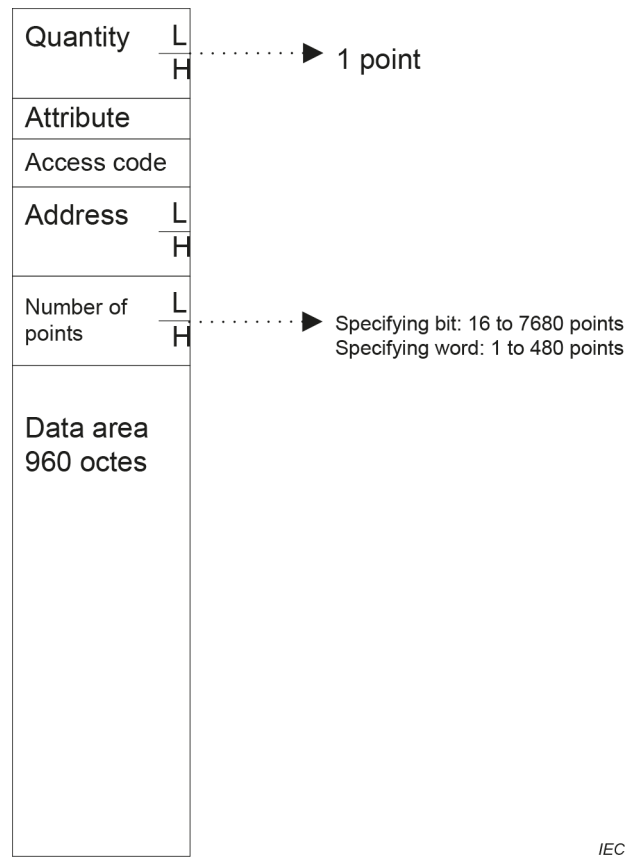


Figure 13 – Structure for batch memory write request

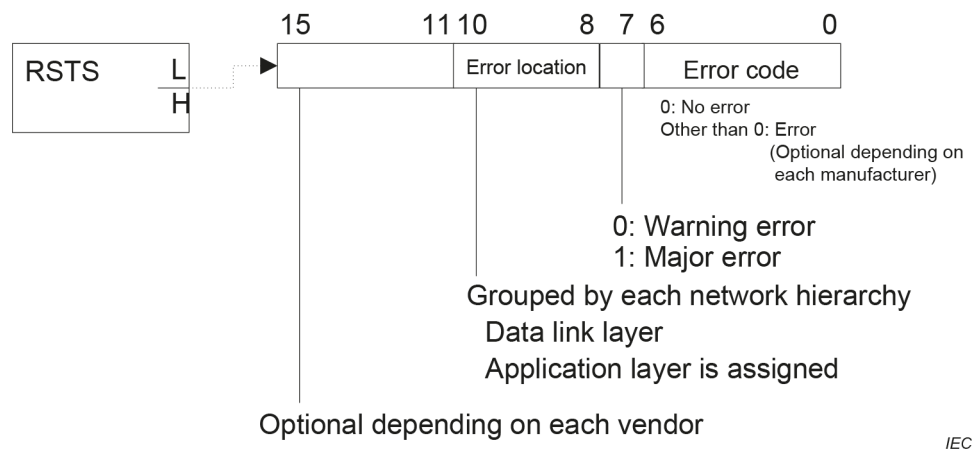


Figure 14 – Structure for batch memory write response

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

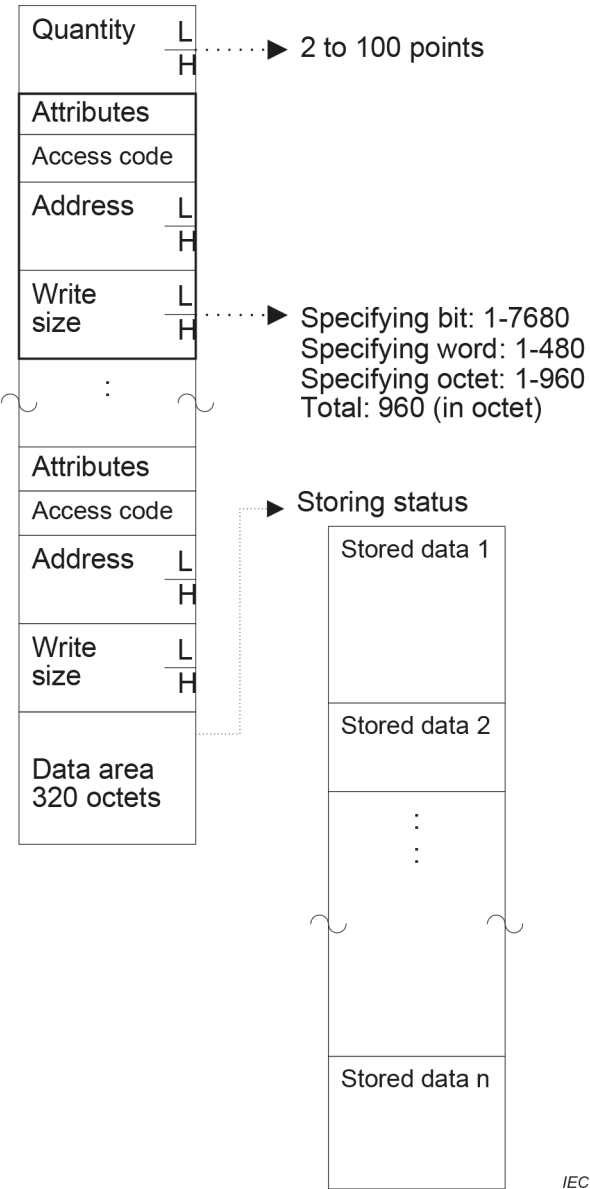


Figure 15 – Structure for random memory write request

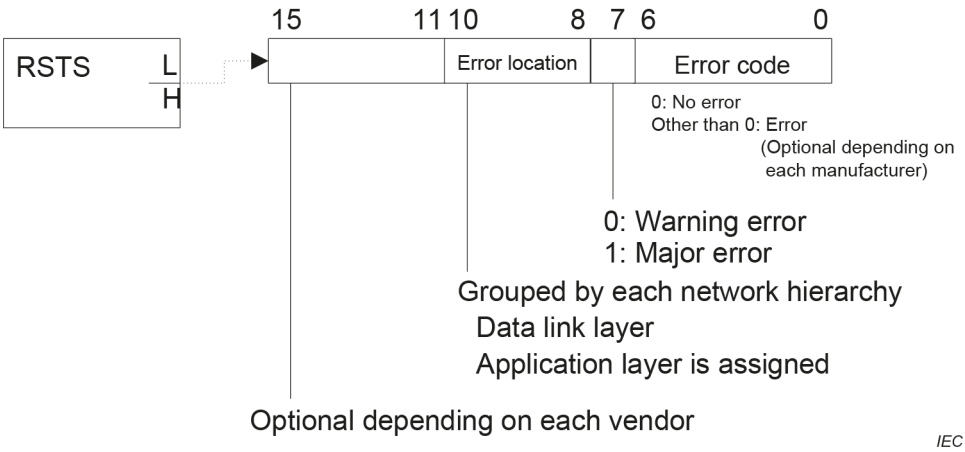


Figure 16 – Structure for random memory write response

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..0xFFFFFFFF. 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 μ s. 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 **rwSize**

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 $\text{tokenHopCounter} \geq \text{traAvailHopCounter}$ and $\text{traAllows} > 0$.

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 transmissions 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 rst

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

Bits 15..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 μ s.

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 cmRxOffset. 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 μ s, 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 Plug&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 0xFFFFF.

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 $(RX/16+RY/16+RWw+RW_r)/8$ 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 μ s, 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 $2^{\text{logMessageInterval}}$ [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 2^4 (–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.

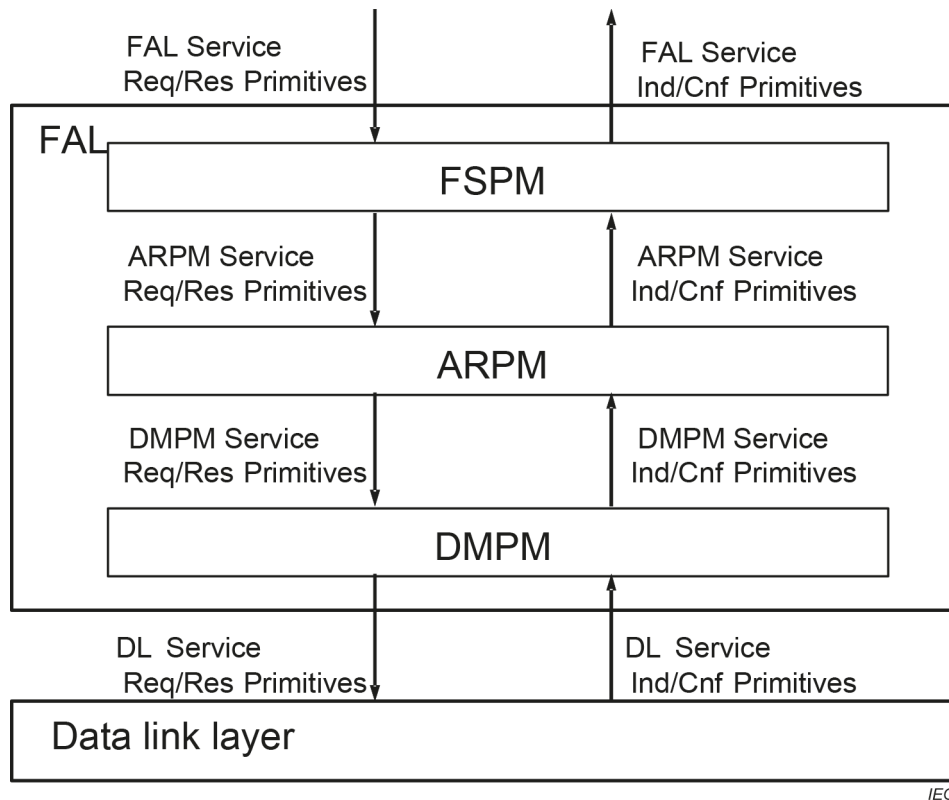


Figure 17 – Relationships between protocol machines

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.

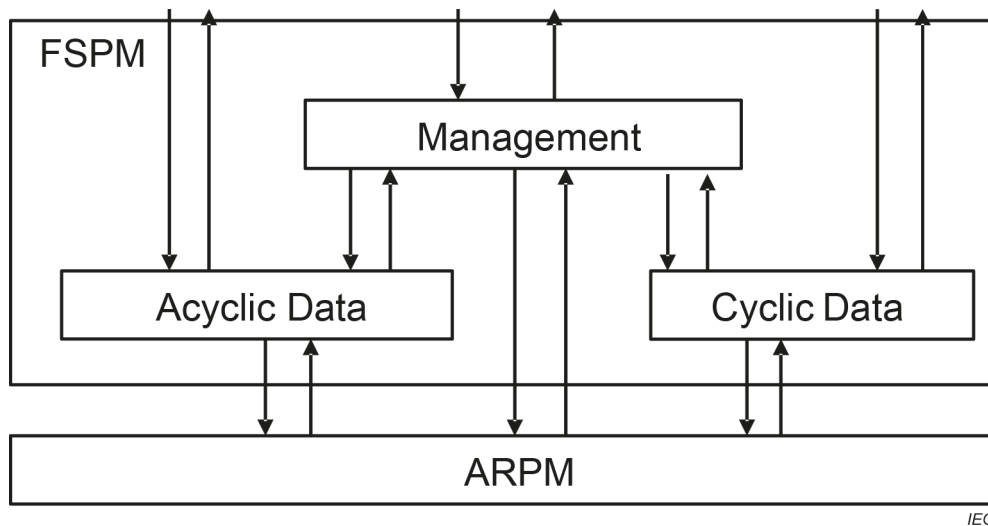


Figure 18 – Structure of FSPM C

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.

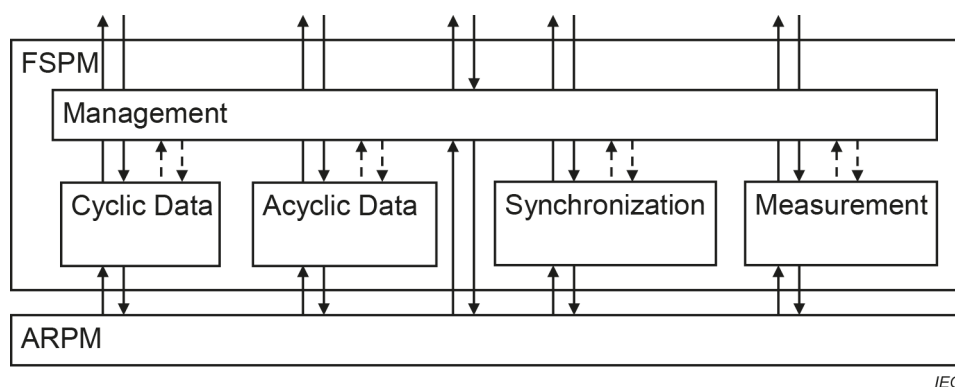
Table 85 – Management state table

#	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.



IEC

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
Synchronous Trigger.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	Write 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	Synchronous Trigger Internal.ind => Synchronous 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.

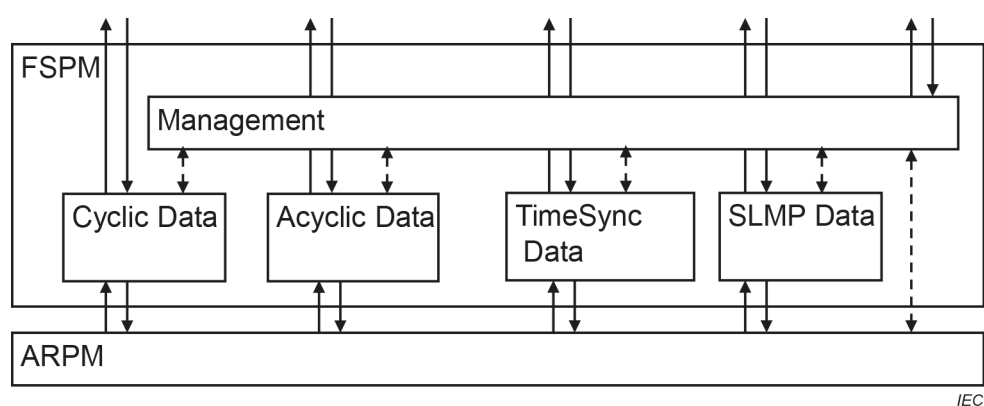


Figure 20 – Structure of FSPM T

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

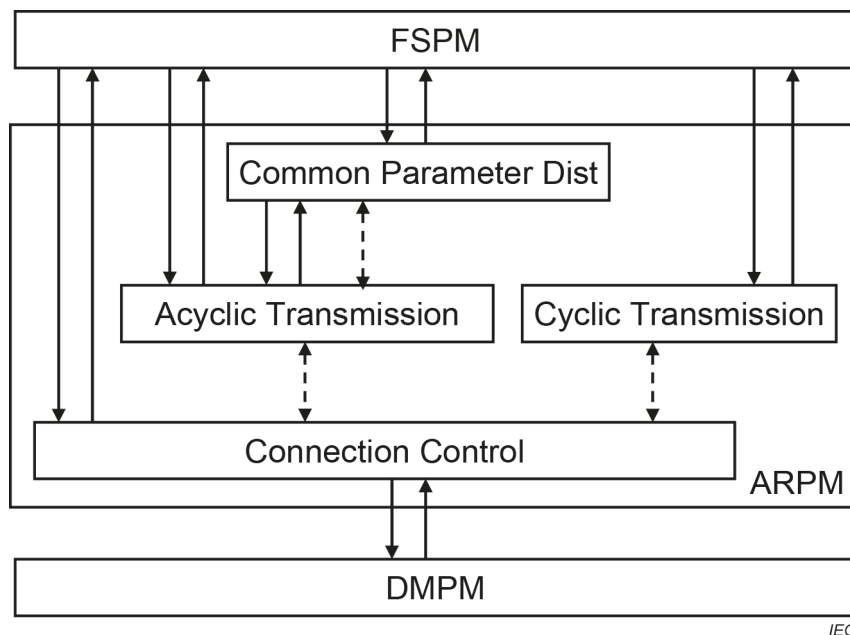
#	Current state	Event /Condition => Action	Next state
1	ACTIVE	SLMP Data.req(SImpData) => SLMPSend.req(SImpData)	ACTIVE
2	ACTIVE	SLMPSend.ind(SImpData) / slmpMTHHeader.fType == 0x6800 => SLMPData.ind(SImpData)	ACTIVE
3	ACTIVE	SLMPSend.ind(SImpData) / slmpMTHHeader.fType == 0xE800 => SLMPData.cnf(SImpData)	ACTIVE
4	ACTIVE	SLMPData.rsp(SImpData) => SLMPSend.req(SImpData)	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.



IEC

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 ACParmSend.req service to Acyclic transmission. Acyclic transmission issues an ACSend.ind service to FSPM. Acyclic transmission issues an ACParmSend.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, Transient1-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 CyclicDataOut2-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

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

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

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

#	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

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

#	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

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

#	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

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

#	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

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

#	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

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

#	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

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

#	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

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

#	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) => CPUpeate(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) => CPUpeate(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) => CPUpdate(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) => CPUpdate(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) => CPUpdate(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(RecvNodeList) == TRUE && CPComp(CP) == 0 => CPIDUpdate(CPID)	SlaveEnd

#	Current state	Event /condition => action	Next state
26	SlaveInit	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) != 0 => CPIDUpdate(CPID); CPCheck(CP); CPCheckResult = CPChecking	ReceivedCPChecking
27	SlaveInit	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) == 0 => CPWCR.req(CheckResult = CPNotReceived)	SlaveInit
28	SlaveInit	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) != 0 => CPWCR.req(CheckResult = CPCheckResult)	SlaveEnd
29	SlaveInit	CPD Set.req(NodeId, Param) => CPUpdate(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(RecvNodeList) == TRUE && CPComp(CP) == 0 => CPIDUpdate(CPID)	SlaveEnd
32	ReceivedCPChecking	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) != 0 => CPIDUpdate(CPID); CPChanged = TRUE	ReceivedCPChecking
33	ReceivedCPChecking	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) == 0 => CPWCR.req(CheckResult = CPNotReceived); CPChanged = TRUE	ReceivedCPChecking
34	ReceivedCPChecking	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecvNodeList) == 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	CPWCRReceiveWaiting
38	CPWCRReceiveWaiting	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
CPUUpdate(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.
IsValidCP()	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.

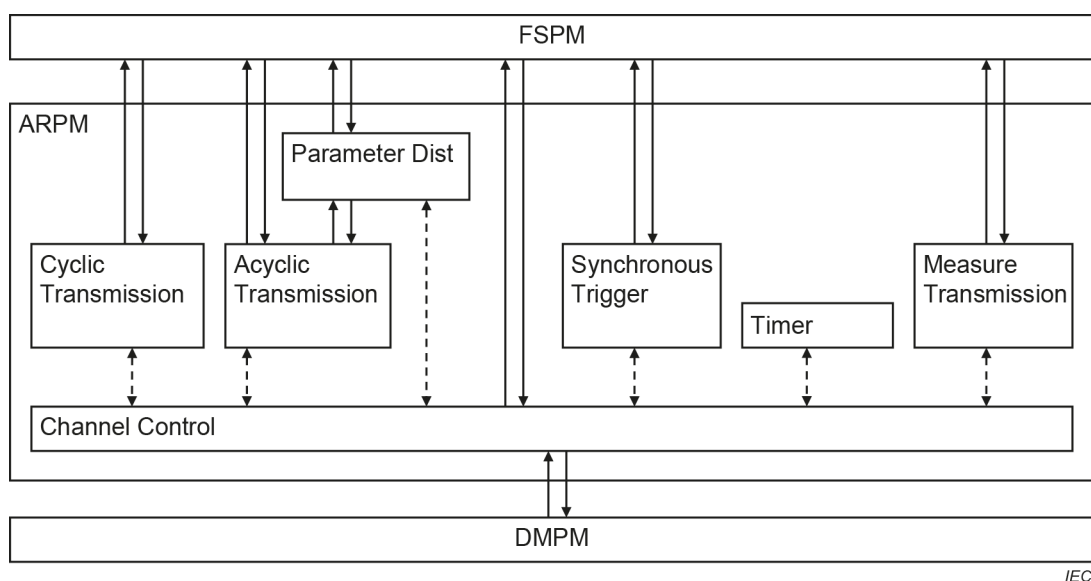


Figure 22 – Structure of ARPM F

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.
Equeue(pdus)	Queues pdus given as arguments. Queues in the order of the pdu seqNumber when multiple pdus are given.
EqueueND(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 RWrRWw 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	MasterArbitration	/ TR2 == TRUE => SynchronizationMaster=FALSE TR2 = FALSE; RvLastArbTimer startup; ChannelGroup = NULL; SendArbTimer startup; pdu = CreatePersuasion(); Send.req (port, BROADCAST, pdu) to all ports	MasterArbitration
2	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; RvLastArbTimer restart	MasterArbitration
3	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than report; SendArbTimer stop; RvLastArbTimer stop; TR8 = TRUE	MasterWaitTD
4	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitration
5	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitration
6	MasterArbitration	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterArbitration

#	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	MasterArbitration
5	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) == Low => Nodes++	PrimaryMaster ScatterTD
6	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) != Low => TDAckTimer stop; TR2 = TRUE	MasterArbitration
7	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster ScatterTD
8	PrimaryMaster 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

#	Current state	Event /condition => action	Next state
13	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	PrimaryMaster ScatterTD
14	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	PrimaryMaster ScatterTD
15	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster ScatterTD
16	PrimaryMaster ScatterTD	TDAckTimer time-out / Nodes > 0 => TDAckTimer stop; TR4 = TRUE	PrimaryMaster SettingUp
17	PrimaryMaster ScatterTD	TDAckTimer time-out / Nodes == 0 => TDAckTimer stop; TR3 = TRUE	PrimaryMaster ScatterTD
18	PrimaryMaster ScatterTD	/ TR3 == TRUE && RingCheck == TRUE => TR3 = FALSE; Nodes=0;ANodes=0; Invalidate port 2; pdu = CreateTestData(); Send.req (port, BROADCAST, pdu) to port 1; ChannelGroup = MCAST(MyAddr); TDAckTimer startup	PrimaryMaster ScatterTD
19	PrimaryMaster ScatterTD	TDAckTimer time-out / Nodes > 0 && RingCheck == TRUE && IsValidPort(port 2) != TRUE => Invalidate port 1; Validate port 2; pdu = CreateTestData(); Send.req (port, BROADCAST, pdu) to port 2; ChannelGroup = MCAST(MyAddr); TDAckTimer startup	PrimaryMaster ScatterTD
20	PrimaryMaster ScatterTD	TDAckTimer time-out / Nodes > 0 && RingCheck == TRUE && IsValidPort(port 1) != TRUE => TDAckTimer stop; TR4 = TRUE	PrimaryMaster SettingUp

Table 139 – Master station state table – PrimaryMasterSettingUp

#	Current state	Event /condition => action	Next state
1	PrimaryMasterSettingUp	/ 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	PrimaryMasterSettingUp
2	PrimaryMasterSettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; SAckTimer stop; TR2 = TRUE	MasterArbitration
3	PrimaryMasterSettingUp	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	PrimaryMasterSettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => SAckTimer stop; TR2 = TRUE	MasterArbitration
5	PrimaryMasterSettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterSettingUp
6	PrimaryMasterSettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterSettingUp
7	PrimaryMasterSettingUp	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	PrimaryMasterSettingUp
8	PrimaryMasterSettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / RingChecked == FALSE && DA == MyAddr && SA != MyAddr && ftype == 0x14 && SARcvd+1== SetupNodes => SAckTimer stop; TR5 = TRUE	PrimaryMasterHoldToken
9	PrimaryMasterSettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterSettingUp

#	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() == TRUE then Invalidate port 2; CreateTokenRoute(); pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) to all ports; SAckTimer startup	PrimaryMaster SettingUp

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

Table 140 – Master station state table – PrimaryMasterHoldToken

#	Current state	Event /condition => action	Next state
1	PrimaryMasterHoldToken	/ TR5 == TRUE => TR5 = FALSE; SetInvitationFlag(); SetMeasureFlag(); CTicket = FALSE; sends = 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);	PrimaryMasterHoldToken
2	PrimaryMasterHoldToken	/ PMChangeFlag == ON => TR2 = TRUE	MasterArbitration
3	PrimaryMasterHoldToken	/ PH == 1 && InvitationFlag == OFF && MyStatusSendTimingFlag == ON => InvitationFlag == NULL; MyStatusSendTimingFlag = OFF pdu = CreateMyStatus(); Send.req (port, ChannelGroup, pdu) to all ports; MyStatusSend(pdu); Enqueue(pdu); CTicket = TRUE; SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMasterHoldToken
4	PrimaryMasterHoldToken	/ PH == 1 && InvitationFlag == ON => InvitationFlag == NULL; TR7 = TRUE	PrimaryMasterInviting
5	PrimaryMasterHoldToken	/ PH == 10 && CTicket == FALSE => SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMasterHoldToken
6	PrimaryMasterHoldToken	/ PH == 11 => ACTicketND = TRUE; SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMasterHoldToken
7	PrimaryMasterHoldToken	/ PH == 12 => ACTicket = TRUE; SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMasterHoldToken
8	PrimaryMasterHoldToken	/ PH == 20 && ((ACTicketND == FALSE) && (ACTicket == FALSE)) => SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMasterInviting

#	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

#	Current state	Event /condition => action	Next state
18	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster HoldToken
19	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster HoldToken
20	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	PrimaryMaster HoldToken
21	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu)	PrimaryMaster HoldToken
22	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	PrimaryMaster HoldToken
23	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	PrimaryMaster HoldToken
24	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster HoldToken
25	PrimaryMaster HoldToken	SendCyclic(pdu) => Send.req (sport, ChannelGroup, pdu) to all ports; Enque(pdu)	PrimaryMaster HoldToken
26	PrimaryMaster HoldToken	/ MeasureFlag == ON => MeasureFlag = NULL TR12 = TRUE	MasterMeasur ement

Table 141 – Master station state table – PrimaryMasterSolicitToken

#	Current state	Event /condition => action	Next state
1	PrimaryMasterSolicitToken	/ TR6 == TRUE => TR6 = FALSE; NetWatchTimer startup; SetNewTokenFlag()	PrimaryMasterSolicitToken
2	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; NetWatchTimer stop; TR2 = TRUE	MasterArbitration
3	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; NetWatchTimer stop; TR8 = TRUE	MasterWaitTD
4	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => NetWatchTimer stop; TR2 = TRUE	MasterArbitration
5	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterSolicitToken
6	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterSolicitToken
7	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterSolicitToken
8	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) to all ports other than rport; NetWatchTimer stop; AddReceivedToken(pdu.tseq_no); TR5 = TRUE	PrimaryMasterHoldToken
9	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== TRUE => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterSolicitToken

#	Current state	Event /condition => action	Next state
10	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport; NetWatchTimer restart	PrimaryMaster SolicitToken
11	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
12	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
13	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
14	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
15	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMaster SolicitToken
16	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() != MyAddr && NewTokenFlag == ON && RingCheck == FALSE => NetWatchTimer stop; pdu = CreateToken(); Send.req (sport, LostTokenDst(), pdu) to all ports; NetWatchTimer restart	PrimaryMaster SolicitToken
17	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() != MyAddr && NewTokenFlag == OFF && RingCheck == FALSE => NetWatchTimer stop; pdu = CreatePersuasion(); Send.req (sport, BROADCAST, pdu) to all ports; TR3 = TRUE	PrimaryMaster ScatterTD

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

Table 142 – Master station state table – PrimaryMasterInviting

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

#	Current state	Event /condition => action	Next state
11	PrimaryMaster Inviting	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);	PrimaryMasterInvit ing
12	PrimaryMaster Inviting	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);	PrimaryMasterInvit ing
13	PrimaryMaster Inviting	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);	PrimaryMasterInvit ing
14	PrimaryMaster Inviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu);	PrimaryMasterInvit ing
15	PrimaryMaster Inviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	PrimaryMasterInvit ing
16	PrimaryMaster Inviting	TDackTimer time-out / ANodes > 0 => TDackTimer stop; TR4 = TRUE	PrimaryMasterSett ingUp
17	PrimaryMaster Inviting	TDackTimer time-out / ANodes == 0 => TDackTimer stop; TR5 = TRUE	PrimaryMasterHol dToken
18	PrimaryMaster Inviting	/ TR7 == TRUE && RingCheck == TRUE && SlavePortStateChanged() == TRUE => TR7 = FALSE; pdu = CreateSetup (Validate port with port status change); Send.req (sport, port with port status change, pdu) to all ports; SlavePortValidating = TRUE	PrimaryMasterInvit ing
19	PrimaryMaster Inviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x14 && RingCheck == TRUE && SlavePortValidating = TRUE => TDackTimer startup; ANodes = 0; pdu = CreateTestData(); Send.req (sport, BROADCAST, pdu) to all ports	PrimaryMasterInvit ing

Table 143 – Master station state table – MasterWaitTD

#	Current state	Event /condition => action	Next state
1	MasterWaitTD	/ TR8 == TRUE => TR8 = FALSE; TDWaitTimer startup; ChannelGroup = NULL	MasterWaitTD
2	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) to all ports other than rport; TDWaitTimer restart	MasterWaitTD
3	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => TDAckTimer stop; pdu = CreateTestDataAck();Send.req(rport, SA, pdu); pdu = CreateTestData(); Send.req (rport, DA, pdu) to all ports other than rport; TR9 = TRUE	MasterWaitSetup
4	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD
5	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD
6	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD
7	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport; TDWaitTimer restart	MasterWaitTD
8	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) ftype == 0x20 ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD
9	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACRReceived(pdu);	MasterWaitTD
10	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitTD

#	Current state	Event /condition => action	Next state
11	MasterWaitTD	TDWaitTimer time-out => TDWaitTimer stop; TR3 = TRUE	PrimaryMasterScatterTD

Table 144 – Master station state table – MasterWaitSetup

#	Current state	Event /condition => action	Next state
1	MasterWaitSetup	/ TR9 == TRUE => TR9 = FALSE; SWaitTimer startup	MasterWaitSetup
2	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) to all ports other than rport; SWaitTimer stop; TR2 = TRUE	MasterArbitration
3	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) to all ports other than rport; SWaitTimer stop; TR8 = TRUE	MasterWaitTD
4	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestDataAck(); Send.req(rport, SA, pdu); pdu = CreateTestData(); Send.req (rport, DA, pdu) to all ports other than rport; SWaitTimer restart	MasterWaitSetup
5	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitSetup
6	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => SWaitTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, pdu); TR10 = TRUE	MasterSolicitToken
7	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterWaitSetup

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

**Table 145 – Master station state table – MasterSolicitToken
(without Transmission path delay measurement)**

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

#	Current state	Event /condition => action	Next state
4	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (rport, DA, pdu) to all ports other than rport	MasterSolicitToken
5	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
6	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, pdu); LeaveTimer startup	MasterSolicitToken
7	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
8	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
9	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no); TR11 = TRUE	MasterHoldToken
10	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== TRUE => Send.req (sport, DA, pdu) to all ports other than rport;	MasterSolicitToken
11	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
12	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	MasterSolicitToken

#	Current state	Event /condition => action	Next state
13	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu)	MasterSolicitToken
14	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	MasterSolicitToken
15	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	MasterSolicitToken
16	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicitToken
17	MasterSolicitToken	LeaveTimer time-out => LeaveTimer stop; TR8 = TRUE	MasterWaitTD

**Table 146 – Master station state table – MasterSolicitToken
(with Transmission path delay measurement)**

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

#	Current state	Event /condition => action	Next state
5	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicit Token
6	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, pdu); LeaveTimer startup	MasterSolicit Token
7	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicit Token
8	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicit Token
9	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no) == FALSE => Send.req (sport, DA, pdu) to all ports other than rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no); TR11 = TRUE	MasterHoldTo ken
10	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no) == TRUE => Send.req (sport, DA, pdu) to all ports other than rport;	MasterSolicit Token
11	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicit Token
12	MasterSolicit Token	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)	MasterSolicit Token
13	MasterSolicit Token	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)	MasterSolicit Token

#	Current state	Event /condition => action	Next state
14	MasterSolicit Token	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)	MasterSolicit Token
15	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	MasterSolicit Token
16	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterSolicit Token
17	MasterSolicit Token	LeaveTimer time-out => LeaveTimer stop; TR8 = TRUE	MasterWaitT D
18	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x40 => MeasureReceived(rport, DA, SA, pdu)	MasterSolicit Token
19	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x41 => MeasureAckReceived(rport, DA, SA, pdu)	MasterSolicit Token
20	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x42 => OffsetReceived(rport, DA, SA, pdu)	MasterSolicit Token
21	MasterSolicit Token	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x43 => UpdateReceived(rport, DA, SA, pdu)	MasterSolicit Token

Table 147 – Master station state table – MasterHoldToken

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

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

#	Current state	Event /condition => action	Next state
18	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	MasterHoldToken
19	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => CStopSending(); ACStopSending() Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu); TR10 = TRUE	MasterSolicitToken
20	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x20 => CStopSending(); ACStopSending() Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu); TR10 = TRUE	MasterSolicitToken
21	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => CStopSending(); ACStopSending() Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu); TR10 = TRUE	MasterSolicitToken
22	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => CStopSending(); ACStopSending() ACReceived(pdu); TR10 = TRUE	MasterSolicitToken
23	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	MasterHoldToken
24	MasterHoldToken	SendCyclic(pdu) => Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu)	MasterHoldToken
25	MasterHoldToken	SendACyclic(pdu) => Send.req (sport, ChannelGroup, pdu) to all ports; Enque(pdu)	MasterHoldToken

**Table 148 – Master station state table – MasterMeasurement
(without Transmission path delay measurement function)**

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

**Table 149 – Master station state table – MasterMeasurement
(with Transmission path delay measurement function)**

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

Table 150 – Slave station state table – SlaveDown

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

Table 151 – Slave station state table – SlaveWaitTD

#	Current state	Event /condition => action	Next state
1	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
2	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestDataAck(); Send.req(rport, SA, pdu); pdu = CreateTestData(); Send.req (sport, DA, pdu) to all ports other than rport;	SlaveWaitSetup
3	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
4	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
5	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
6	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
7	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) ftype == 0x20 ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD
8	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	SlaveWaitTD
9	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitTD

Table 152 – Slave station state table – SlaveWaitSetup

#	Current state	Event /condition => action	Next state
1	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) to all ports other than rport; ChannelGroup = NULL	SlaveWaitTD
2	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestDataAck(); Send.req(rport, SA, pdu); pdu = CreateTestData(); Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup
3	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup
4	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU); TR3 = TRUE	SlaveSolicitToken
5	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup
6	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup
7	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup
8	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) ftype == 0x20 ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	SlaveWaitSetup
9	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	SlaveWaitSetup
10	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveWaitSetup

**Table 153 – Slave station state table – SlaveSolicitToken
(without Transmission path delay measurement)**

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

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

**Table 154 – Slave station state table – SlaveSolicitToken
(with Transmission path delay measurement)**

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

#	Current state	Event /condition => action	Next state
10	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
11	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) to all ports other than rport; CReceived(pdu)	SlaveSolicitToken
12	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && ftype ≥ 0x20 => Send.req (sport, DA, pdu) to all ports other than rport; MyStatusReceived(pdu)	SlaveSolicitToken
13	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) to all ports other than rport; ACReceived(pdu)	SlaveSolicitToken
14	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	SlaveSolicitToken
15	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveSolicitToken
16	SlaveSolicitToken	LeaveTimer time-out => LeaveTimer stop; ChannelGroup = NULL	SlaveWaitTD
17	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x40 => MeasureRecrived(rport, DA, SA, pdu)	SlaveSolicitToken
18	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x41 => MeasureAckReceived(rport, DA, SA, pdu)	SlaveSolicitToken
19	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x42 => OffsetReceived(rport, DA, SA, pdu)	SlaveSolicitToken
20	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x43 => UpdateReceived(rport, DA, SA, pdu)	SlaveSolicitToken

Table 155 – Slave station state table – SlaveHoldToken

#	Current state	Event /condition => action	Next state
1	SlaveHoldToken	/ TR4 == TRUE => TR4 = FALSE; SetNextPhase(SlaveHoldToken,0);	SlaveHoldToken
2	SlaveHoldToken	/ PH == 1 => pdu = CreateMyStatus(); Send.req(port,ChannelGroup,pdu) to all ports; CTicket = TRUE; sents = 0; TokenHopCounter = token-PDU.tokenHopCounter; TraAvailHopCounter = token-PDU.traAvailHopCounter; TraLastHopCounter = token-PDU.traLastHopCounter; TraAllows = token-PDU.traAllows; TraAllowsND = GetTraAllowsND(); SetNextPhase(SlaveHoldToken,0);	SlaveHoldToken
3	SlaveHoldToken	/ PH == 10 && CTicket == FALSE => SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
4	SlaveHoldToken	/ PH == 11 => ACTicketND = TRUE; SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
5	SlaveHoldToken	/ PH == 12 => ACTicket = TRUE; SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
6	SlaveHoldToken	/ PH == 20 && ((ACTicketND == FALSE) && (ACTicket == FALSE)) => SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
7	SlaveHoldToken	/ PH == 3 && sents < Multiple Transmit => sents = sents + 1; PduNum = QueLen(); while (PduNum > 0) pdu = Deque(); Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu); PduNum = PduNum -1	SlaveHoldToken
8	SlaveHoldToken	/ PH == 3 && sents == Multiple Transmit => QueDelete(); pdu = CreateToken(); Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu); sents = 1; PH = 3	SlaveHoldToken
9	SlaveHoldToken	/ PH == 4 && sents == Multiple Token => pdu = Deque(); Send.req (port, ChannelGroup, pdu) to all ports; Enque(pdu); sents = sents + 1	SlaveHoldToken

#	Current state	Event /condition => action	Next state
10	SlaveHoldToken	/ PH == 4 && sents == Multiple Token => QueDelete(); PH = 0; TR3 = TRUE	SlaveSolicitToken
11	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) to all ports other than rport; CStopSending(); ACStopSending(); ChannelGroup = NULL	SlaveWaitTD
12	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
13	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
14	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => CStopSending(); ACStopSending(); ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU); TR3 = TRUE	SlaveSolicitToken
15	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
16	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
17	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken
18	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) to all ports other than rport	SlaveHoldToken

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

8.2.4.2 Functions

The Channel control functions are shown in Table 156 and Table 157 for Master and Slave stations respectively.

Table 156 – Master station channel control functions

Name	Description
getNextPhase(current_state, current_phase)	Sets up the next internal phase (represented by the variable PH).
CmpPriority(mst_pri, SA)	Compares the priority levels and returns Low or High. Low: mst_pri < MyPriority (mst_pri == MyPriority && SA < MyAddr) High: mst_pri > MyPriority (mst_pri == MyPriority && SA > MyAddr)
CreateTokenRoute()	Generates the token passing path, and sets SetupNodes.
MCAST(mac_addr)	Sets the I/G bit of addr to 1, and returns the address converted to a multicast address.
SetInvitationFlag()	Sets the flag indicating whether or not the node is to return to the system. The available values are as follows: ON: Return to system OFF: Do not return to system NULL: Undefined
SetMeasureFlag()	Sets the flag indicating whether or not the transmission path delay is to be measured. The available values are as follows: ON: Measure transmission path delay OFF: Do not measure transmission path delay NULL: Undefined
SetNewTokenFlag()	Sets a flag indicating whether or not to regenerate a token. The available values are as follows: ON: Regenerate OFF: Do not regenerate
IsValidPort(port)	Assesses whether the port is valid, and returns TRUE if it is and FALSE if it is not.
IsSlavePortInvalidated()	Assesses whether or not a slave station with an invalid port exists, and returns TRUE if it does, and FALSE if it does not.
SlavePortStateChanged()	Returns TRUE if the port status of the slave station changed from link disconnected to link up, and FALSE if it did not change from link disconnected to link up.
LostTokenDst()	Finds the destination of the lost token.
AddReceivedToken(tseq_no)	Holds the sequence number of the received token.
IsReceivedToken(tseq_no)	Assesses whether tseq_no is included in the token sequence numbers held by AddReceivedToken, and returns TRUE if it is and FALSE if it is not.
IsMulticast(addr)	Returns TRUE if the address is a multicast address, and FALSE if the address is not a multicast address.
CreatePersuasion()	Creates and returns the persuasion-PDU.
CreateTestData()	Creates and returns the testData-PDU.
CreateTestDataAck()	Creates and returns the testDataAck-PDU.
CreateSetup()	Creates and returns the setup-PDU.
CreateSetupAck()	Creates and returns the setupAck-PDU.
CreateToken()	Creates and returns the token-PDU.
CReceived(pdu)	Delivers pdu to the Cyclic Transmission state machine.
ACReceived(pdu)	Delivers pdu to the Acyclic Transmission state machine.
Enque(pdus)	Queues pdus given in arguments. Queues in the seqNumber order of pdu when multiple pdus are given.
Deque()	Retrieves pdus queued by Enque from the start of the queue.
QueDelete()	Deletes all pdus from the queue.

Name	Description
QueLen()	Returns the size of the queue.
SetTrInfo(pdu)	Holds the information included in the setup-PDU received as pdu.
MyStatusSend(pdu)	Sends MyStatus frame transmission notification
GetTraAllowsND()	Returns a limit value for node unt transient transmissions.

Table 157 – Slave station channel control functions

Name	Description
SetTrInfo(pdu)	Holds the information included in the setup-PDU received as pdu.
AddReceivedToken(tseq_no)	Holds the sequence number of the received token.
IsReceivedToken(tseq_no)	Assesses whether or not the tseq_no is included in the token sequence numbers held by AddReceivedToken, and returns TRUE if it is and FALSE if it is not.
CreatePersuasion()	Creates and returns the persuasion-PDU.
CreateTestDataAck()	Creates and returns the testDataAck-PDU.
CreateSetupAck()	Creates and returns the setupAck-PDU.
CreateToken()	Creates and returns the token-PDU.
CReceived(pdu)	Delivers pdu to the Cyclic Transmission state machine.
ACReceived(pdu)	Delivers pdu to the Acyclic Transmission state machine.
GetTraAllowsND()	Returns a limit value for node unt transient transmissions.

8.2.4.3 Variables

The variables used in Channel control are shown in Table 158 and Table 159 for Master and Slave stations respectively.

Table 158 – Master station channel control variables

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

Name	Description
RingCheck	Flag indicating whether or not a ring check is to be performed
SynchronizationMaster	Flag indicating the synchronization manager.
PH	Indicates the subphase.

Table 159 – Slave station channel control variables

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

8.2.4.4 Timers

The timers used in Channel control are shown in Table 160 and Table 161 for Master and Slave stations respectively.

Table 160 – Master station channel control timers

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

Table 161 – Slave station channel control timers

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

8.2.5 Parameter dist

8.2.5.1 Parameter dist state machine

The states of the Parameter dist state machine for the Mater station are described in Table 162 with the details specified in Table 164.

The states of the Parameter dist state machine for the Slave station are described Table 163 with the details specified in Table 165.

Table 162 – Master station parameter dist states

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

Table 163 – Slave station parameter dist states

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

Table 164 – Master station parameter dist state table

#	Current state	Event /condition => action	Next state
1	IDLE	ParameterDistReady() => AC Send.req(ParameterCheck, data)	InitParamChecking
2	InitParamChecking	MyStatusReceived(pdu) / MyStatusFromAllNodes() == FALSE =>	InitParamChecking
3	InitParamChecking	MyStatusReceived(pdu) / MyStatusFromAllNodes() == TRUE && ParamAllNodesReceived() == FALSE => GetNonReceivedNodes(); AC Send.req(netno, nodeno, Parameter, data)	ParamSending
4	InitParamChecking	MyStatusReceived(pdu) / MyStatusFromAllNodes() == TRUE && ParamAllNodesReceived() == TRUE => AC Send.req(ParameterCheck, data)	ParamChecking
5	ParamSending	MyStatusReceived(pdu) / ParamAllNodesReceived() == FALSE =>	ParamSending
6	ParamSending	MyStatusReceived(pdu) / MyStatusFromAllNodes() == TRUE => AC Send.req(ParameterCheck, data)	ParamChecking
7	ParamChecking	MyStatusReceived(pdu) / ParamAllNodesChecked() == TRUE => AC Send.req(ParameterCheck, data)	ParamSent

#	Current state	Event /condition => action	Next state
8	ParamChecking	MyStatusReceived(pdu) / ParamAllNodesChecked() == FALSE =>	ParamChecking
9	ParamSent	MyStatusReceived(pdu) / ParamAllNodesChecked() == FALSE => GetNonCheckedNodes(); AC Send.req(netno, nodeno, Parameter, data)	ParamSending

Table 165 – Slave station parameter dist state table

#	Current state	Event /condition => action	Next state
1	ParamNone	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck =>	ParamNone
2	ParamNone	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter => DoParameterCheck(pdu)	ParamCheck
3	ParamCheck	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter =>	ParamCheck
4	ParamCheck	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE =>	ParamCheck
5	ParamCheck	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
6	ParamCheck	ParameterCheckComplete(Result) / Result == TRUE =>	ParamHold
7	ParamCheck	ParameterCheckComplete(Result) / Result == FALSE =>	ParamErr
8	ParamErr	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter => DoParameterCheck(pdu)	ParamCheck
9	ParamErr	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE =>	ParamErr
10	ParamErr	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone

#	Current state	Event /condition => action	Next state
11	ParamHold	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter =>	ParamCheck
12	ParamHold	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE => UpdateParameter(pdu)	ParamOK
13	ParamHold	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
14	ParamOK	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter =>	ParamCheck
15	ParamOK	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE => UpdateParameter(pdu)	ParamOK
16	ParamOK	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
17	ParamOK	MasterWatchTimer time-out =>	ParamHold

8.2.5.2 Functions

The Parameter dist functions are shown in Table 166 and Table 167 for Master and Slave stations respectively.

Table 166 – Master station parameter dist functions

Name	Description
ParameterDistReady	Ready for parameter distribution
ParamAllNodesReeived()	Assesses whether or not all nodes have received parameters. Returns TRUE if so and FALSE if not.
ParamAllNodesChecked()	Assesses whether or not all nodes have checked parameters. Returns TRUE if so and FALSE if not.
MyStatusFromAllNodes()	Assesses whether or not MyStatus has been received from all nodes. Returns TRUE if so and FALSE if not.
GetNonReceivedNodes()	Gets the network numbers and node numbers of nodes that have not received parameters.
GetNonCheckedNodes()	Gets the network numbers and node numbers of the nodes that have not checked parameters.
GetNetNo(pdu)	Gets the network number from pdu.
GetNodeNo(pdu)	Gets the node number from pdu.

Table 167 – Slave station parameter dist functions

Name	Description
DoParameterCheck(pdu)	Checks the parameters given by pdu. Calls ParameterCheckComplete(result) once completed. Sets result to TRUE if parameters are normal and to FALSE if parameters are abnormal.
CmpParamID(pdu)	Compares the parameter ID of the parameter given by pdu with the parameter ID of the parameter given by pdu in DoParameterCheck. Returns TRUE if they match and FALSE if they do not match.
UpdateParameter(pdu)	Sets the parameters given in pdu as the new parameters.

8.2.6 Synchronous trigger

8.2.6.1 Primitive definition

Synchronous trigger issues Synchronous trigger internal Internal.ind service to FSPM.

8.2.6.2 Synchronous trigger state machine

The states of the Synchronous trigger state machine for the Master station are described in Table 168 with the details specified in Table 170.

The states of the Synchronous trigger state machine for the Slave station are described Table 169 with the details specified in Table 171.

Table 168 – Master station synchronous trigger states

Name	Description
ACTIVE	Operating state

Table 169 – Slave station synchronous trigger states

Name	Description
ACTIVE	Operating state

Table 170 – Master station synchronous trigger state table

#	Current state	Event /condition => action	Next state
1	ACTIVE	MyStatusSend (pdu) / (SynchronizationMaster == TRUE) && (getMyStatusSyncFlag (pdu) == TRUE) => Synchronous Trigger Internal.ind	ACTIVE
2	ACTIVE	MyStatusReceived(pdu) / (SynchronizationMaster == FALSE) && getMyStatusSyncFlag () == TRUE => Synchronous Trigger Internal.ind	ACTIVE

Table 171 – Slave station synchronous trigger state table

#	Current state	Event /condition => action	Next state
1	ACTIVE	MyStatusReceived(pdu) / getMyStatusSyncFlag () == TRUE => Synchronous Trigger Internal.ind	ACTIVE

8.2.6.3 Functions

Functions enabled in Synchronous trigger are shown in Table 172.

Table 172 – Synchronous trigger functions

Name	Description
getMyStatusSyncFlag	Acquires the value of SyncFlag of the MyStatus frame.

8.2.6.4 Variables

Synchronous trigger uses no variables.

8.2.7 Timer**8.2.7.1 Timer state machine**

The Timer state machine states are described in Table 173 and Table 174 with the detail is specified in Table 175 and Table 176.

Table 173 – Timer states – Best effort type

Name	Description
ACTIVE	Operating state

Table 174 – Timer states – Fixed cycle type

Name	Description
ACTIVE	Operating state
WAIT	Waiting state

Table 175 – Timer state table – Best effort type

#	Current state	Event /condition => action	Next state
1	ACTIVE	/ => MyStatusSendTimingFlag = ON	ACTIVE

Table 176 – Timer state table – Fixed cycle type

#	Current state	Event /condition => action	Next state
1	WAIT	/ MyStatusSendTimingFlag == OFF => ScanTimer startup	ACTIVE
2	ACTIVE	ScanTimer time-out => MyStatusSendTimingFlag = ON	WAIT

8.2.7.2 Functions

The Timer uses no functions.

8.2.7.3 Variables

The variables used by the Timer are shown in Table 177.

Table 177 – Timer variables

Name	Description
MyStatusSendTimingFlag	Sets the communication start flag. ON; Start communication OFF: Wait

8.2.7.4 Timers

The timers used by the fixed cycle timer are shown in Table 178.

Table 178 – Fixed cycle timer

Name	Description
ScanTimer	Determines the communication cycle time.

8.2.8 Measure transmission

8.2.8.1 Primitive definition

FSPM issues a StartMeasureInternal.req service to MeasureTransmission. MeasureTransmission issues a StartMeasureInternal.cnf service to FSPM.

8.2.8.2 Measure transmission state machine

The states of the Measure transmission state machine for the Mater station are described in Table 179 with the details specified in Table 181.

The states of the Measure transmission state machine for the Slave station are described in Table 180 with the details specified in Table 182.

Table 179 – Master station measure transmission states

Name	Description
M_IDLE	Initial state
SM_MEASURE	Measuring as synchronization manager
SM_WAIT_MEASUREACK	Waiting for MeasureAck as synchronization manager
M_WAIT_MEASUREACK	Measuring
M_WAIT_UPDATE	Waiting for measurement result update

Table 180 – Slave station measure transmission states

Name	Description
S_IDLE	Waiting
S_WAIT_MEASUREACK	Measuring
S_WAIT_UPDATE	Waiting for measurement result update

Table 181 – Master station measure transmission state table

#	Current state	Event /condition => action	Next state
1	M_IDLE	Start Measure Internal.req / (SynchronizationMaster == TRUE) && (MeasureTransmission Flag == TRUE) => MeasureTransmission Flag == FALSE slaveCounterMeasureDone = 0 slaveCounterMeasureAll = getSlaveCounterMeasureAll()	SM_MEASURE
2	M_IDLE	MeasureReceived(rport, DA, SA, pdu) / (SynchronizationMaster == FALSE) && DA == MyAddr => startRoundtripTimer(); DA = SA; pdu = CreateMeasureAck(); SendMeasureAck(rport, DA, pdu); roundtrip = stopRoundtripTimer();	M_WAIT_OFFSET
3	M_IDLE	MeasureReceived(rport, DA, SA, pdu) / (SynchronizationMaster == FALSE) && (DA != MyAddr) => SendMeasureAck(sport, DA, pdu) to all ports other than rport	M_WAIT_MEASUREACK
4	M_IDLE	Get Offset Internal.req / SynchronizationMaster == FALSE => Get Offset Internal.cnf(offset)	M_IDLE

#	Current state	Event /condition => action	Next state
5	SM_MEASURE	/ slaveCounterMeasureDone < slaveCounterMeasureAll => startRoundTripTimer(); DA = getTerminatedSlaveAddress(slaveCounterMeasureDone); port = getTerminatedSlaveDirection(slaveCounterMeasureDone); pdu = CreateMeasure(); SendMeasure(port, DA, pdu)	SM_WAIT_MEASURE_ACK
6	SM_MEASURE	/ slaveCounterMeasureDone == slaveCounterMeasureAll => DA = ChannelGroup; pdu = CreateUpdate(); SendUpdate(sport, DA, pdu) to all ports Start Measure Internal.cnf(TRUE)	M_IDLE
7	SM_WAIT_MEASURE_ACK	MeasureAckReceived() => roundtrip = stopRoundTripTimer(); offset = getOffset(roundtrip); DA = getTerminatedSlaveAddress(SlaveCounterMeasureDone); sport = getTerminatedSlaveDirection(SlaveCounterMeasureDone); pdu = CreateOffset(offset); SendOffset(sport, DA, pdu); slaveCounterMeasureDone++	SM_MEASURE
8	SM_WAIT_MEASURE_ACK	IsRoundtripTimerOver() => Start Measure Internal.cnf(FALSE)	M_IDLE
9	M_WAIT_MEASURE_ACK	MeasureAckReceived(rport, DA, SA, pdu) => roundtrip = stopRoundtripTimer(); SendMeasureAck(sport, DA, pdu); to all ports other than rport startArrivalOffsetTimer()	M_WAIT_OFFSET
10	M_WAIT_MEASURE_ACK	IsRoundtripTimerOver()	M_IDLE
11	M_WAIT_OFFSET	OffsetReceived(rport, DA, SA, pdu) => roundtrip = stopRoundtripTimer(); SendOffset(sport, DA, pdu) to all ports other than rport startArrivalOffsetTimer()	M_WAIT_UPDATE
12	M_WAIT_OFFSET	IsArrivalOffsetTimerOver()	M_IDLE
13	M_WAIT_UPDATE	UpdateReceived(rport, DA, SA, pdu) => SendUpdate(sport, DA, pdu) to all ports other than rport offset = generateOffset(roundtrip, distributeOffset)	M_IDLE
14	M_WAIT_UPDATE	IsArrivalUpdateTimerOver()	M_IDLE

Table 182 – Slave station measure transmission state table

#	Current state	Event /condition => action	Next state
1	S_IDLE	MeasureReceived(rport, DA, SA, pdu) / DA == MyAddr => startRoundtripTimer(); DA = SA; pdu = CreateMeasureAck(); SendMeasureAck(rport, DA, pdu); roundtrip = stopRoundtripTimer();	S_WAIT_OFFSET
2	S_IDLE	MeasureReceived(rport, DA, SA, pdu) / DA != MyAddr => SendMeasure(sport, DA, pdu) to all ports other than rport startRoundtripTimer();	S_WAIT_MEASUREACK
3	S_IDLE	Get Offset Internal.req => Get Offset Internal.cnf(offset)	S_IDLE
4	S_WAIT_MEASUREACK	MeasureAckReceived(rport, DA, SA, pdu) => roundtrip = stopRoundtripTimer(); SendMeasureAck(sport, DA, pdu) to all ports other than rport startArrivalOffsetTimer();	S_WAIT_OFFSET
5	S_WAIT_MEASUREACK	IsRoundtripTimerOver()	S_IDLE
6	S_WAIT_OFFSET	OffsetReceived(rport, DA, SA, pdu) => roundtrip = stopRoundtripTimer(); SendOffset(sport, DA, pdu) to all ports other than rport startArrivalOffsetTimer();	S_WAIT_UPDATE
7	S_WAIT_OFFSET	IsArrivalOffsetTimerOver()	S_IDLE
8	S_WAIT_UPDATE	UpdateReceived(rport, DA, SA, pdu) => SendUpdate(sport, DA, pdu) to all ports other than rport offset = generateOffset(roundtrip, distributeOffset)	S_IDLE
9	S_WAIT_UPDATE	IsArrivalUpdateTimerOver() =>	S_IDLE

8.2.8.3 Functions

Functions used by the Measure transmission are shown in Table 183 and Table 184 for the Master and the Slave stations respectively.

Table 183 – Master station measure transmission functions

Name	Description
getSlaveCounterMeasureAll()	Acquires the number of slave stations that perform delay measurement.
getTerminatedSlaveAddress(s)	Acquires the MAC address of the terminating slave station.
getTerminatedSlaveDirection(s)	Acquires the port connected to the terminating slave station.
getOffset(roundtrip)	Acquires the offset value from roundtrip.
CreateMeasure()	Creates and returns the measure-PDU.
CreateMeasureAck()	Creates and returns the measureAck-PDU.
CreateOffset(offset)	Creates and returns the offset-PDU that reflects the offset value.

Name	Description
CreateUpdate()	Creates and returns the update-PDU.
SendMeasure(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
SendMeasureAck(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
SendOffset(port, DA, pdu)	Sends port, DA, SA, and pdu to the Channel Control state machine.
SendUpdate(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
startRoundtripTimer()	Starts the RoundTripTimer.
stopRoudtripTimer()	Stops the RoundTripTimer, and returns the RoundTripTimer value.
IsRoundtripTimerOver()	Returns TRUE when the RoundTripTimer times out and FALSE in all other cases.
startArrivalOffsetTimer()	Starts the ArrivalOffsetTimer.
stopArrivalOffsetTimer()	Stops the ArrivalOffsetTimer.
IsArrivalOffsetTimerOver()	Returns TRUE when theArrivalOffsetTimer times out and FALSE in all other cases.
startArrivalUpdateTimer()	Starts the ArrivalUpdateTimer.
stopArrivalUpdateTimer()	Stops the ArrivalUpdateTimer.
IsArrivalUpdateTimerOver()	Returns TRUE when the ArrivalUpdateTimer times out and FALSE in all other cases.

Table 184 – Slave station measure transmission functions

Name	Description
CreateMeasure()	Creates and returns the measure-PDU.
CreateMeasureAck()	Creates and returns the measureAck-PDU.
CreateUpdate()	Creates and returns the update-PDU.
SendMeasure(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
SendMeasureAck(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
SendUpdate(port, DA, pdu)	Sends port, DA, and pdu to the Channel Control state machine.
startRoundtripTimer()	Starts the RoundTripTimer.
stopRoudtripTimer()	Stops the RoundTripTimer, and returns the RoundTripTimer value.
IsRoundtripTimerOver()	Returns TRUE when the RoundTripTimer times out, and FALSE in all other cases.
startArrivalOffsetTimer()	Starts the ArrivalOffsetTimer.
stopArrivalOffsetTimer()	Stops the ArrivalOffsetTimer.
IsArrivalOffsetTimerOver()	Returns TRUE when the ArrivalOffsetTimer times out, and FALSE in all other cases.
startArrivalUpdateTimer()	Starts the ArrivalUpdateTimer.
stopArrivalUpdateTimer()	Stops the ArrivalUpdateTimer.
IsArrivalUpdateTimerOver()	Returns TRUE when the ArrivalUpdateTimer times out, and FALSE in all other cases.

8.2.8.4 Variables

Measure transmission variables are shown in Table 185.

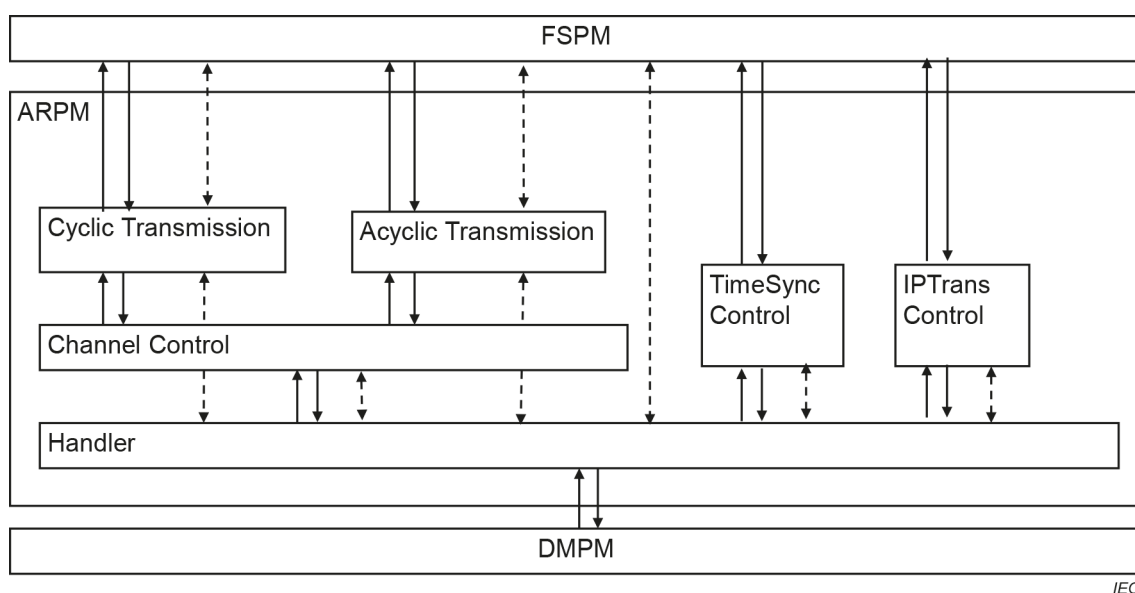
Table 185 – Master station measure transmission variables

Name	Description
slaveCounterMeasureDone	Indicates the number of slave stations for which delay measurement is completed.
SlaveCounterMeasureAll	Indicates the total number of slave stations for which delay measurement is performed.
roundtrip	Indicates the round-trip time.
offset	Indicates the offset value.
roundTrip	Indicates the round-trip time.
offset	Indicates the offset value.

8.3 ARPM type T

8.3.1 Overview

ARPM provides an interface to FSPM or DMPM. As shown in Figure 23, ARPM consists of 6 sub protocol machines. A solid line indicates issuance of a service, and a broken line indicates linkage between protocol machines using variables.



IEC

Figure 23 – Structure of ARPM T

8.3.2 Cyclic Transmission

8.3.2.1 Primitive definition

Services provided by Cyclic Transmission are shown in Table 186.

Table 186 – Primitives provided by Cyclic Transmission

Primitive Name	Publisher
C_Update.req	FSPM
C_Update.ind	ARPM

8.3.2.2 Cyclic Transmission state machine

States of the Cyclic Transmission state machine are shown in Table 187.

Table 187 – Cyclic Transmission state

Name	Description
INACTIVE	Waiting for activation
ACTIVE	Cyclic transmission in progress

Details of Cyclic Transmission state machine are shown in Table 188.

Table 188 – Cyclic Transmission state table

#	Current state	Event /Condition => Action	Next state
1	INACTIVE	/ CTicket = TRUE => seqNo = 0	ACTIVE
2	INACTIVE	C_Update.req(fType, Address, Size, Data) => Update(Address, Size, Data)	INACTIVE
3	ACTIVE	/ Length(RemainingData) > 0 => subPayloads = CreateSubPayloadBlock(DA, TxAsyncInfo, seqNo) pdu = CreateCyclic-PDU(fType, subPayloads) SendCyclic.req(pdu)	ACTIVE
4	ACTIVE	/ Length(RemainingData) == 0 => CTicket = FALSE	INACTIVE
5	ACTIVE	SendCyclic.ind(pdu) => C_Update.ind(fType, Address, Size, Data)	ACTIVE
6	INACTIVE	SendCyclic.ind(pdu) => C_Update.ind(fType, Address, Size, Data)	INACTIVE

8.3.2.3 Function

Functions used for Cyclic Transmission are shown in Table 189.

Table 189 – Functions used for Cyclic Transmission

Name	Contents
Length(data)	Acquires the size of data given by the argument
Update()	Updates data to be sent by cyclic transmission
CreatSubPayloadBlock(DA, TxAsyncInfo, seqNo)	The corresponding SubPayload or SubPayloadBlock is generated using the destination address, the request for transient transmission, and the sequential number, and the remaining data is set as RemainingData
CreateCyclic-PDU(fType, subPayloads)	Generates the cyclic transmission frame specified by the frame type

8.3.2.4 Variable

Variables used for Cyclic Transmission are shown in Table 190.

Table 190 – Variables used in Cyclic Transmission

Name	Contents
CTicket	The flag that represents a transmission permission of cyclic transmission
RemainingData	Remaining data to be sent in cyclic transmission

8.3.3 Acyclic Transmission

8.3.3.1 Primitive definition

Services provided by Acyclic Transmission are shown in Table 191.

Table 191 – Primitives provided by AcyclicTransmission

Primitive Name	Publisher
AC_Update.req	FSPM
AC_Update.ind	ARPM

8.3.3.2 Acyclic Transmission state machine

States of the Acyclic Transmission state machine are shown in Table 192.

Table 192 – Acyclic Transmission state

Name	Description
INACTIVE	No transient transmission in progress
ACTIVE	Transient transmission in progress

Details of Acyclic Transmission state machine are shown in Table 193.

Table 193 – Acyclic Transmission state table

#	Current state	Event /Condition => Action	Next state
1	INACTIVE	/ ACTicket == TRUE =>	ACTIVE
2	ACTIVE	AC_Update.req(fType, DA, Data) / fType != 0xC2 => pdu = CreateAcyclicPDU (fType, DA, Data) SendAcyclic.req(pdu)	ACTIVE
3	ACTIVE	AC_Update.req(fType, DA, Data) / (Status ==C_Detection Status == S_Detection) && (fType == 0xC2) => pdu = CreateAcyclicPDU (fType, DA, Data) SendAcyclic.req(pdu) ACTicket = FALSE	INACTIVE

#	Current state	Event /Condition => Action	Next state
4	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Priority) && (fType == 0xC0) && (CmpPriority() == High) => AC_Update.ind(fType, pdu)	ACTIVE
5	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Priority) && (fType == 0xC0) && (CmpPriority() == Low) => AC_Update.ind(fType, pdu) TimerStop(MMTimer) NodeType = C SetStatus(P_ON) ACTicket = FALSE	INACTIVE
6	ACTIVE	TimerExpired(MMTimer) => ACTicket = FALSE	INACTIVE
7	ACTIVE	SendAcyclic.ind(pdu) / (Status == C_Detection Status == S_Detection) && (fType == 0xC1) && (IsPrevChanged() == TRUE) => AC_Update.ind(fType, pdu)	ACTIVE
8	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Detection) && (fType == 0xC2) => ANodes++ AC_Update.ind(fType, pdu)	ACTIVE
9	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Detection) && (fType == 0xC1) => AC_Update.ind(fType, pdu) TimerStop(CCDTimer) SetStatus(M_Priority) TimerStart(MMTimer)	ACTIVE
10	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Detection) && (fType == 0xC0) && (CmpPriority() == High) => AC_Update.ind(fType, pdu) TimerStop(CCDTimer) SetStatus(M_Priority) TimerStart(MMTimer)	ACTIVE
11	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Detection) && (fType == 0xC0) && (CmpPriority() == Low) => AC_Update.ind(fType, pdu) TimerStop(CCDTimer) NodeType = C SetStatus(P_ON) ACTicket = FALSE	INACTIVE
12	ACTIVE	TimerExpired(CCDTimer) / ANodes > 0 => ACTicket = FALSE	INACTIVE
13	ACTIVE	TimerExpired(CCDTimer) / ANodes == 0 => TimerStart(CCDTimer)	ACTIVE

#	Current state	Event /Condition => Action	Next state
14	INACTIVE	SendAcyclic.ind(pdu) / (NodeType == M NodeType == C) && (Status == TimeSync Status == NetworkConfig) && (fType == 0xC0) && (CmpPriority() == High) => AC_Update.ind(fType, pdu) SetStatus(P_ON)	INACTIVE
15	INACTIVE	SendAcyclic.ind(pdu) / (NodeType == M NodeType == C) && (Status == TimeSync Status == NetworkConfig) && (fType == 0xC0) && (CmpPriority() == Low) => AC_Update.ind(fType, pdu) NodeType = C SetStatus(P_ON)	INACTIVE
16	INACTIVE	SendAcyclic.ind(pdu) / (Status == TimeSync Status == NetworkConfig) && (fType == 0xC1) && (IsPrevChanged() == FALSE) => AC_Update.ind(fType, pdu)	INACTIVE
17	ACTIVE	SendAcyclic.ind(pdu) / (Status == TS_FT) && (fType == 0xC0) && (CmpPriority() == High) => AC_Update.ind(fType, pdu) SetStatus(P_ON)	ACTIVE
18	ACTIVE	SendAcyclic.ind(pdu) / (Status == TS_FT) && (fType == 0xC0) && (CmpPriority() == Low) => AC_Update.ind(fType, pdu) NodeType = C SetStatus(P_ON) ACTicket = FALSE	INACTIVE
19	ACTIVE	TimerExpired(TSChangeTimer) => ACTicket = FALSE	INACTIVE
20	ACTIVE	SendAcyclic.ind(pdu) / (NodeType == M NodeType == C) && (Status == TS_FT) && (fType == 0xC1) => AC_Update.ind(fType, pdu) SetStatus(P_ON) ACTicket = FALSE	INACTIVE
21	ACTIVE	SendAcyclic.ind(pdu) / (NodeType == M) && (Status == TS_FT) && (fType == 0xC2) => AC_Update.ind(fType, pdu) SetStatus(M_Priority) ACTicket = FALSE	INACTIVE
22	ACTIVE	SendAcyclic.ind(pdu) / (NodeType == C) && (Status == TS_FT) && (fType == 0xC2) => AC_Update.ind(fType, pdu) SetStatus(P_ON) ACTicket = FALSE	INACTIVE

#	Current state	Event /Condition => Action	Next state
23	ACTIVE	SendAcyclic.ind(pdu) / (Status == C_Detection Status == S_Detection) && (fType == 0xC1) && (IsPrevChanged() == FALSE) => AC_Update.ind(fType, pdu) ACTicket = FALSE	INACTIVE
24	INACTIVE	SendAcyclic.ind(pdu) / (Status == TimeSync Status == NetworkConfig) && (fType == 0xC1) && (IsPrevChanged() == TRUE) => AC_Update.ind(fType, pdu) SetStatus(P_ON)	INACTIVE

8.3.3.3 Function

Functions used for Acyclic Transmission are shown in Table 194.

Table 194 – Functions used for Acyclic Transmission

Name	Contents
CreateAcyclicPDU (fType, DA, Data)	When the data given as an argument for generating the transient transmission frame specified by fType exceeds the maximum size possible in the PDU, the data is divided and sent and received using multiple PDUs.
TimerStart()	Starts timer
TimerStop()	Stops timer
TimerExpired()	Detects timer timeout
CmpPriority()	Determines the priority as the management master station and returns Low (the own station's priority is low) or High (the own station's priority is high)
IsPrevChanged()	As the configuration information difference with the previously received Detection, returns TRUE if it is received for the first time or different, and returns FALSE if it is the same

8.3.3.4 Variable

Variables used for Acyclic Transmission are shown in Table 195.

Table 195 – Variables used in Acyclic Transmission

Name	Description
ACTicket	Flag that represents the start of transient transmission
ANodes	The number of connected stations detected

8.3.4 Channel Control

8.3.4.1 Primitive definition

Services provided by Channel Control are shown in Table 196.

Table 196 – Primitives provided by Channel Control

Primitive Name	Publisher
SendCyclic.req	ARPM
SendCyclic.ind	ARPM
SendAcyclic.req	ARPM
SendAcyclic.ind	ARPM

8.3.4.2 Channel Control state machine

States of the Channel Control state machine are shown in Table 197.

Table 197 – Channel Control state

State	Description
INACTIVE	Waiting for activation
ACTIVE	In active

The Channel Control state machine is shown in Table 198.

Table 198 – Channel Control state machine

#	Current state	Event /Condition => Action	Next state
1	INACTIVE	/ ChannelCtrl_Flag == TRUE =>	ACTIVE
2	ACTIVE	SendAcyclic.req(pdu) / (Status == M_Priority Status == M_Detection Status == C_Detection Status == S_Detection) && (CTicket == FALSE) => sdports = CheckAvaPorts() Send.req(sdport, DA, pdu)	ACTIVE
4	ACTIVE	SendCyclic.req(pdu) / Status == TS_CT => sdports = CheckAvaPorts() Send.req(sdport, DA, pdu)	ACTIVE
5	ACTIVE	SendAcyclic.req(pdu) / (Status == TS_CT) && (CTicket == FALSE) && (IsRSVAcyclic() == TRUE) => sdports = CheckAvaPorts() Send.req(sdport, DA, pdu)	ACTIVE
7	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (Status == TS_CT) && (DA == MyAdd) && (SA != MyAdd) && (fType == 0xC4 fType == 0xC5 fType == 0xC6 fType == 0xC7) => SendCyclic.ind(pdu)	ACTIVE

#	Current state	Event /Condition => Action	Next state
8	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (Status == TS_CT) && (DA == BROADCAST DA == MULTICAST) && (SA != MyAdd) && (fType == 0xC4 fType == 0xC5 fType == 0xC6 fType == 0xC7) => SendCyclic.ind(pdu) sdports = CheckAvaPort() Send.req(sdports, DA, pdu)	ACTIVE
9	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == BROADCAST) && (SA != MyAdd) && (fType == 0xC0) => SendAcyclic.ind(pdu) sdports = CheckAvaPort() Send.req(sdports, BROADCAST, pdu)	ACTIVE
10	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == BROADCAST) && (SA != MyAdd) && (fType == 0xC1) => SendAcyclic.ind(pdu) pdu = CreateAcyclicPDU (fType, pdu) sdports = CheckAvaPort() Send.req(sdports, BROADCAST, pdu)	ACTIVE
11	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == MyAdd) && (SA != MyAdd) && (fType == 0xC2 fType == 0xC3) => SendAcyclic.ind(pdu)	ACTIVE
12	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA != MyAdd) && (SA != MyAdd) && (fType == 0xC2 fType == 0xC3) => sdports = CheckAvaPort() Send.req(sdports, DA, pdu)	ACTIVE
13	ACTIVE	TimerExpired(TSChangeTimer) => CTicket = FALSE ACTicket = FALSE ChannelCtrl_Flag = FALSE	INACTIVE
14	ACTIVE	/ (CTicket == FALSE) && (ACTicket == FALSE) => ChannelCtrl_Flag = FALSE	INACTIVE
15	INACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == BROADCAST) && (SA != MyAdd) && (fType == 0xC0) => SendAcyclic.ind(pdu) sdports = CheckAvaPort() Send.req(sdports, BROADCAST, pdu)	INACTIVE
16	INACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == BROADCAST) && (SA != MyAdd) && (fType == 0xC1) => SendAcyclic.ind(pdu) pdu = CreateAcyclicPDU (fType, pdu) sdports = CheckAvaPort() Send.req(sdports, BROADCAST, pdu)	INACTIVE

8.3.4.3 Function

Functions used for Channel Control are shown in Table 199.

Table 199 – Functions used for Channel Control

Name	Description
CheckAvaPorts()	Checks available transmission ports
TimerExpired()	Detects timer time out

8.3.4.4 Variable

Variables used for Channel Control are shown in Table 200.

Table 200 – Variables used in Channel Control

Name	Description
MyAdd	Host station address
BROADCAST	Broadcast address
MULTICAST	Multicast address
ChannelCtrl_Flag	Flag that represents Channel Control execution permission
Status	Indicates the current operating state • During power-on or exceptional operation: P_ON • During management master station arbitration: M_Priority • The host station is the management master station, and the connection configuration is being detected: M_Detection • The host station is the control master station and connection configuration detection is being awaited: C_Detection • The host station is a slave station and connection detection is being awaited: S_Detection • During control communication by CC-Link IE TSN: TS_CT • Time synchronization in progress: TimeSync • Network setting in progress: NetworkConfig • During free communication by TS0: TS_FT • During steady operation other than the above: Running

8.3.5 TimeSync Control

8.3.5.1 Primitive definition

Services provided by TimeSync Control are shown in Table 201.

Table 201 – Primitives provided by TimeSync Control

Primitive Name	Publisher
TimeSync.req	FSPM
TimeSync.ind	ARPM

8.3.5.2 TimeSync Control state machine

TimeSync Control manages the state of time synchronization based on IEEE Std 802.1AS or IEEE Std 1588. States of the state machine are shown in Table 202.

Table 202 – TimeSync Control state

Name	Description
ACTIVE	Time synchronization is in active
INACTIVE	Waiting for time synchronization to be active

Details of TimeSync Control state machine are shown in Table 203.

Table 203 – TimeSync Control state table

#	Current state	Event /Condition => Action	Next state
1	INACTIVE	/ TimeSyncCtrl_Flag == TRUE => TimeSyncStatus = NotCompleted	ACTIVE
2	ACTIVE	/ TimeSyncStatus == NotCompleted => Time synchronization compliant with IEEE Std 802.1AS or IEEE Std 1588 TimeSyncStatus = CheckTimeSyncStatus()	ACTIVE
3	ACTIVE	/ TimeSyncStatus == Completed TimeSyncStatus == Error => TimeSyncCtrl_Flag = FALSE TimeSync.ind(TimeSyncStatus)	INACTIVE
4	ACTIVE	TimeSync.req(data) => TimeSync.cnf(TimeSyncManagement(data))	ACTIVE
5	INACTIVE	TimeSync.req(data) => TimeSync.cnf(TimeSyncManagement(data))	INACTIVE

8.3.5.3 Function

Functions used for TimeSync Control are shown in Table 204.

Table 204 – Functions used for TimeSync Control

Name	Description
CheckTimeSyncStatus()	Checks the execution state of time synchronization • In Progress: NotCompleted • Normal completion: Completed • Error: Error
TimeSyncManagement(data)	According to the arguments, acquires and sets data related to IEEE Std 802.1AS or IEEE Std 1588, and returns the result

8.3.5.4 Variable

Variables used for TimeSync Control are shown in Table 205.

Table 205 – Variables used in TimeSync Control

Name	Description
TimeSync_Flag	Flag indicating start of TimeSync Control
TimeSyncStatus	Execution state of time synchronization

8.3.6 IPTrans Control

8.3.6.1 Primitive definition

Services provided by IPTrans Control are shown in Table 206.

Table 206 – Primitives provided by IPTrans Control

Primitive Name	Publisher
SLMPSend.req	FSPM
SLMPSend.ind	ARPM

8.3.6.2 IPTrans Control state machine

States of the IPTrans Control state machine are shown in Table 207.

Table 207 – IPTrans Control state in master station

Name	Description
ACTIVE	In active
INACTIVE	Waiting to be active

The IPTrans Control state machine is shown in Table 208.

Table 208 – IPTrans Control state table

#	Current state	Event /Condition => Action	Next state
1	INACTIVE	/ IPTransCtrl_Flag == TRUE => NetworkConfigStatus = NotCompleted	ACTIVE
2	ACTIVE	SLMPSend.req(SImpData) / NetworkConfigStatus == NotCompleted => pdu = CreateSImpPdu(SImpData) sdports = CheckAvaPorts() Send.req(sdport, pdu)	ACTIVE
3	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (NetworkConfigStatus == NotCompleted) && (IsSLMPPdu(pdu) == TRUE) => SLMPSend.ind(pdu) NetworkConfigStatus = CheckNetworkConfigStatus()	ACTIVE
4	ACTIVE	/ NetworkConfigStatus == Completed NetworkConfigStatus == Error => IPTransCtrl_Flag = FALSE	INACTIVE

#	Current state	Event /Condition => Action	Next state
5	INACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (IsSLMPPdu(pdu) == TRUE) && (Status == TS_FT) => SetStatus(TimeSync)	INACTIVE

8.3.6.3 Function

Functions used for IPTrans Control are shown in Table 209.

Table 209 – Functions used for IPTrans Control

Name	Contents
CreatSimpPdu(SImpData)	Generates an IP frame that includes SImpData
CheckAvaPorts()	Checks available transmission ports
CheckNetworkConfigStatus()	Checks the execution state of network setting • In Progress: NotCompleted • Normal completion: Completed • Error: Error

8.3.6.4 Variable

Variables used for IPTrans Control are shown in Table 210.

Table 210 – Variables used in IPTrans Control

Name	Description
IPTransCtrl_Flag	Trigger of executing IP data transmission
NetworkConfigStatus	Execution state of network setting

8.3.7 Handler

8.3.7.1 Handler state machine

States of the Handler state machine are shown in Table 211.

Table 211 – Handler state

Name	Description
DOWN	Not activated
INIT	During initialization
RUN	During runtime communication

Handler state machines are shown in Table 212, Table 213, and Table 214.

Table 212 – DOWN state machine

#	Current state	Event /Condition => Action	Next state
1	DOWN	POWER ON => SetStatus(P_ON) NodeType = ValidNodeType()	INIT

Table 213 – INIT state machine

#	Current state	Event /Condition => Action	Next state
1	INIT	/ (NodeType == M) && (Status == P_ON) => SetStatus(M_Priority) TimerStart(MMTimer) CallTsCtrl(ChannelCtrl)	INIT
2	INIT	/ (NodeType == C) && (Status == P_ON) => SetStatus(C_Detection) CallTsCtrl(ChannelCtrl)	INIT
3	INIT	/ (NodeType == S) && (Status == P_ON) => SetStatus(S_Detection) CallTsCtrl(ChannelCtrl)	INIT
4	INIT	/ (NodeType == M) && (Status == M_Priority) && (ACTicket == FALSE) => SetStatus(M_Detection) ANodes = 0 TimerStart(CCDTimer) CallTsCtrl(ChannelCtrl)	INIT
5	INIT	/ (NodeType == M) && (Status == M_Detection) && (ACTicket == FALSE) => SetStatus(TimeSync) CallTsCtrl(TimeSyncCtrl)	INIT
6	INIT	/ ((NodeType == C && Status == C_Detection) (NodeType == S && Status == S_Detection)) && (ACTicket == FALSE) => SetStatus(TimeSync) CallTsCtrl(TimeSyncCtrl)	INIT
7	INIT	/ (Status == TimeSync) && (TimeSyncStatus == Completetd) => SetStatus(NetworkConfig) CallTsCtrl(IPTransCtrl)	INIT
8	INIT	/ (NodeType == M) && (Status == TimeSync) && (TimeSyncStatus == Error) => SetStatus(M_Detection) ANodes = 0 TimerStart(CCDTimer) CallTsCtrl(ChannelCtrl)	INIT

#	Current state	Event /Condition => Action	Next state
9	INIT	/ (NodeType == C) && (Status == TimeSync) && (TimeSyncStatus == Error) => SetStatus(C_Detection) CallTsCtrl(ChannelCtrl)	INIT
10	INIT	/ (NodeType == S) && (Status == TimeSync) && (TimeSyncStatus == Error) => SetStatus(S_Detection) CallTsCtrl(ChannelCtrl)	INIT
11	INIT	/ (Status == NetworkConfig) && (NetworkConfigStatus == Completetd) => SetupTS() TsCounter = 0 TimerStart(TSChangeTimer) SetStatus(TsCounter) CallTsCtrl(TsCounter)	RUN
12	INIT	/ (NodeType == M) && (Status == NetworkConfig) && (NetworkConfigStatus == Error) => SetStatus(M_Detection) ANodes = 0 TimerStart(CCDTimer) CallTsCtrl(ChannelCtrl)	INIT
13	INIT	/ (NodeType == C) && (Status == NetworkConfig) && (NetworkConfigStatus == Error) => SetStatus(C_Detection) CallTsCtrl(ChannelCtrl)	INIT
14	INIT	/ (NodeType == S) && (Status == NetworkConfig) && (NetworkConfigStatus == Error) => SetStatus(S_Detection) CallTsCtrl(ChannelCtrl)	INIT

Table 214 – RUN state machine

#	Current state	Event /Condition => Action	Next state
1	RUN	TimerExpired(TSChangeTimer) => SetNextTs(TsCounter) TimerStart(TSChangeTimer) SetStatus(TsCounter) CallTsCtrl(TsCounter)	RUN
2	RUN	TimerExpired(TSChangeTimer) / (IsInvitation() == TRUE) && (GetNextTs() == 0) => TimerStart(TSChangeTimer) SetStatus(M_Detection) CallTsCtrl(M_Detection)	RUN
3	RUN	/ (TsCounter == 0) && (NodeType == M) && (Status == P_ON) => SetStatus(M_Priority) TimerStart(MMTimer) CallTsCtrl(ChannelCtrl)	RUN
4	RUN	/ (TsCounter == 0) && (NodeType == C) && (Status == P_ON) => SetStatus(C_Detection) CallTsCtrl(ChannelCtrl)	RUN
5	RUN	/ (TsCounter == 0) && (NodeType == M) && (Status == M_Priority) && (ACTicket == FALSE) => SetStatus(M_Detection) ANodes = 0 TimerStart(CCDTimer) CallTsCtrl(ChannelCtrl)	RUN
6	RUN	/ (TsCounter == 0) && (Status == TimeSync) => SetStatus(NetworkConfig) CallTsCtrl(IPTransCtrl)	RUN
7	RUN	/ (TsCounter == 0) && (NodeType == C) && (Status == C_Detection) && (ACTicket == FALSE) => SetStatus(NetworkConfig) CallTsCtrl(IPTransCtrl)	RUN
8	RUN	/ (TsCounter == 0) && (NodeType == C) && (Status == C_Detection) && (ACTicket == FALSE) && (IsPrevChanged() == TRUE) => SetStatus(NetworkConfig) CallTsCtrl(IPTransCtrl)	RUN
9	RUN	/ (TsCounter == 0) && (NodeType == C) && (Status == C_Detection) && (ACTicket == FALSE) && (IsPrevChanged() == FALSE) => SetStatus(TS_FT) CallTsCtrl(TS_FT)	RUN

8.3.7.2 Function

Function used for Handler are shown in Table 215.

Table 215 – Functions used for Handler

Name	Description
SetStatus(Status)	Sets the operating state in accordance with the argument
SetupTS()	Sets the time slot
SetNextTS(TsCounter)	Sets the next time slot of the argument and sets the value of TSChangeTimer assigned to the corresponding time slot
CallTsCtrl(Status)	Controls Channel Control, Cyclic Transmission, Acyclic Transmission, TimeSync Control, and IPTrans Control according to arguments
GetNextTs(TsCounter)	Gets the next time slot of the argument
ValidNodeType()	Checks the node type of the host station
TimerStart()	Starts the timer
TimerExpired()	Detects timer timeout
IsPrevChanged()	As the configuration information difference with the previously received Detection, returns TRUE if it is received for the first time or different, and returns FALSE if it is the same

8.3.7.3 Variable

Variables used for Handler are shown in Table 216.

Table 216 – Variables used for Handler

Name	Description
Status	Indicates the current operating state • During power-on or exceptional operation: P_ON • During management master station arbitration: M_Priority • The host station is the management master station, and the connection configuration is being detected: M_Detection • The host station is the control master station and connection configuration detection is being awaited: C_Detection • The host station is a slave station and connection detection is being awaited: S_Detection • During control communication by CC-Link IE TSN: TS_CT • Time synchronization in progress: TimeSync • Network setting in progress: NetworkConfig • During free communication by TS0: TS_FT • During steady operation other than the above: Running
NodeType	Indicates the node type • Management master station: M • Control master station: C • Slave station: S
TsCounter	The counter for time slot management, 0 shall be TS0

9 DLL mapping protocol machine (DMPM)

9.1 DMPM type C

The structure of the DMPM is shown in Figure 24. The DMPM maps the DMPM service to the data link layer service (DL service). DMPM uses services provided by the MAC sub layer of the data link layer as the DL services. The mapping of DMPM and DLL is shown in Table 217 and the PDU MAC destination address mapping is shown in Table 218.

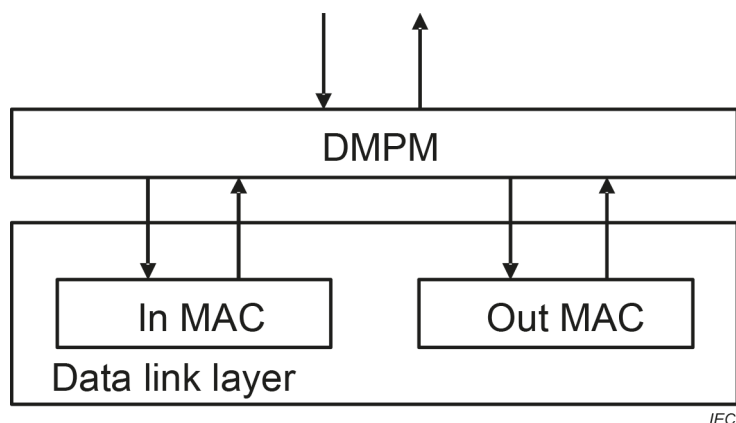


Figure 24 – Structure of type C DMPM

Table 217 – Mapping of type C DMPM service and DL service

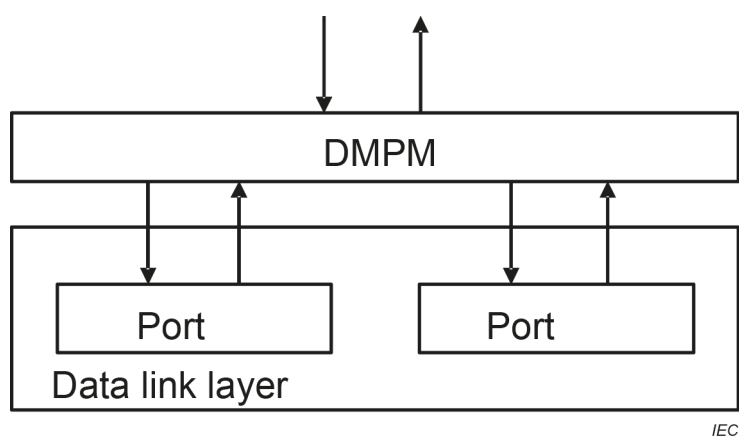
DMPM service	DL service	Remarks
InPort.req(PDU)	MA_DATA.req(destination_address, source_address, frame_check_sequence) in In MAC.	Refer to Table 218 for destination_address. source_address is the MAC address of a node.
InPort.ind(PDU)	MA_DATA.ind(destination_address, source_address, PDU, frame_check_sequence) in In MAC.	Refer to Table 218 for destination_address. source_address is the MAC address of a source.
InPort.ind(Port_State)	Shows the link status of In MAC.	
OutPort.req(PDU)	MA_DATA.req(destination_address, source_address, frame_check_sequence) in Out MAC.	Refer to Table 218 for destination_address. source_address is the MAC address of a node.
OutPort.ind(PDU)	MA_DATA.ind(destination_address, source_address, PDU, frame_check_sequence) in Out MAC.	Refer to Table 218 for destination_address. source_address is the MAC address of a source.
OutPort.ind(Port_State)	Shows the link status of Out MAC.	

Table 218 – Destination address for each type C PDU

Name of PDU	Destination address
Connect-PDU	FF-FF-FF-FF-FF-FF
ConnectAck-PDU	The MAC address of a node to which a response is given (source_address of corresponding Connect-PDU)
Scan-PDU	FF-FF-FF-FF-FF-FF
Collect-PDU	FF-FF-FF-FF-FF-FF
Select-PDU	FF-FF-FF-FF-FF-FF
Launch-PDU	FF-FF-FF-FF-FF-FF
Token-PDU	FF-FF-FF-FF-FF-FF
MyStatus-PDU	FF-FF-FF-FF-FF-FF
Transient1-PDU	dataType == 0:FF-FF-FF-FF-FF-FF
Transient2-PDU	A destination is within in the same network: The MAC address of a destination node A network whose destination address is different: The MAC address of a relaying node
NTNTest-PDU	MAC address of own node.
Dummy-PDU	MAC address of own node.
CyclicDataW-PDU	FF-FF-FF-FF-FF-FF
CyclicDataB-PDU	FF-FF-FF-FF-FF-FF
CyclicDataOut1-PDU	FF-FF-FF-FF-FF-FF
CyclicDataOut2-PDU	FF-FF-FF-FF-FF-FF
CyclicDataIn1-PDU	FF-FF-FF-FF-FF-FF
CyclicDataIn2-PDU	FF-FF-FF-FF-FF-FF

9.2 DMPM type F

The structure of the DMPM is shown in Figure 25. The DMPM maps the DMPM service to the data link layer service (DL service). DMPM uses services provided by the MAC sub layer of the data link layer as the DL services. The mapping of DMPM and DLL is shown in Table 219.



IEC

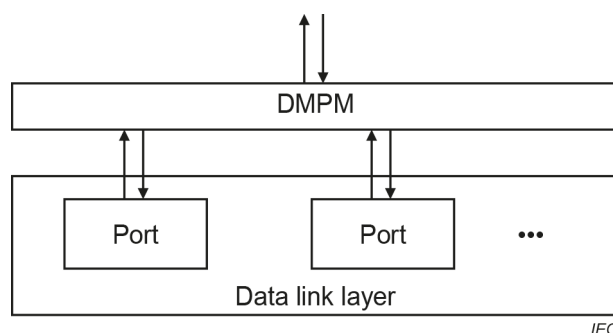
Figure 25 – Structure of type F DMPM

Table 219 – Mapping of type F DMPM service and DL service

DMPM service	DL service	Remarks
Send.req(port, DA, pdu)	MA_DATA.req (destination_address, source_address, mac_service_data_unit, frame_check_sequence) of port	destination_address = source_address is the MAC address of a node.
Receive.ind(port, DA, SA, ftype, pdu)	MA_DATA.ind (destination_address, source_address, mac_service_data_unit, frame_check_sequence) of port	DA = destination_address SA = source_address ftype = mac_service_data_unit. ArFType pdu = mac_service_data_unit

9.3 DMPM type T

The structure of DMPM is shown in Figure 26. DMPM maps the DMPM service to the data link layer service (DL service). DMPM uses the service provided by the MAC sublayer of the data link layer as the DL service. The mapping between DMPM and DLL is shown in Table 220.

**Figure 26 – Structure of type T DMPM****Table 220 – Primitives provided by DMPM**

Primitive Name	Publisher	DL Service	Remarks
Send.req(ports, DA, eType, fType, pdu)	DMPM	MA_DATA.req at a transmittable port(destination_address, source_address, mac_service_Data_unit, frame_check_sequence)	destination_address = DA source_address = SA(MAC address of node) mac_service_Data_unit = pdu
Receive.ind(port, DA, SA, eType, fType, pdu)	Data Link Layer	MA_DATA.ind at a receive port(destination_address, source_address,eType, mac_service_Data_unit, frame_check_sequence)	DA = destination_address SA = source_address eType = mac_service_Data_unit. ethernetType pdu = mac_service_Data_unit

The state of the DMPM state machine is shown in Table 221.

Table 221 – Cyclic Data state

Name	Description
ACTIVE	Operating state

Details of DMPM state machine are shown in Table 222.

Table 222 – DMPM state table

#	Current state	Event /Condition => Action	Next state
1	ACTIVE	send.req() => MAC_DATA.req()	ACTIVE
2	ACTIVE	MAC_DATA.ind() => Receive.ind	ACTIVE
3	ACTIVE	TimerExpired(MACTimer) => ignore	ACTIVE

Bibliography

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

IEC 61784-1 (all parts), *Industrial networks – Profiles – Part 1-x: Fieldbus profiles – CPF x*

IEC 61784-2 (all parts), *Industrial networks – Profiles – Part 2-x: Additional real-time fieldbus profiles based on ISO/IEC/IEEE 8802-3 – CPF x*

SOMMAIRE

AVANT-PROPOS	322
INTRODUCTION.....	324
1 Domaine d'application	325
1.1 Généralités	325
1.2 Spécifications	326
1.3 Conformité	326
2 Références normatives	326
3 Termes, définitions, symboles, abréviations et conventions	327
3.1 Termes et définitions référencés	327
3.1.1 Termes de l'ISO/IEC 7498-1	327
3.1.2 Termes de l'ISO/IEC 8822	327
3.1.3 Termes de l'IEC 61158-1	327
3.2 Termes et définitions supplémentaires du Type 23.....	328
3.3 Symboles et abréviations	330
3.4 Conventions.....	331
3.4.1 Concept général	331
3.4.2 Convention de codage des bits et octets réservés	331
3.4.3 Conventions pour la description de la syntaxe abstraite.....	331
3.4.4 Conventions pour la description de bit en octets	332
3.4.5 Conventions relatives aux descriptions des diagrammes d'états	332
4 Description de la syntaxe FAL	333
4.1 FALPDU type C – syntaxe abstraite	333
4.1.1 Syntaxe abstraite de base	333
4.1.2 Connect-PDU	334
4.1.3 ConnectAck-PDU.....	334
4.1.4 Scan-PDU	334
4.1.5 Collect-PDU.....	334
4.1.6 Select-PDU.....	335
4.1.7 Launch-PDU	335
4.1.8 PDU Token	335
4.1.9 MyStatus-PDU	335
4.1.10 Transient1-PDU	336
4.1.11 Dummy-PDU.....	336
4.1.12 Transient2-PDU	337
4.1.13 NTNTest-PDU.....	337
4.1.14 CyclicDataW-PDU.....	337
4.1.15 CyclicDataB-PDU	338
4.1.16 CyclicDataOut1-PDU	338
4.1.17 CyclicDataOut2-PDU	338
4.1.18 CyclicDataIn1-PDU	339
4.1.19 CyclicDataIn2-PDU	339
4.2 FALPDU type F – syntaxe abstraite	339
4.2.1 Syntaxe abstraite de base	339
4.2.2 Persuasion-PDU	341
4.2.3 TestData-PDU	341
4.2.4 TestDataAck-PDU.....	342
4.2.5 Setup-PDU	342

4.2.6	SetupAck-PDU.....	343
4.2.7	F-Token-PDU	343
4.2.8	F-MyStatus-PDU.....	344
4.2.9	Measure-PDU	344
4.2.10	F-Offset-PDU.....	344
4.2.11	F-Update-PDU	345
4.2.12	F-CyclicData-PDU	345
4.2.13	Transient1-PDU	345
4.2.14	TransientAck-PDU	348
4.2.15	Transient2-PDU	349
4.2.16	ParamCheck-PDU	349
4.2.17	Parameter-PDU	350
4.2.18	Timer-PDU	351
4.3	Affectation des types de données pour le type C.....	351
4.4	Affectation des types de données pour le type F	352
4.5	FALPDU type T – syntaxe abstraite	353
4.5.1	Syntaxe abstraite de base	353
4.5.2	CyclicM-PDU	356
4.5.3	CyclicS-PDU.....	356
4.5.4	CyclicMs-PDU	357
4.5.5	CyclicSs-PDU	357
4.5.6	AcyclicPriority-PDU	358
4.5.7	AcyclicDetection-PDU.....	358
4.5.8	AcyclicDetectionAck-PDU	359
4.5.9	AcyclicTestDataHeader	360
4.5.10	AcyclicTestDataHeader	360
4.5.11	AcyclicData-PDU	361
4.5.12	PtpSync-PDU	361
4.5.13	PtpPdelayReq-PDU	361
4.5.14	PtpPdelayResp-PDU.....	361
4.5.15	PtpFollowUp-PDU.....	361
4.5.16	PtpPdelayRespFollowUp-PDU	361
4.5.17	PtpAnnounce-PDU.....	361
4.5.18	SImpIPAddressSet-PDU	361
4.5.19	SImpNetworkConfigMain-PDU	362
4.5.20	SImpNetworkConfigTslt-PDU	362
4.5.21	SImpNotification-PDU	362
4.5.22	SImpMasterConfig-PDU	362
4.5.23	SImpSlaveConfig-PDU	362
4.5.24	SImpCyclicConfigMain-PDU.....	362
4.5.25	SImpCyclicConfigTrnSubPayload-PDU	363
4.5.26	SImpCyclicConfigRcvSubPayload-PDU.....	363
4.5.27	SImpCyclicConfigRcvSrcInfo-PDU	363
4.5.28	SImpLinkDevicePrmWrite-PDU	363
4.5.29	SImpLinkDevicePrmWriteCheckRequest-PDU.....	363
4.5.30	SImpLinkDevicePrmWriteCheckResponse-PDU	363
4.5.31	SImpNMTStateUpload-PDU	364
4.5.32	SImpNMTStateDownload-PDU.....	364
4.5.33	SImpReadObject-PDU	364

4.5.34	SImpWriteObject-PDU	364
4.5.35	SImpObjectSubIDReadBlock-PDU	364
4.5.36	SImpObjectSubIDWriteBlock-PDU	364
4.5.37	SImpGetODList-PDU	365
4.5.38	SImpGetObjectDescription-PDU	365
4.5.39	SImpGetEntryDescription-PDU	365
4.5.40	SImpStopOwnStationCyclic-PDU	365
4.5.41	SImpStartOwnStationCyclic-PDU	365
4.5.42	SImpStopOtherStationsCyclic-PDU	365
4.5.43	SImpStartOtherStationsCyclic-PDU	366
4.5.44	SImpAllParameterGet-PDU	366
4.5.45	SImpParameterGet-PDU	366
4.5.46	SImpAllParameterSizeGet-PDU	366
4.5.47	SImpParameterSizeGet-PDU	366
4.5.48	SImpStationSubIDListGet-PDU	366
4.5.49	SImpDeviceIdentificationInfoGet-PDU	367
4.5.50	SImpDataMonitoring-PDU	367
4.5.51	SImpAllParameterSet-PDU	367
4.5.52	SImpParameterSet-PDU	367
4.5.53	SImpParameterVersionCheck-PDU	367
4.5.54	SImpDeviceIdentificationInfoCompare-PDU	367
4.5.55	SImpNodeSearch-PDU	368
4.5.56	SImpIPAddressSet-PDU	368
4.5.57	SImpDeviceInfoCompare-PDU	368
4.5.58	SImpParameterGet-PDU	368
4.5.59	SImpParameterSet-PDU	368
4.5.60	SImpParameterSetStart-PDU	368
4.5.61	SImpParameterSetEnd-PDU	369
4.5.62	SImpVerifyCheckCode-PDU	369
4.5.63	SImpOutputMapFileNameGet-PDU	369
4.5.64	SImpNewFile-PDU	369
4.5.65	SImpParameterSetCancel-PDU	369
4.5.66	SImpOpenFile-PDU	369
4.5.67	SImpCloseFile-PDU	370
4.5.68	SImpReadFile-PDU	370
4.5.69	SImpWriteFile-PDU	370
4.5.70	SImpStatusRead-PDU	370
4.5.71	SImpCommunicationSettingGet-PDU	370
4.5.72	SImpGetDeviceInfo-PDU	370
4.5.73	SImpGetBackupListFileName-PDU	371
4.5.74	SImpStartBackup-PDU	371
4.5.75	SImpEndBackup-PDU	371
4.5.76	SImpCheckRestoreVersion-PDU	371
4.5.77	SImpStartRestore-PDU	371
4.5.78	SImpEndRestore-PDU	371
4.5.79	SImpStatusRead2-PDU	372
4.5.80	SImpReqSearchNode-PDU	372
4.5.81	SImpGetSearchNodeState-PDU	372
4.5.82	SImpGetNodeList-PDU	372

4.5.83	SImpReqSetIPAddress-PDU	372
4.5.84	SImpSearchPrmControlStation-PDU	372
4.5.85	SImpRequestRestore-PDU	373
4.5.86	SImpCheckPrmDelivery-PDU	373
4.5.87	SImpRsvStationConfigTemporaryRelease-PDU	373
4.5.88	SImpRsvStationConfig-PDU	373
4.5.89	SImpGetEventNum-PDU	373
4.5.90	SImpGetEventHistory-PDU	374
4.5.91	SImpClearEventHistory-PDU	374
4.5.92	SImpClockOffsetDataSend-PDU	374
4.5.93	SImpSetWatchdogCounterInfo-PDU	374
4.5.94	SImpWatchdogCounterOffsetConfig-PDU	374
4.5.95	SImpRemoteReset-PDU	374
4.5.96	SImpGetCommunicationSet-PDU	375
4.5.97	SImpGetStationSubIDList-PDU	375
4.5.98	SImpGetDeviceInfo-PDU	375
4.5.99	SImpStartBackup-PDU	375
4.5.100	SImpEndBackup-PDU	375
4.5.101	SImpRequestBackup-PDU	375
4.5.102	SImpGetBackupPrm-PDU	376
4.5.103	SImpCheckRestore-PDU	376
4.5.104	SImpStartRestore-PDU	376
4.5.105	SImpEndRestore-PDU	376
4.5.106	SImpSetBackupPrm-PDU	376
4.5.107	SImpLinkupSpeed-PDU	376
4.5.108	SImpNodeIndication-PDU	377
4.6	Affectation des types de données pour le type T	377
5	Syntaxe de transfert FAL	377
5.1	Règles d'encodages	377
5.1.1	Codage de valeur Unsigned	377
5.1.2	Codage de chaîne d'octets	377
5.1.3	Codage de SEQUENCE	377
5.1.4	Codage de LOctetString	377
5.2	FALPDU type C – encodage des éléments	378
5.2.1	FALARHeader	378
5.2.2	Connect-PDU	380
5.2.3	ConnectAck-PDU	380
5.2.4	Scan-PDU	381
5.2.5	Collect-PDU	382
5.2.6	Select-PDU	384
5.2.7	Launch-PDU	385
5.2.8	PDU Token	385
5.2.9	MyStatus-PDU	385
5.2.10	Transient1-PDU	387
5.2.11	Dummy-PDU	392
5.2.12	Transient2-PDU	392
5.2.13	NTNTest-PDU	409
5.2.14	CyclicDataW-PDU	410
5.2.15	CyclicDataB-PDU	411

5.2.16	CyclicDataOut1-PDU	411
5.2.17	CyclicDataOut2-PDU	412
5.2.18	CyclicDataIn1-PDU	413
5.2.19	CyclicDataIn2-PDU	414
5.3	FALPDU type F – encodage des éléments	415
5.3.1	FALARHeader	415
5.3.2	Persuasion-PDU	419
5.3.3	TestData-PDU	419
5.3.4	TestDataAck-PDU	420
5.3.5	Setup-PDU	421
5.3.6	SetupAck-PDU	423
5.3.7	F-Token-PDU	425
5.3.8	F-Measure-PDU	426
5.3.9	F-Offset-PDU	427
5.3.10	F-Update-PDU	427
5.3.11	F-MyStatus-PDU	427
5.3.12	F-CyclicData-PDU	432
5.3.13	Transient1-PDU	434
5.3.14	TransientAck-PDU	438
5.3.15	Transient2-PDU	439
5.3.16	ParamCheck-PDU	442
5.3.17	Parameter-PDU	443
5.3.18	Timer-PDU	451
5.4	FALPDU type T – encodage des éléments	451
5.4.1	CyclicM-PDU	451
5.4.2	CyclicS-PDU	455
5.4.3	CyclicMs-PDU	456
5.4.4	CyclicSs-PDU	458
5.4.5	AcyclicPriority-PDU	459
5.4.6	AcyclicDetection-PDU	460
5.4.7	AcyclicDetectionAck-PDU	462
5.4.8	AcyclicTestData-PDU	468
5.4.9	AcyclicTestDataAck-PDU	471
5.4.10	AcyclicData-PDU	473
5.4.11	Ptp-PDU	473
5.4.12	IpData-PDU	475
6	Structure du diagramme d'états de protocole FAL	475
7	Machine de protocole de service FAL (FSPM)	476
7.1	Vue d'ensemble	476
7.2	FSPM – Type C	476
7.2.1	Vue d'ensemble	476
7.2.2	FSPM	478
7.3	FSPM – Type F	480
7.3.1	Vue d'ensemble	480
7.3.2	FSPM	483
7.4	FSPM – Type T	487
7.4.1	Vue d'ensemble	487
7.4.2	Diagramme d'états FSPM	489
8	Machine de protocole de relation d'application (ARPM)	492

8.1	ARPM – Type C	492
8.1.1	Vue d'ensemble	492
8.1.2	Acyclic transmission (Transmission acyclique).....	493
8.1.3	Transmission cyclique	495
8.1.4	Commande de connexion	500
8.1.5	Common parameter dist.....	538
8.2	ARPM – Type F	543
8.2.1	Vue d'ensemble	543
8.2.2	Acyclic transmission (Transmission acyclique).....	544
8.2.3	Transmission cyclique	547
8.2.4	Channel control (Contrôle de canal).....	550
8.2.5	Parameter dist	589
8.2.6	Synchronous trigger (Déclenchement synchrone)	592
8.2.7	Temporisateur	593
8.2.8	Transmission de mesure	594
8.3	ARPM – Type T	599
8.3.1	Vue d'ensemble	599
8.3.2	Cyclic transmission (Transmission cyclique)	600
8.3.3	Acyclic transmission (Transmission acyclique).....	601
8.3.4	Channel control (Contrôle de canal).....	605
8.3.5	TimeSync Control	607
8.3.6	IPTrans Control	609
8.3.7	Handler.....	610
9	Machine de protocole de mapping DLL (DMPM)	614
9.1	DMPM – Type C.....	614
9.2	DMPM – Type F	616
9.3	DMPM – Type T	616
	Bibliographie.....	618
	Figure 1 – Description de bit en octets	332
	Figure 2 – Structure de réponse à une demande de récupération d'informations sur l'accès à la mémoire	396
	Figure 3 – Définitions des attributs.....	397
	Figure 4 – Définitions des codes d'accès	397
	Figure 5 – Structure en cas de demande RUN	399
	Figure 6 – Structure en cas de réponse à une demande RUN	400
	Figure 7 – Structure en cas de demande STOP	400
	Figure 8 – Structure en cas de réponse à une demande STOP	401
	Figure 9 – Structure en cas de demande de lecture de mémoire par lots	402
	Figure 10 – Structure en cas de réponse à une demande de lecture de mémoire par lots ...	403
	Figure 11 – Structure en cas de demande de lecture de mémoire aléatoire.....	404
	Figure 12 – Structure en cas de réponse à une demande de lecture de mémoire aléatoire	405
	Figure 13 – Structure en cas de demande d'écriture mémoire par lots	406
	Figure 14 – Structure en cas de réponse à une demande d'écriture mémoire par lots	407
	Figure 15 – Structure en cas de demande d'écriture mémoire aléatoire	408
	Figure 16 – Structure en cas de réponse à une demande d'écriture mémoire aléatoire	409

Figure 17 – Relations entre les machines de protocole	476
Figure 18 – Structure du type C de FSPM	477
Figure 19 – Structure du type F de FSPM	481
Figure 20 – Structure du type T de FSPM	488
Figure 21 – Structure du type C de ARPM.....	493
Figure 22 – Structure du type F de ARPM.....	544
Figure 23 – Structure du type T d'ARPM.....	599
Figure 24 – Structure du type C de DMPM	614
Figure 25 – Structure du type F de DMPM	616
Figure 26 – Structure du type T de DMPM	617
Tableau 1 – Éléments de la description d'un diagramme d'états	332
Tableau 2 – Description des éléments d'un diagramme d'états	332
Tableau 3 – Conventions utilisées dans les diagrammes d'états	333
Tableau 4 – afFType	378
Tableau 5 – priority	379
Tableau 6 – portChoice	380
Tableau 7 – portCheckResult.....	381
Tableau 8 – dstPortInfo.....	381
Tableau 9 – scanState	381
Tableau 10 – nodeType	382
Tableau 11 – loopState	383
Tableau 12 – État cyclique.....	383
Tableau 13 – Mode paramétrage	383
Tableau 14 – opState.....	386
Tableau 15 – errorState	387
Tableau 16 – Data type.....	388
Tableau 17 – CPW.....	389
Tableau 18 – CPWC	389
Tableau 19 – CPWCR	389
Tableau 20 – cmParam	389
Tableau 21 – Détails de param area	390
Tableau 22 – Détails des paramètres d'application	391
Tableau 23 – Détails de LB/LW CM area et de LB/LW CM additional area	391
Tableau 24 – Détails de LX/LY CM 1 area et de LX/LY CM 2 area	391
Tableau 25 – Indicateur de module de destination	393
Tableau 26 – Types de commandes.....	394
Tableau 27 – Codes d'accès à la mémoire du module de réseau	398
Tableau 28 – Codes d'accès à la mémoire du contrôleur	398
Tableau 29 – byteValidity.....	410
Tableau 30 – afFType	415
Tableau 31 – dataType	416
Tableau 32 – varField	417

Tableau 33 – nodeType	417
Tableau 34 – ProtocolVerType.....	418
Tableau 35 – Etat de liaison.....	421
Tableau 36 – Spécification d'activation/de désactivation de port	422
Tableau 37 – État de maintien du paramètre de transmission cyclique.....	429
Tableau 38 – État de fonctionnement d'application détaillé	430
Tableau 39 – État de détection d'erreur	430
Tableau 40 – État de réception d'événement spécifique à une station subordonnée	432
Tableau 41 – dataSupType de dataType (0x07)	434
Tableau 42 – FieldSpecificTransient opHeader	435
Tableau 43 – command (dataType: 0x07, dataSubType: 0x0002)	435
Tableau 44 – Type subCommand pour chaque type de commande	436
Tableau 45 – Structure de la "Remise des informations relatives au nœud"	436
Tableau 46 – Structure de la remise des informations relatives au nœud – message	436
Tableau 47 – Structure de la réponse à une demande d'obtention d'informations statistiques (Get statistical information response)	437
Tableau 48 – Structure de la réponse à une demande d'acquisition des détails du nœud (Acquisition of node details response).....	438
Tableau 49 – Spécification du module d'exécution	440
Tableau 50 – Type de commande	441
Tableau 51 – frameType	451
Tableau 52 – cycleNo	452
Tableau 53 – sa	452
Tableau 54 – da	452
Tableau 55 – commInfo	453
Tableau 56 – txAsynInfo	453
Tableau 57 – seqNo.....	453
Tableau 58 – L'octet supérieur de diagnosticData	454
Tableau 59 – Les trois octets inférieurs de diagnosisData.....	454
Tableau 60 – L'octet supérieur de diagnosticData	455
Tableau 61 – Les trois octets inférieurs de diagnosisData.....	456
Tableau 62 – mngPriority	459
Tableau 63 – KindFlag	460
Tableau 64 – previousNodePort.....	461
Tableau 65 – optionFlag	461
Tableau 66 – sendInfo	462
Tableau 67 – nodeType	462
Tableau 68 – 4e octet de l'adresse IP	463
Tableau 69 – detectionRcvPort	463
Tableau 70 – myPort.....	463
Tableau 71 – Quatre bits de myPortLinkStatus	464
Tableau 72 – Quatre bits de myPortFilterStatus	464
Tableau 73 – Performance	465
Tableau 74 – gmPriority	465

Tableau 75 – syncType	465
Tableau 76 – cyclicSize	467
Tableau 77 – fonction	467
Tableau 78 – optionInfo	468
Tableau 79 – Version de protocole.....	469
Tableau 80 – Type de protocole.....	469
Tableau 81 – Informations relatives à la source	470
Tableau 82 – Etat de liaison.....	473
Tableau 83 – Table d'états des données cycliques	478
Tableau 84 – Table d'états des données acycliques	478
Tableau 85 – Table d'états de gestion	480
Tableau 86 – Table d'états des données cycliques	483
Tableau 87 – Table d'états des données acycliques	484
Tableau 88 – Table d'états de gestion	487
Tableau 89 – Table d'états de Synchronization (synchronisation)	487
Tableau 90 – Table d'états de Measurement (mesure).....	487
Tableau 91 – Primitives fournies par la machine FSPM.....	488
Tableau 92 – Etat des données cycliques (Cyclic Data)	489
Tableau 93 – Table d'états des données cycliques	489
Tableau 94 – Fonction utilisée pour les données cycliques	489
Tableau 95 – Etat des données acycliques (Acyclic data)	490
Tableau 96 – Table d'états des données acycliques	490
Tableau 97 – Etat de la gestion (Management)	491
Tableau 98 – Table d'états de gestion	491
Tableau 99 – Etat des données de synchronisation temporelle (TimeSync Data)	491
Tableau 100 – Table d'états des données de synchronisation temporelle.....	492
Tableau 101 – Etat des données SLMP (SLMP Data)	492
Tableau 102 – Table d'états des données SLMP.....	492
Tableau 103 – Table d'états de la transmission acyclique	494
Tableau 104 – Fonctions de transmission acyclique.....	495
Tableau 105 – Table d'états de la transmission cyclique (Cyclic transmission).....	495
Tableau 106 – Fonctions de transmission cyclique (Cyclic transmission)	499
Tableau 107 – Diagramme d'états de contrôle de connexion (Connection control) – Initial..	500
Tableau 108 – Diagramme d'états de contrôle de connexion (Connection control) – Connect	501
Tableau 109 – Diagramme d'états de contrôle de connexion (Connection control) – Scan	503
Tableau 110 – Diagramme d'états de contrôle de connexion (Connection control) – ScanWait	506
Tableau 111 – Diagramme d'états de contrôle de connexion (Connection control) – Collect	509
Tableau 112 – Diagramme d'états de contrôle de connexion (Connection control) – CollectWait	512
Tableau 113 – Diagramme d'états de contrôle de connexion (Connection control) – Select	515

Tableau 114 – Diagramme d'états de contrôle de connexion (Connection control) – TokenStartWait	518
Tableau 115 – Diagramme d'états de contrôle de connexion (Connection control) – LaunchWait	521
Tableau 116 – Diagramme d'états de contrôle de connexion (Connection control) – TokenReleaseWait	524
Tableau 117 – Diagramme d'états de contrôle de connexion (Connection control) – TokenReleased	527
Tableau 118 – Diagramme d'états de contrôle de connexion (Connection control) – TokenWait	532
Tableau 119 – Diagramme d'états de contrôle de connexion (Connection control) – NTNTestMaster	537
Tableau 120 – Diagramme d'états de contrôle de connexion (Connection control) – NTNTestSlave	538
Tableau 121 – Liste des fonctions de Connection control (contrôle de connexion)	538
Tableau 122 – Table d'états de Common parameter dist (distribution de paramètres communs)	538
Tableau 123 – Liste des fonctions de Connection control (contrôle de connexion)	542
Tableau 124 – Mapping du service interne (internal service) et du service de transmission acyclique (acyclic transmission service)	543
Tableau 125 – États de la transmission acyclique (Acyclic transmission)	544
Tableau 126 – Table d'états de la transmission acyclique	545
Tableau 127 – Fonctions de transmission acyclique	546
Tableau 128 – Variables de transmission acyclique	547
Tableau 129 – États de la transmission cyclique (Cyclic transmission)	547
Tableau 130 – Table d'états de la transmission cyclique	548
Tableau 131 – Fonctions de transmission cyclique (Cyclic transmission)	550
Tableau 132 – Variables de la transmission cyclique (Cyclic transmission)	550
Tableau 133 – États du contrôle de canal de la station principale	550
Tableau 134 – États du contrôle de canal de la station subordonnée	551
Tableau 135 – Table d'états de station principale – MasterDown	551
Tableau 136 – Table d'états de station principale – Listen	551
Tableau 137 – Table d'états de station principale – MasterArbitration	553
Tableau 138 – Table d'états de station principale – PrimaryMasterScatterTD	554
Tableau 139 – Table d'états de station principale – PrimaryMasterSettingUp	556
Tableau 140 – Table d'états de station principale – PrimaryMasterHoldToken	559
Tableau 141 – Table d'états de station principale – PrimaryMasterSolicitToken	562
Tableau 142 – Table d'états de station principale – PrimaryMasterInviting	565
Tableau 143 – Table d'états de station mprincipaleaître – MasterWaitTD	567
Tableau 144 – Table d'états de station principale – MasterWaitSetup	568
Tableau 145 – Table d'états de station principale – MasterSolicitToken (sans mesure de retard du chemin de transmission)	569
Tableau 146 – Table d'états de station principale – MasterSolicitToken (avec mesure de retard du chemin de transmission)	571
Tableau 147 – Table d'états de station principale – MasterHoldToken	573
Tableau 148 – Table d'états de station principale – MasterMeasurement (sans fonction de mesure de retard du chemin de transmission)	576

Tableau 149 – Table d'états de station principale – MasterMeasurement (avec fonction de mesure de retard du chemin de transmission)	576
Tableau 150 – Table d'états de station subordonnée – SlaveDown	577
Tableau 151 – Table d'états de station subordonnée – SlaveWaitTD	577
Tableau 152 – Table d'états de station subordonnée – SlaveWaitSetup	578
Tableau 153 – Table d'états de station subordonnée – SlaveSolicitToken (sans mesure de retard du chemin de transmission)	579
Tableau 154 – Table d'états de station subordonnée – SlaveSolicitToken (avec mesure de retard du chemin de transmission)	581
Tableau 155 – Table d'états de station subordonnée – SlaveHoldToken	583
Tableau 156 – Fonctions de contrôle de canal de station principale	586
Tableau 157 – Fonctions de contrôle de canal de la station subordonnée	587
Tableau 158 – Variables de contrôle de canal de station principale	587
Tableau 159 – Variables de contrôle de canal de la station subordonnée	588
Tableau 160 – Temporisateurs de contrôle de canal de station principale	588
Tableau 161 – Temporisateurs de contrôle de canal de la station subordonnée	588
Tableau 162 – États de la distribution des paramètres de la station principale	589
Tableau 163 – États de la distribution des paramètres de la station subordonnée	589
Tableau 164 – Table d'états de la distribution des paramètres de la station principale	589
Tableau 165 – Table d'états de la distribution des paramètres de la station subordonnée	590
Tableau 166 – Fonctions de la distribution des paramètres de la station principale	591
Tableau 167 – Fonctions de la distribution des paramètres de la station subordonnée	592
Tableau 168 – États du déclencheur synchrone (Synchronous trigger) de la station principale	592
Tableau 169 – États du déclencheur synchrone (Synchronous trigger) de la station subordonnée	592
Tableau 170 – Table d'états du déclencheur synchrone (Synchronous trigger) de la station principale	592
Tableau 171 – Table d'états du déclencheur synchrone (Synchronous trigger) de la station subordonnée	593
Tableau 172 – Fonctions du déclencheur synchrone (Synchronous trigger)	593
Tableau 173 – États du temporisateur – Type "au mieux" (Best effort)	593
Tableau 174 – Tableau – États du temporisateur – Type "cycle fixe" (Fixed cycle)	593
Tableau 175 – États du temporisateur – Type "au mieux"	593
Tableau 176 – Table d'états du temporisateur – Type "cycle fixe"	594
Tableau 177 – Variables du temporisateur	594
Tableau 178 – Temporisateur à cycle fixe	594
Tableau 179 – États de la transmission de mesure (Measure transmission) de la station principale	595
Tableau 180 – États de la transmission de mesure (Measure transmission) de la station subordonnée	595
Tableau 181 – Table d'états de la transmission de mesure (Measure transmission) de la station principale	595
Tableau 182 – Table d'états de la transmission de mesure (Measure transmission) de la station subordonnée	596

Tableau 183 – Fonctions de la transmission de mesure (Measure transmission) de la station principale	597
Tableau 184 – Fonctions de la transmission de mesure (Measure transmission) de la station subordonnée	598
Tableau 185 – Variables de la transmission de mesure (Measure transmission) de la station principale	599
Tableau 186 – Primitives fournies par la transmission cyclique	600
Tableau 187 – Etat de la transmission cyclique (Cyclic transmission)	600
Tableau 188 – Table d'états de la transmission cyclique	600
Tableau 189 – Fonctions utilisées pour la transmission cyclique	601
Tableau 190 – Variables utilisées dans la transmission cyclique	601
Tableau 191 – Primitives fournies par la transmission acyclique	601
Tableau 192 – Etats de la transmission acyclique (Acyclic transmission)	601
Tableau 193 – Table d'états de la transmission acyclique	602
Tableau 194 – Fonctions utilisées pour la transmission acyclique	604
Tableau 195 – Variables utilisées dans la transmission acyclique	604
Tableau 196 – Primitives fournies par le contrôle de canal.....	605
Tableau 197 – Etat de contrôle de canal (Channel Control)	605
Tableau 198 – Diagramme d'états du contrôle de canal	605
Tableau 199 – Fonctions utilisées pour le contrôle de canal.....	607
Tableau 200 – Variables utilisées dans le contrôle de canal.....	607
Tableau 201 – Primitives fournies par TimeSync Control	607
Tableau 202 – Etat de TimeSync Control	608
Tableau 203 – Table d'états de TimeSync Control	608
Tableau 204 – Fonctions utilisées pour TimeSync Control	608
Tableau 205 – Variables utilisées dans TimeSync Control	609
Tableau 206 – Primitives fournies par IPTrans Control.....	609
Tableau 207 – Etat d'IPTrans Control dans la station principale.....	609
Tableau 208 – Table d'états d'IPTrans Control	609
Tableau 209 – Fonctions utilisées pour IPTrans Control.....	610
Tableau 210 – Variables utilisées dans IPTrans Control.....	610
Tableau 211 – Etat du Handler	610
Tableau 212 – Diagramme d'états DOWN	611
Tableau 213 – Diagramme d'états INIT	611
Tableau 214 – Diagramme d'états RUN	612
Tableau 215 – Fonctions utilisées pour le Handler	613
Tableau 216 – Variables utilisées pour le Handler.....	614
Tableau 217 – Mapping du type C de service DMPM et de service DL	615
Tableau 218 – Adresse de destination de chaque PDU de type C	615
Tableau 219 – Mapping du type F de service DMPM et de service DL	616
Tableau 220 – Primitives fournies par DMPM.....	617
Tableau 221 – Etat des données cycliques (Cyclic Data)	617
Tableau 222 – Table d'états de DMPM	617

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-23: Spécification du protocole de la couche application – Éléments de type 23

AVANT-PROPOS

- 1) La Commission Électrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. À cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
- 2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, les Comités nationaux de l'IEC intéressés étant représentés dans chaque comité d'études.
- 3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
- 4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
- 5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
- 6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
- 7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
- 8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
- 9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets.

L'attention est attirée sur le fait que l'utilisation du type de protocole associé est restreinte par les détenteurs des droits de propriété intellectuelle. En tout état de cause, l'engagement de renonciation partielle aux droits de propriété intellectuelle pris par les détenteurs de ces droits autorise l'utilisation d'un type de protocole de couche avec les autres protocoles de couche du même type, ou dans des combinaisons avec d'autres types autorisées explicitement par les détenteurs des droits de propriété intellectuelle pour ce type.

NOTE Les combinaisons de types de protocoles sont spécifiées dans la série IEC 61784-1 et dans la série IEC 61784-2.

L'IEC 61158-6-23 a été établie par le sous-comité 65C: Réseaux industriels, du comité technique 65 de l'IEC: Mesure, commande et automation dans les processus industriels. Il s'agit d'une Norme internationale.

Cette troisième édition annule et remplace la deuxième édition publiée en 2019. Cette édition constitue une révision technique.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

- a) ajout du mode étendu de transmission et de l'attribut associé (voir 3.2.28, 4.1.9, 4.4, 5.2.9.2 et 5.3);
- b) mise à jour du Tableau 4, du Tableau 5, du Tableau 16 et du Tableau 48.

La présente version bilingue (2023-11) correspond à la version anglaise monolingue publiée en 2023-03.

La version française de cette norme n'a pas été soumise au vote.

La langue employée pour l'élaboration de cette Norme internationale est l'anglais.

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2, il a été développé selon les Directives ISO/IEC, Partie 1 et les Directives ISO/IEC, Supplément IEC, disponibles sous www.iec.ch/members_experts/refdocs. Les principaux types de documents développés par l'IEC sont décrits plus en détail sous www.iec.ch/publications.

Une liste de toutes les parties de la série IEC 61158, publiées sous le titre général *Réseaux de communication industriels – Spécifications des bus de terrain*, peut être consultée sur le site web de l'IEC.

Le comité a décidé que le contenu de ce document ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous webstore.iec.ch dans les données relatives au document recherché. À cette date, le document sera

- reconduit,
- supprimé,
- remplacé par une édition révisée, ou
- amendé.

INTRODUCTION

Le présent document fait partie d'une série élaborée pour faciliter l'interconnexion des composants de systèmes d'automatisation. Il est lié aux autres normes de la série telle que définie par le modèle de référence de bus de terrain "à trois couches" décrit dans l'IEC 61158-1.

Le protocole d'application fournit le service d'application au moyen des services disponibles au niveau de la couche Liaison de données ou de la couche immédiatement inférieure. Le principal objectif du présent document est de définir un ensemble de règles de communication, exprimées en matière de procédures qu'ont à suivre les entités d'application (AE, Application Entity) homologues au moment de la communication. Ces règles de communication sont destinées à fournir une base de développement solide de façon à atteindre plusieurs objectifs:

- servir de guide aux intégrateurs et aux concepteurs;
- être appliquées dans le cadre des essais et de l'achat d'équipements;
- être incorporées dans un accord sur l'accès des systèmes à l'environnement de systèmes ouverts;
- affiner la compréhension des communications à temps critique au sein de l'OSI.

Le présent document traite, en particulier, de la communication et de l'interfonctionnement des capteurs, effecteurs et autres appareils d'automatisation. L'utilisation conjointe du présent document avec d'autres normes entrant dans les modèles de référence OSI ou de bus de terrain permet à des systèmes qui ne pourraient pas, sans cela, fonctionner ensemble dans toute combinaison.

La Commission Électrotechnique Internationale (IEC) attire l'attention sur le fait qu'il est déclaré que la conformité avec les dispositions du présent document peut impliquer l'utilisation d'un brevet. L'IEC ne prend pas position quant à la preuve, à la validité et à la portée de ces droits de propriété.

Le détenteur de ces droits de propriété a donné l'assurance à l'IEC qu'il consent à négocier des licences avec des demandeurs du monde entier à des termes et conditions raisonnables et non discriminatoires. À ce propos, la déclaration du détenteur des droits de propriété est enregistrée à l'IEC. Des informations peuvent être obtenues dans la base de données des droits de propriété, disponible à l'adresse suivante: <http://patents.iec.ch>.

L'attention est attirée sur le fait que certains des éléments du présent document peuvent faire l'objet de droits de propriété autres que ceux figurant dans la base de données des brevets. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-23: Spécification du protocole de la couche application – Eléments de type 23

1 Domaine d'application

1.1 Généralités

La couche application de bus de terrain (FAL, Fieldbus Application Layer) fournit aux programmes d'utilisateur un moyen d'accéder à l'environnement de communication du bus de terrain. À cet égard, la FAL peut être vue comme une "fenêtre entre des programmes d'application correspondants".

La présente partie de l'IEC 61158 fournit des éléments communs pour les communications à temps critique ou non entre des programmes d'application dans un environnement et avec un matériel d'automatisation spécifiques aux bus de terrain de type 23. Le terme "à temps critique" est utilisé pour indiquer la présence d'une fenêtre temporelle, dans laquelle il est exigé de réaliser une ou plusieurs actions spécifiées selon un niveau défini de certitude. Tout manquement à réaliser ces actions dans la fenêtre de temps prévue risque de provoquer la défaillance des applications qui les demandent, avec le risque de mettre en danger l'équipement, l'usine, voire les personnes.

Le présent document définit de manière abstraite le comportement visible de l'extérieur fourni par les différents types de la couche application de bus de terrain en ce qui concerne:

- a) la syntaxe abstraite définissant les unités de données du protocole de la couche application transmises entre les entités d'application de communication;
- b) la syntaxe de transfert définissant les unités de données du protocole de la couche application transmises entre les entités d'application de communication;
- c) les diagrammes d'états de contexte d'application définissant le comportement de service d'application observable entre les entités d'application en communication; et
- d) les diagrammes d'états de relations entre applications définissant le comportement de communication visible entre les entités d'application en communication.

Le but de la présente norme est de définir le protocole fourni à:

- a) définir la représentation filaire des primitives de service définies dans l'IEC 61158-5-23; et
- b) définir le comportement visible de l'extérieur associé à leur transfert.

Le présent document spécifie le protocole de la couche Application de bus de terrain de l'IEC, en conformité avec le modèle de référence de base de l'OSI (ISO/IEC 7498) et avec la structure de la couche Application OSI (ISO/IEC 9545).

Les services et protocoles FAL sont fournis par des entités d'application FAL présentes dans les processus d'application. Une entité AE de couche FAL se compose d'un ensemble d'éléments de service d'application (ASE) orientés objet et d'une entité de gestion de couche (LME) qui gère l'entité AE. Les éléments de service d'application proposent des services de communication opérant sur un ensemble de classes d'objets de processus d'application (APO) connexes. L'un des éléments de service d'application FAL est un ASE de gestion qui fournit un ensemble commun de services de gestion des instances de classes FAL.

Bien que ces services spécifient, du point de vue des applications, la manière dont les demandes et les réponses sont données et délivrées, ils ne précisent pas ce qu'il est nécessaire que les applications demandeuses et répondantes en fassent. En d'autres termes, les aspects liés au comportement des applications ne sont pas précisés; seule est spécifiée une définition des demandes et réponses qu'elles peuvent envoyer/recevoir. Cela permet d'assurer une plus grande souplesse aux utilisateurs FAL quant à la normalisation du comportement de ce type d'objet. Outre ces services, le présent document définit également certains services de soutien donnant accès à la couche FAL dans un but de commande de certains aspects de son fonctionnement.

1.2 Spécifications

Le présent document a pour objectif principal de spécifier la syntaxe et le comportement du protocole de la couche application qui véhicule les services de la couche application définis dans l'IEC 61158-5-23.

Un objectif secondaire est de fournir des trajets de migration à partir de protocoles de communication industriels préexistants. Ce dernier objectif explique la diversité des protocoles normalisés dans les sous-parties de la série IEC 61158-6.

1.3 Conformité

Le présent document ne spécifie aucune mise en œuvre ou aucun produit individuel, de même qu'il ne restreint nullement les mises en œuvre des entités de couche application dans les systèmes d'automatisation industriels.

Il n'est pas défini de conformité d'équipement à la présente norme de définition des services de la couche application. A la place, la conformité est obtenue par la mise en œuvre de cette spécification de protocole de couche application.

2 Références normatives

Les documents suivants sont cités dans le texte de sorte qu'ils constituent, pour tout ou partie de leur contenu, des exigences du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

NOTE Toutes les parties de la série IEC 61158, ainsi que la série IEC 61784-1 et la série IEC 61784-2 font l'objet d'une maintenance simultanée. Les références croisées à ces documents dans le texte se rapportent par conséquent aux éditions datées dans la présente liste de références normatives.

IEC 61158-1:2023, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 1: Vue d'ensemble et recommandations pour les séries IEC 61158 et IEC 61784*

IEC 61158-5-23:2023, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-23: Définition des services de la couche application – Eléments de type 23*

ISO/IEC 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Le modèle de base*

ISO/IEC 8822, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Définition du service de présentation*

ISO/IEC 8824-1, *Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1) – Partie 1: Spécification de la notation de base*

ISO/IEC 9545, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Structure de la couche application*

ISO/IEC 10731, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base – Conventions pour la définition des services OSI*

IEEE Std 802.1AS, *Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks* (disponible en anglais seulement)

IEEE Std 1588, *Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems* (disponible en anglais seulement)

IETF RFC 768, J. Postel, *User Datagram Protocol*, août 1980, disponible à l'adresse <https://www.rfc-editor.org/info/rfc768> [consulté le 18/02/2022].

IETF RFC 791, J. Postel, *Internet Protocol*, septembre 1981, disponible à l'adresse <https://www.rfc-editor.org/info/rfc791> [consulté le 18/02/2022].

3 Termes, définitions, symboles, abréviations et conventions

Pour les besoins du présent document, les termes, définitions, symboles, termes abrégés et conventions suivants s'appliquent.

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse <https://www.electropedia.org/>
- ISO Online browsing platform: disponible à l'adresse <https://www.iso.org/obp>

3.1 Termes et définitions référencés

3.1.1 Termes de l'ISO/IEC 7498-1

Pour les besoins du présent document, les termes suivants donnés dans l'ISO/IEC 7498-1 s'appliquent:

- a) entité d'application,
- b) processus d'application,
- c) unité de données de protocole d'application,
- d) élément de service d'application,
- e) syntaxe de transfert

3.1.2 Termes de l'ISO/IEC 8822

Pour les besoins du présent document, les termes suivants définis dans l'ISO/IEC 8822 s'appliquent:

- a) syntaxe abstraite.

3.1.3 Termes de l'IEC 61158-1

Pour les besoins du présent document, les termes suivants donnés dans l'IEC 61158-1 s'appliquent:

- a) machine de protocole de mapping DLL,
- b) couche application de bus de terrain,
- c) Machine de protocole de service FAL,
- d) unité de données de protocole.

3.2 Termes et définitions supplémentaires du Type 23

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

3.2.1

transmission cyclique

transmission qui est exécutée périodiquement et sert à mettre à jour l'appareil de liaison

3.2.2

station d'appareil intelligent

nœud capable d'effectuer la transmission cyclique et la transmission transitoire de données de bits 1:n et de données de mots avec la station principale, et la transmission transitoire avec les stations subordonnées, à l'exclusion des stations d'E/S distantes et ayant des fonctions de client et des fonctions de serveur durant la transmission transitoire

3.2.3

bit de liaison

données de bit de relais de liaison, qui sont partagées par tous les nœuds au moyen de la transmission cyclique et qui sont utilisées en tant que mémoire partagée d'unité un bit du type n:n

3.2.4

appareil de liaison

bit de liaison, mot de liaison, liaison x et liaison y ou RX, RY, RWr et RWw

3.2.5

mot de liaison

données d'unité deux octets de registre de liaison, qui sont partagées par tous les nœuds au moyen de la transmission cyclique et qui sont utilisées en tant que mémoire partagée d'unité deux octets du type n:n

3.2.6

liaison x

données de bit reçues sur une entrée de liaison, qui sont émises depuis chaque nœud au moyen de la transmission cyclique et qui sont utilisées en tant que mémoire partagée d'entrée du type 1:n

3.2.7

liaison y

données de bit de sortie de liaison, qui sont envoyées à chaque nœud au moyen de la transmission cyclique et qui sont utilisées en tant que mémoire partagée de sortie du type 1:n

3.2.8

station locale

nœud capable d'effectuer la transmission cyclique et la transmission transitoire de données de bits n:n et de données de mots avec la station principale et d'autres stations locales, et la transmission transitoire avec les stations subordonnées, à l'exclusion des stations d'E/S distantes et ayant des fonctions de serveur et des fonctions de client durant la transmission transitoire

3.2.9

nœud de gestion

nœud dans lequel des paramètres sont définis

3.2.10

station principale

nœud qui possède des informations de commande (paramètres) et qui gère la transmission cyclique

3.2.11**nœud**

élément qui forme un réseau et qui effectue l'émission, la réception et le transfert de données

3.2.12**essai de nœud à nœud**

essai de couche physique entre deux nœuds

3.2.13**nœud normal**

nœud autre qu'un nœud de gestion

3.2.14**station de dispositif distante**

nœud capable d'effectuer la transmission cyclique et la transmission transitoire de données de bits 1:n et de données de mots avec la station principale, et la transmission transitoire avec les stations subordonnées, à l'exclusion des stations d'E/S distantes et ayant des fonctions de serveur durant la transmission transitoire

3.2.15**station d'E/S distante**

nœud capable d'effectuer la transmission cyclique de données de bits 1:n avec la station principale

3.2.16**nœud de réserve**

nœud qui n'est pas encore connecté, mais qui est compté dans le nombre de nœuds total du réseau; il n'effectue pas de transmission cyclique mais est toujours considéré comme normal par les applications

3.2.17**RX**

entrée distante vue de la station principale avec des données de bits qui sont mises à jour périodiquement par transmission cyclique, de l'appareil subordonné vers l'appareil principal; ou dans une station locale vue de la station principale, il s'agit de la RY de la station locale

3.2.18**RY**

sortie distante vue de la station principale avec des données de bits qui sont mises à jour périodiquement par transmission cyclique, de l'appareil principal vers la station subordonnée; ou dans une station locale vue de la station principale, il s'agit de la RX de la station locale

3.2.19**RWr**

registre distant (entrée) vu de la station principale avec des données de mots qui sont mises à jour périodiquement par transmission cyclique, de l'appareil subordonné vers l'appareil principal; ou dans une station locale vue de la station principale, il s'agit du RWw de la station locale

3.2.20**RWw**

registre distant (sortie) vu de la station principale avec des données de mots qui sont mises à jour périodiquement par transmission cyclique, de l'appareil principal vers la station subordonnée; ou dans une station locale vue de la station principale, il s'agit du RWw de la station locale

3.2.21**station subordonnée**

nœud autre que la station principale

3.2.22

station

nœud d'un réseau

3.2.23

gestionnaire de synchronisation

nœud unique dans un rôle de station principal par réseau qui gère la synchronisation, en distribuant la synchronisation temporelle aux autres nœuds

3.2.24

transmission transitoire

transmission qui est exécutée lors de chaque demande

3.2.25

fonction de client de transmission transitoire

fonction qui émet une demande transitoire

3.2.26

fonction de serveur de transmission transitoire

fonction qui reçoit une demande transitoire et émet une réponse

3.2.27

gestionnaire de commande de transmission

nœud unique dans un rôle de station principale par réseau qui accomplit la gestion du passage de jeton

3.2.28

mode étendu de transmission

mode de fonctionnement étendant le nombre de bits de liaison et de mots de liaison

3.2.29

mot

unité représentant des données, de longueur 16 bits

3.3 Symboles et abréviations

AE	Application Entity (Entité d'application)
AL	Application Layer (Couche application)
AP	Application Process (Processus d'application)
APDU	Application Protocol Data Unit (Unité de données de protocole d'application)
APO	Application Process Object (objet de processus d'application)
AR	Application Relationship (Relation d'application)
AREP	Application Relationship Endpoint (Point d'extrémité de relation entre applications)
ASE	Application Service Element (Elément de service d'application)
ASN.1	Abstract Syntax Notation 1 (Notation de syntaxe abstraite numéro 1)
CRC	Cyclic Redundancy Check (contrôle de redondance cyclique)
DLL	Data-Link Layer (couche liaison de données)
DMPM	DLL Mapping Protocol Machine (Machine de protocole de mapping DLL)
FAL	Fieldbus Application Layer (Couche application de bus de terrain)
FSPM	FAL Service Protocol Machine (Machine de protocole de service FAL)
LB	Link Bit (Bit de liaison)
LSB	Least Significant Bit (bit de poids faible)
LW	Link Word (Mot de liaison)

LX	Link X (Liaison X)
LY	Link Y (Liaison Y)
MSB	Most Significant Bit (bit de poids fort)
OSI	Open Systems Interconnection (Interconnexion de systèmes ouverts)
PDU	Protocol Data Unit (Unité de données de protocole)

3.4 Conventions

3.4.1 Concept général

La couche application de bus de terrain (FAL) se compose d'un ensemble d'éléments ASE orientés objet. Chaque ASE est spécifié dans un paragraphe distinct. Chaque spécification d'élément ASE est divisée en trois parties: ses définitions de classe, ses services et la spécification de ses protocoles. Les deux premières se trouvent dans l'IEC 61158-5-23. La spécification de protocole de chaque ASE est définie dans le présent document.

Les définitions de classe définissent les attributs des classes prises en charge par chaque élément ASE. Les attributs sont accessibles à partir d'instances de la classe qui utilise les services d'ASE de gestion spécifiés dans l'IEC 61158-5-23. La spécification de service définit les services fournis par l'ASE.

Le présent document utilise les conventions descriptives données dans l'ISO/IEC 10731.

3.4.2 Convention de codage des bits et octets réservés

Il est admis d'utiliser le terme "Reserved" (réservé) pour décrire des bits compris dans des octets ou des octets entiers. Il convient que tous les bits ou octets qui sont réservés soient mis à zéro du côté envoi et ils ne doivent pas être soumis à l'essai du côté réception, sauf si cela est énoncé de façon explicite ou si les bits ou octets réservés sont vérifiés par un diagramme d'états.

Le terme "reserved" (réservé) peut aussi être utilisé pour indiquer que certaines valeurs dans la plage d'un paramètre sont réservées pour des extensions futures. Dans ce cas, il convient de ne pas utiliser les valeurs réservées du côté d'envoi et elles ne doivent pas être soumises à l'essai du côté réception, sauf si cela est énoncé de façon explicite ou si les valeurs réservées sont vérifiées par un diagramme d'états.

3.4.3 Conventions pour la description de la syntaxe abstraite

Cette description de la FAL de type 23 utilise un sous-ensemble de l'ASN.1, conformément à l'ISO/IEC 8824-1. Les structures suivantes sont utilisées.

Type sélectif (CHOICE) – Représente une sélection parmi les types de candidats

Type de séquence (SEQUENCE) – Représente une liste à ordre fixe comme dans l'exemple suivant:

```

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

```

NOTE Cet exemple indique que la DLPDU, qui représente la trame Ethernet, est définie comme une SEQUENCE. La DLPDU comprend ce qui suit: Preamble, SFD, DestAddr, SrcAddr, LT, FAL-PDU et FCS.

3.4.4 Conventions pour la description de bit en octets

Lors de l'identification de chaque bit d'un octet, chaque bit est identifié par un numéro, tel qu'indiqué à la Figure 1 et décrit comme le Bit n.

MSB (Bit de poids fort)							LSB (Bit de poids faible)	
7	6	5	4	3	2	1	0	Numéro d'identification de bit

Figure 1 – Description de bit en octets

Lors de la spécification de plusieurs bits situés de façon séquentielle, le symbole de plage (..) est utilisé (par exemple: 7..0 spécifie les bits 7 à 0 inclusifs).

Lors de la spécification de plusieurs octets, le bit de poids faible (LSB) de l'octet le plus bas est considéré comme étant le Bit 0, puis les numéros d'identification de bit sont attribués par ordre croissant.

NOTE Par exemple, lors de la spécification de 4 octets, le bit de poids fort (MSB) de l'octet le plus haut est le Bit 31, le MSB du deuxième octet est le Bit 23, le MSB du troisième octet est le Bit 15 et le MSB de l'octet le plus bas est le Bit 7.

3.4.5 Conventions relatives aux descriptions des diagrammes d'états

La description du diagramme d'états est définie sous forme de tableau, tel qu'indiqué dans le Tableau 1. La signification des éléments est indiquée dans le Tableau 2. Les conventions utilisées dans les diagrammes d'états sont indiquées dans le Tableau 3.

Chaque ligne du tableau d'états représente une transition d'état. La première colonne indique le nom ou le numéro de la transition d'état. La deuxième colonne indique l'état actuel. La troisième colonne indique les événements, les conditions et les actions. La quatrième colonne indique l'état suivant. Lorsqu'un événement ou une condition se produit, l'action est exécutée et le diagramme d'états passe à l'état suivant.

Tableau 1 – Éléments de la description d'un diagramme d'états

#	Etat actuel	Événement /Condition => Action	Etat suivant

Tableau 2 – Description des éléments d'un diagramme d'états

Galerie pilote	Description
#	nom ou numéro de transition d'état
Etat actuel	état courant
Etat suivant	état de destination
Événement	description de l'événement
Condition	expression logique représentant la condition
=>Action	action exécutée après que l'événement ou la condition se sont produits

Tableau 3 – Conventions utilisées dans les diagrammes d'états

Symbole	Description
=	Substitution du côté droit par le côté gauche
==	Condition logique indiquant qu'une entité de gauche est égale à une entité de droite.
!=	Condition logique indiquant qu'une entité de gauche n'est pas égale à une entité de droite.
<	Condition logique indiquant qu'une entité de gauche est inférieure à une entité de droite.
>	Condition logique indiquant qu'une entité de gauche est supérieure à une entité de droite.
&&	"AND" (c'est-à-dire: PAS) logique
	"OR" (c'est-à-dire: PAS) logique
!	Opérateur de négation
+ – * /	Opérateur arithmétique
;	Point d'interruption

4 Description de la syntaxe FAL

4.1 FALPDU type C – syntaxe abstraite

4.1.1 Syntaxe abstraite de base

Les définitions des FAL-PDU sont indiquées ci-dessous.

```

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

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

```

Les éléments FALARHeader à utiliser dans chaque PDU sont indiqués ci-dessous:

```
FALARHeader ::= SEQUENCE {
    arFType          ARFType,
    priorité          Priority,
    scanNumber       ScanNumber,
    reserved1        Unsigned8,
    srcNodeNumber    NodeNumber,
    reserved2        Unsigned16,
    hec              Hec
}
```

4.1.2 Connect-PDU

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

4.1.3 ConnectAck-PDU

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

4.1.4 Scan-PDU

```
Scan-PDU ::= SEQUENCE {
    falArHeader      FALARHeader,
    scanState        ScanState,
    sendTime         SendTime,
    Remplissage      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,
Remplissage	OctetString SIZE(8),
dcs	dcs

}

CommonParamId ::= SEQUENCE {

date	ParamDate,
timeNodeId	ParamTime,
Somme de contrôle	ParamChecksum

}

4.1.6 Select-PDU

Select-PDU ::= SEQUENCE {

falArHeader	FALARHeader,
Remplissage	OctetString SIZE(28),
dcs	dcs

}

4.1.7 Launch-PDU

Launch-PDU ::= SEQUENCE {

falArHeader	FALARHeader,
Remplissage	OctetString SIZE(28),
dcs	dcs

}

4.1.8 PDU Token

Token-PDU ::= SEQUENCE {

falArHeader	FALARHeader,
Remplissage	OctetString SIZE(28),
dcs	dcs

}

4.1.9 MyStatus-PDU

MyStatus-PDU ::= SEQUENCE {

falArHeader	FALARHeader,
availableFuncs	AvailableFuncs,
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,
données	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,
données	[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,
réservé	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,

dc	dc
}	

4.1.18 CyclicDataIn1-PDU

CyclicDataIn1-PDU::= SEQUENCE {	
falArHeader	FALARHeader,
seqNumber	SeqNumber,
byteValidity	ByteValidity,
dataSize	CycDataSize,
offsetAddr	CycOffsetAddr,
reserved1	Unsigned16,
reserved2	Unsigned16,
in1Data	Cycln1Data,
evenPadding	[0] Unsigned8 OPTIONAL,
dc	dc
}	

4.1.19 CyclicDataIn2-PDU

CyclicDataIn2-PDU::= SEQUENCE {	
falArHeader	FALARHeader,
seqNumber	SeqNumber,
byteValidity	ByteValidity,
dataSize	CycDataSize,
offsetAddr	CycOffsetAddr,
reserved1	Unsigned16,
reserved2	Unsigned16,
in2Data	Cycln2Data,
evenPadding	[0] Unsigned8 OPTIONAL,
dc	dc
}	

4.2 FALPDU type F – syntaxe abstraite

4.2.1 Syntaxe abstraite de base

Les définitions des FAL-PDU sont indiquées ci-dessous.

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

```

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

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

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

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

```

Les éléments FALARHeader utilisés dans chaque PDU sont indiqués ci-dessous.

```

FALAR-FHeader ::= SEQUENCE {
    arFType                ARFType,
    dataType                DataType,
    varField                CHOICE {
        vField0 [0] SEQUENCE {
            persPriority    PersPriority,
            nodeType        NodeType
        },
        vField1 [1] SEQUENCE {
            reserved1       OCTET STRING (SIZE (4))
        },
        vField2 [2] SEQUENCE {

```

nodeId	NodeId,
reserved2	OCTET STRING (SIZE (2))
},	
vField3 [3] SEQUENCE {	
nodeId	NodeId,
syncFlag	SyncFlag,
nodeType	NodeType
},	
vField4 [4] SEQUENCE {	
nodeId	NodeId,
connectionInfo	ConnectionInfo,
reserved4	OCTET STRING (SIZE (1))
}	
},	
srcNodeNumber	NodeNumber,
protocolVerType	ProtocolVerType,
réservé	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,
réservé	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,
    réservé                 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,
    réservé              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,
    réservé              OCTET STRING (SIZE (28)),
    dcs                  dcs
}

```

4.2.10 F-Offset-PDU

```

F-Offset-PDU ::= SEQUENCE {
    falArHeader          FALAR-FHeader,
    réservé              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,
réservé	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,
réservé	OCTET STRING (SIZE (4)),
cycData	CycData,
dcs	dcs
}	

4.2.13 Transient1-PDU

Transient1-PDU::= SEQUENCE {	
falArHeader	FALAR-FHeader,
traMsgHeader	TraMsgHeader,
données	OCTET STRING (SIZE (12 ..1 466)),
dcs	dcs
}	

TraMsgHeader::= SEQUENCE {	
réservé	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 {
    instruction      TraCommand,
    subCommand       TraSubCommand,
    rtn              CHOICE {
        réservé      [0] OCTET STRING (SIZE (2)),
        Valeur        [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,
        réservé         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,
    réservé              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,
    rst                  TraReturnStatus,
    données              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,
    Somme de contrôle     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,
temps	Timer,
réservé	OCTET STRING (SIZE (22)),
dcs	dcs
}	

4.3 Affectation des types de données pour le type C

Les types de données utilisés dans la syntaxe abstraite de FAL-PDU de type C sont indiqués ci-dessous.

```

ARFType ::= Unsigned8
DCS ::= Unsigned8
Priority ::= Unsigned8
ScanNumber ::= Unsigned24
NodeNumber ::= Unsigned16
Hec ::= Unsigned32
PortChoice ::= Unsigned32
PortCheckResult ::= Unsigned32
DestPortInfo ::= Unsigned32
ScanState ::= Unsigned32
SendTime ::= Unsigned16
VendorCode ::= Unsigned16
NodeType ::= Unsigned8
NetNumber ::= Unsigned8
LoopState ::= Unsigned8
ParmTypeCyclicStatus ::= Unsigned8
ParamDate ::= Unsigned32
ParamTime ::= Unsigned32
ParamChecksum ::= Unsigned32
OpState ::= Unsigned16
ErrorState ::= Unsigned16
ErrorCode ::= Unsigned16
DeviceType ::= Unsigned16
UnitTypeName ::= VisibleString SIZE(20)
UnitTypeCode ::= Unsigned16
NodeInfo ::= OctetString SIZE(96)
DestinationGroup ::= Unsigned32
SeqNumber ::= Unsigned8
TraDataId ::= Unsigned8
TraWholeDataSize ::= Unsigned16
TraOffsetAddr ::= Unsigned32
TraDataSize ::= Unsigned16

```

```

TraDataType ::= Unsigned16
TraData ::= LOctetString SIZE(12..1466)
Length ::= Unsigned16
GateCount ::= Unsigned8
TypeSeqF ::= Unsigned8
FrameSequence ::= Unsigned8
DataFrameType ::= Unsigned8
TraDstAddr ::= Unsigned8
TraSrcAddr ::= Unsigned8
TraDstAppType ::= Unsigned8
TraSrcAppType ::= Unsigned8
TraDstModuleFlag ::= Unsigned8
TraSrcModuleFlag ::= Unsigned8
TraDstNetAddr ::= Unsigned8
TraDstStaNo ::= Unsigned8
TraDstID ::= Unsigned16
TraSrcNetAddr ::= Unsigned8
TraSrcStaNo ::= Unsigned8
TraSrcID ::= Unsigned16
TraCmdLen ::= Unsigned16
TraCmdType ::= Unsigned8
TraAppSeq ::= Unsigned16
Tra2Data ::= LOctetString SIZE(12..1466)
NTNTestData ::= OctetString SIZE(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 Affectation des types de données pour le type F

Les types de données utilisés dans la syntaxe abstraite des FAL-PDU de type F sont indiqués ci-dessous.

```

DCS ::= Unsigned32
ARFType ::= Unsigned8
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 – syntaxe abstraite

4.5.1 Syntaxe abstraite de base

Les définitions des FAL-PDU sont indiquées ci-dessous.

```

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

```

<pre> Cyclic-PDU ::= CHOICE { cyclicM-PDU cyclicS-PDU cyclicMs-PDU cyclicSs-PDU } Acyclic-PDU ::= CHOICE { acyclicPriority-PDU acyclicDetection-PDU acyclicDetectionAck-PDU acyclicTestData-PDU acyclicTestDataAck-PDU acyclicData-PDU } PTP-PDU ::= CHOICE { ptpSync-PDU ptpPdelayReq-PDU ptpPdelayResp-PDU ptpFollowUp -PDU ptpPdelayRespFollowUp-PDU ptpAnnounce -PDU ptpdelayReq-PDU ptpdelayResp-PDU ptpdelayRespFollowUp-PDU } IP-PDU ::= SEQUENCE { ipHeader udpHeader udpData slmp-networkConfigMain-PDU slmp-networkConfigTsIt-PDU slmp-notification-PDU slmp-masterConfig-PDU slmp-slaveConfig -PDU slmp-cyclicConfigMain-PDU slmp-cyclicConfigTrnSubPayload-PDU slmp-cyclicConfigRcvSubPayload-PDU slmp-cyclicConfigRcvSrcInfo-PDU } </pre>	<pre> CyclicM-PDU, CyclicS-PDU, CyclicMs-PDU, CyclicSs-PDU AcyclicPriority-PDU, AcyclicDetection-PDU, AcyclicDetectionAck-PDU, AcyclicTestData-PDU, AcyclicTestDataAck-PDU, AcyclicData-PDU PtpSyn-PDU, PtpPdelayReq-PDU, PtpPdelayResp-PDU, PtpFollowUp-PDU, PtpPdelayRespFollowUp-PDU, PtpAnnounce-PDU PtpdelayReq-PDU, PtpdelayResp-PDU, PtpdelayRespFollowUp-PDU, IPHeader, UDPHeader, CHOICE { Slmp-networkConfigMain-PDU, Slmp-networkConfigTsIt-PDU, Slmp-notification-PDU, Slmp-masterConfig-PDU, Slmp-slaveConfig-PDU, Slmp-cyclicConfigMain-PDU, Slmp-cyclicConfigTrnSubPayload-PDU, Slmp-cyclicConfigRcvSubPayload-PDU, Slmp-cyclicConfigRcvSrcInfo-PDU, </pre>

L'en-tête utilisé dans chaque PDU est représenté ci-dessous.

```

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

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

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

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

```

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

AcyclicDataHeader::= SEQUENCE {	
frameType	FrameType,
réservé	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,

<pre> applicationData sdCRC } }</pre>	<pre> OCTET STRING(SIZE(1..1 454)), Unsigned32</pre>

4.5.6 AcyclicPriority-PDU

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

4.5.7 AcyclicDetection-PDU

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

4.5.8 AcyclicDetectionAck-PDU

```

AcyclicDetectionAck-PDU ::= SEQUENCE {
    acyclicDetectionHeader
    acyclicDetectionAckData
        nodeType
        protocolVer
        ipAddressFourthOctet
        srcMAC
        reserved1
        previousNodeMAC
        previousNodePort
        detectRcvPort
        myPort
        reserved2
        myPortLinkStatus
        myPortFilterStatus
        currentManager
        reserved3
        ipAdd
        performance
        reserved4
        gmPriority
        synctype
        reserved5
        pdelayResTime
        delaySetTime
        announceRelayTime
        reserved6
        deviceVer
        vendorCode
        modelCode
        extensionmodelCode
        DeviceType
        memoryAddress
        cyclicSize
        reserved7
        function
        optionInfo
        stationMode
        vide
    }
}

```

```

AcyclicDetectionHeader,
SEQUENCE {
    Unsigned8,
    Unsigned8,
    Unsigned16,
    OCTET STRING (SIZE (6)),
    OCTET STRING (SIZE (2)),
    OCTET STRING (SIZE (6)),
    Unsigned8
    Unsigned8,
    Unsigned8,
    OCTET STRING (SIZE (3)),
    Unsigned96,
    Unsigned96,
    OCTET STRING (SIZE (6)),
    OCTET STRING (SIZE (2)),
    OCTET STRING (SIZE (45)),
    Unsigned8,
    OCTET STRING (SIZE (2)),
    Unsigned48,
    Unsigned8,
    OCTET STRING (SIZE (1)),
    Unsigned8,
    Unsigned8,
    Unsigned8,
    OCTET STRING (SIZE (1)),
    OCTET STRING (SIZE (2)),
    Unsigned16,
    OCTET STRING (SIZE (4)),
    OCTET STRING (SIZE (2)),
    Unsigned16,
    Unsigned224,
    Unsigned112,
    OCTET STRING (SIZE (2)),
    Unsigned8,
    Unsigned8,
    Unsigned16,
    OCTET STRING (SIZE (4))
}

```

AcyclicTestDataHeader ::= SEQUENCE {

```

AcyclicTestDataHeader,
SEQUENCE {
    MACAddress,
    PortNumber,
    OCTET STRING (SIZE (1)),
    SendInfo,
    SEQUENCE {
        OCTET STRING (SIZE (1)),
        HopCount,
        Ipv4Address,
        Ipv4Subnet,
        OCTET STRING (SIZE (8)) ,
        SEQUENCE {
            Ipv6Subnet,
            HopCount,
            Ipv6Address,

```

AcyclicTestDataHeader ::= SEQUENCE {

[illegible]

```

        port22port21
        port24port23
    },
    dcs
}
}

```

```

PortStatus,
PortStatus

```

```

dcs

```

4.5.11 AcyclicData-PDU

```

AcyclicData-PDU ::= SEQUENCE {
    acyclicDataHeader
    données
}

```

AcyclicDataHeader,
SEQUENCE OF OctetString

4.5.12 PtpSync-PDU

Identique à la synchronisation définie dans l'IEEE Std 802.1AS et l'IEEE Std 1588.

4.5.13 PtpPdelayReq-PDU

Identique à Pdelay_Req définie dans l'IEEE Std 802.1AS et l'IEEE Std 1588.

4.5.14 PtpPdelayResp-PDU

Identique à Pdelay_Resp définie dans l'IEEE Std 802.1AS et l'IEEE Std 1588.

4.5.15 PtpFollowUp-PDU

Identique à Follow_Up défini dans l'IEEE Std 802.1AS et l'IEEE Std 1588.

4.5.16 PtpPdelayRespFollowUp-PDU

Identique à Pdelay_Resp_Follow_Up défini dans l'IEEE Std 802.1AS et l'IEEE Std 1588.

4.5.17 PtpAnnounce-PDU

Identique à Announce définie dans l'IEEE Std 802.1AS et l'IEEE Std 1588.

4.5.18 SImpIPAddressSet-PDU

```

SImpIPAddressSet-PDU ::= CHOICE {
    reqIPAddressSet
    resIPAddressSet
}

```

ReqIPAddressSet,
ResIPAddressSet

NOTE Voir la spécification SLMP pour ResIPAddressSet et ResIPAddressSet g.

4.5.19 SImpNetworkConfigMain-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqNetworkConfigMain et ResNetworkConfigMain.

4.5.20 SImpNetworkConfigTsIt-PDU

```

SImpNetworkConfigTsIt-PDU ::= CHOICE {
    reqNetworkConfigTsIt          ReqNetworkConfigTsIt,
    resNetworkConfigTsIt          ResNetworkConfigTsIt
}

```

NOTE Voir la spécification SLMP pour ReqNetworkConfigTsIt et ResNetworkConfigTsIt.

4.5.21 SImpNotification-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqNotification et ResNotification.

4.5.22 SImpMasterConfig-PDU

```

SImpMasterConfig-PDU ::= CHOICE {
    reqMasterConfig                ReqMasterConfig,
    resMasterConfig                ResMasterConfig
}

```

NOTE Voir la spécification SLMP pour ReqMasterConfig et ResMasterConfig.

4.5.23 SImpSlaveConfig-PDU

```

SImpSlaveConfig-PDU ::= CHOICE {
    reqSlaveConfig                 ReqSlaveConfig,
    resSlaveConfig                 ResSlaveConfig
}

```

NOTE Voir la spécification SLMP pour ReqSlaveConfig et ResSlaveConfig.

4.5.24 SImpCyclicConfigMain-PDU

```

SImpCyclicConfigMain-PDU ::= CHOICE {
    reqCyclicConfigMain            ReqCyclicConfigMain,
    resCyclicConfigMain            ResCyclicConfigMain
}

```

NOTE Voir la spécification SLMP pour ReqCyclicConfigMain et ResCyclicConfigMain.

4.5.25 SImpCyclicConfigTrnSubPayload-PDU

```

SImpCyclicConfigTrnSubPayload-PDU ::= CHOICE {
    reqCyclicConfigTrnSubPayload          ReqCyclicConfigTrnSubPayload,
    resCyclicConfigTrnSubPayload          ResCyclicConfigTrnSubPayload
}

```

NOTE Voir la spécification SLMP pour ReqCyclicConfigTrnSubPayload et ResCyclicConfigTrnSubPayload.

4.5.26 SImpCyclicConfigRcvSubPayload-PDU

```

SImpCyclicConfigRcvSubPayload-PDU ::= CHOICE {
    reqCyclicConfigRcvSubPayload          ReqCyclicConfigRcvSubPayload,
    resCyclicConfigRcvSubPayload          ResCyclicConfigRcvSubPayload
}

```

NOTE Voir la spécification SLMP pour ReqCyclicConfigRcvSubPayload et ResCyclicConfigRcvSubPayload.

4.5.27 SImpCyclicConfigRcvSrcInfo-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqCyclicConfigRcvSrcInfo et ResCyclicConfigRcvSrcInfo.

4.5.28 SImpLinkDevicePrmWrite-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqLinkDevicePrmWrite et ResLinkDevicePrmWrite.

4.5.29 SImpLinkDevicePrmWriteCheckRequest-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqLinkDevicePrmWriteCheckReq et ResLinkDevicePrmWriteCheckReq.

4.5.30 SImpLinkDevicePrmWriteCheckResponse-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqLinkDevicePrmWritecheckResp et ResLinkDevicePrmWritecheckResp.

4.5.31 SImpNMTStateUpload-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqNMTStateUpload et ResNMTStateUpload.

4.5.32 SImpNMTStateDownload-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqNMTStateDownload et ResNMTStateDownload.

4.5.33 SImpReadObject-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqReadObject et ResReadObject.

4.5.34 SImpWriteObject-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqWriteObject et ResWriteObject.

4.5.35 SImpObjectSubIDReadBlock-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqObjectSubIDReadBlock et ResObjectSubIDReadBlock.

4.5.36 SImpObjectSubIDWriteBlock-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqObjectSubIDWriteBlock et ResObjectSubIDWriteBlock.

4.5.37 SImpGetODList-PDU

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

NOTE Voir la spécification SLMP pour ReqGetODList et ResGetODList.

4.5.38 SImpGetObjectDescription-PDU

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

NOTE Voir la spécification SLMP pour ReqGetObjectDescription et ResGetObjectDescription.

4.5.39 SImpGetEntryDescription-PDU

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

NOTE Voir la spécification SLMP pour ReqGetEntryDescription et ResGetEntryDescription.

4.5.40 SImpStopOwnStationCyclic-PDU

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

NOTE Voir la spécification SLMP pour ReqStopOwnStationCyclic et ResStopOwnStationCyclic.

4.5.41 SImpStartOwnStationCyclic-PDU

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

NOTE Voir la spécification SLMP pour ReqStartOwnStationCyclic et ResStartOwnStationCyclic.

4.5.42 SImpStopOtherStationsCyclic-PDU

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

NOTE Voir la spécification SLMP pour ReqStopOtherStationsCyclic et ResStopOtherStationsCyclic.

4.5.43 SImpStartOtherStationsCyclic-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqStartOtherStationsCyclic et ResStartOtherStationsCyclic.

4.5.44 SImpAllParameterGet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqAllParameterGet et ResAllParameterGet.

4.5.45 SImpParameterGet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqParameterGet et ResParameterGet.

4.5.46 SImpAllParameterSizeGet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqAllParameterSizeGet et ResAllParameterSizeGet.

4.5.47 SImpParameterSizeGet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqParameterSizeGet et ResParameterSizeGet.

4.5.48 SImpStationSubIDListGet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqStationSubIDListGet et ResStationSubIDListGet.

4.5.49 SImpDeviceIdentificationInfoGet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqDeviceIdentificationInfoGet et ResDeviceIdentificationInfoGet.

4.5.50 SImpDataMonitoring-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqDataMonitoring et ResDataMonitoring.

4.5.51 SImpAllParameterSet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqAllParameterSet et ResAllParameterSet.

4.5.52 SImpParameterSet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqParameterSet et ResParameterSet.

4.5.53 SImpParameterVersionCheck-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqParameterVersionCheck et ResParameterVersionCheck.

4.5.54 SImpDeviceIdentificationInfoCompare-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqDeviceIdentificationInfoCompare et ResDeviceIdentificationInfoCompare.

4.5.55 SImpNodeSearch-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqNodeSearch et ResNodeSearch.

4.5.56 SImpIPAddressSet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqIPAddressSet et ResIPAddressSet.

4.5.57 SImpDeviceInfoCompare-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqDeviceInfoCompare et ResDeviceInfoCompare.

4.5.58 SImpParameterGet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqParameterGet et ResParameterGet.

4.5.59 SImpParameterSet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqParameterSet et ResParameterSet.

4.5.60 SImpParameterSetStart-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqParameterSetStart et ResParameterSetStart.

4.5.61 SImpParameterSetEnd-PDU

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

NOTE Voir la spécification SLMP pour ReqParameterSetEnd et ResParameterSetEnd.

4.5.62 SImpVerifyCheckCode-PDU

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

NOTE Voir la spécification SLMP pour ReqVerifyCheckCode et ResVerifyCheckCode.

4.5.63 SImpOutputMapFileNameGet-PDU

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

NOTE Voir la spécification SLMP pour ReqOutputMapFileNameGet et ResOutputMapFileNameGet.

4.5.64 SImpNewFile-PDU

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

NOTE Voir la spécification SLMP pour ReqNewFile et ResNewFile.

4.5.65 SImpParameterSetCancel-PDU

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

NOTE Voir la spécification SLMP pour ReqParameterSetCancel et ResParameterSetCancel.

4.5.66 SImpOpenFile-PDU

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

NOTE Voir la spécification SLMP pour ReqOpenFile et ResOpenFile.

4.5.67 SImpCloseFile-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqCloseFile et ResCloseFile.

4.5.68 SImpReadFile-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqReadFile et ResReadFile.

4.5.69 SImpWriteFile-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqWriteFile et ResWriteFile.

4.5.70 SImpStatusRead-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqStatusRead et ResStatusRead.

4.5.71 SImpCommunicationSettingGet-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqCommunicationSettingGet et ResCommunicationSettingGet.

4.5.72 SImpGetDeviceInfo-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqGetDeviceInfo et ResGetDeviceInfo.

4.5.73 SImpGetBackupListFileName-PDU

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

NOTE Voir la spécification SLMP pour ReqGetBackupListFileName et ResGetBackupListFileName.

4.5.74 SImpStartBackup-PDU

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

NOTE Voir la spécification SLMP pour ReqStartBackup et ResStartBackup.

4.5.75 SImpEndBackup-PDU

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

NOTE Voir la spécification SLMP pour ReqEndBackup et ResEndBackup.

4.5.76 SImpCheckRestoreVersion-PDU

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

NOTE Voir la spécification SLMP pour ReqCheckRestoreVersion et ResCheckRestoreVersion.

4.5.77 SImpStartRestore-PDU

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

NOTE Voir la spécification SLMP pour ReqStartRestore et ResStartRestore.

4.5.78 SImpEndRestore-PDU

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

NOTE Voir la spécification SLMP pour ReqEndRestore et ResEndRestore.

4.5.79 SImpStatusRead2-PDU

```

SImpStatusRead2-PDU ::= CHOICE {
    reqStatusRead2                               ReqStatusRead2,
    resStatusRead2                               ResStatusRead2
}

```

NOTE Voir la spécification SLMP pour ReqStatusRead2 et ResStatusRead2.

4.5.80 SImpReqSearchNode-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqReqSearchNode et ResReqSearchNode.

4.5.81 SImpGetSearchNodeState-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqGetSearchNodeState et ResGetSearchNodeState.

4.5.82 SImpGetNodeList-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqGetNodeList et ResGetNodeList.

4.5.83 SImpReqSetIPAddress-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqReqSetIPAddress et ResReqSetIPAddress.

4.5.84 SImpSearchPrmControlStation-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqSearchPrmControlStation et ResSearchPrmControlStation.

4.5.85 SImpRequestRestore-PDU

```

SImpRequestRestore-PDU ::= CHOICE {
    ReqRequestRestore          ReqRequestRestore,
    resRequestRestore          ResRequestRestore,
    errRequestRestore          ErrRequestRestore
}

```

NOTE Voir la spécification SLMP pour ReqRequestRestore, ResRequestRestore et ErrRequestRestore.

4.5.86 SImpCheckPrmDelivery-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqCheckPrmDelivery, ResCheckPrmDelivery et ErrCheckPrmDelivery.

4.5.87 SImpRsvStationConfigTemporaryRelease-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqRsvStationConfigTemporaryRelease et ResRsvStationConfigTemporaryRelease.

4.5.88 SImpRsvStationConfig-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqRsvStationConfig et ResRsvStationConfig.

4.5.89 SImpGetEventNum-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqGetEventNum et ResGetEventNum.

4.5.90 SImpGetEventHistory-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqGetEventHistory et ResGetEventHistory.

4.5.91 SImpClearEventHistory-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqClearEventHistory et ResClearEventHistory.

4.5.92 SImpClockOffsetDataSend-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqClockOffsetDataSend et ResClockOffsetDataSend.

4.5.93 SImpSetWatchdogCounterInfo-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqSetWatchdogCounterInfo et ResSetWatchdogCounterInfo.

4.5.94 SImpWatchdogCounterOffsetConfig-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqWatchdogCounterOffsetConfig et ResWatchdogCounterOffsetConfig.

4.5.95 SImpRemoteReset-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqRemoteReset et ResRemoteReset.

4.5.96 SImpGetCommunicationSet-PDU

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

NOTE Voir la spécification SLMP pour ReqGetCommunicationSet et ResGetCommunicationSet.

4.5.97 SImpGetStationSubIDList-PDU

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

NOTE Voir la spécification SLMP pour ReqGetStationSubIDList et ResGetStationSubIDList.

4.5.98 SImpGetDeviceInfo-PDU

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

NOTE Voir la spécification SLMP pour ReqGetDeviceInfo et ResGetDeviceInfo.

4.5.99 SImpStartBackup-PDU

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

NOTE Voir la spécification SLMP pour ReqStartBackup et ResStartBackup.

4.5.100 SImpEndBackup-PDU

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

NOTE Voir la spécification SLMP pour ReqEndBackup et ResEndBackup.

4.5.101 SImpRequestBackup-PDU

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

NOTE Voir la spécification SLMP pour ReqRequestBackup et ResRequestBackup.

4.5.102 SImpGetBackupPrm-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqGetBackupPrm et ResGetBackupPrm.

4.5.103 SImpCheckRestore-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqCheck Restore et ResCheck Restore.

4.5.104 SImpStartRestore-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqStartRestore et ResStartRestore.

4.5.105 SImpEndRestore-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqEndRestore et ResEndRestore.

4.5.106 SImpSetBackupPrm-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqSetBackupPrm et ResSetBackupPrm.

4.5.107 SImpLinkupSpeed-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqCommunicationSpeed et ResCommunicationSpeed.

4.5.108 SImpNodeIndication-PDU

```

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

```

NOTE Voir la spécification SLMP pour ReqNodeIndication et ResNodeIndication.

4.6 Affectation des types de données pour le type T

Les types de données utilisés dans la syntaxe abstraite de FAL-PDU de type T sont indiqués ci-dessous.

```

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 Syntaxe de transfert FAL

5.1 Règles d'encodages

5.1.1 Codage de valeur Unsigned

Les valeurs non signées de longueur fixe Unsigned8, Unsigned16, Unsigned24 et Unsigned32 sont respectivement codées comme valeurs entières non signées d'un octet, deux octets, trois octets et quatre octets de longueur. Les valeurs Unsigned16, Unsigned24 et Unsigned32 sont codées comme gros-boutistes. L'octet de poids fort est considéré comme le premier octet, l'octet suivant comme le deuxième octet et l'octet de poids faible comme le dernier octet.

5.1.2 Codage de chaîne d'octets

La valeur OctetString, dont la chaîne d'octets est de longueur variable, est codée octet par octet, dans l'ordre séquentiel.

5.1.3 Codage de SEQUENCE

Le codage de SEQUENCE (et de SEQUENCE OF) est exécuté selon une séquence, en commençant par l'élément initial. Les identificateurs et la longueur utilisés dans l'ASN.1 ne sont pas utilisés.

5.1.4 Codage de LOctetString

La valeur LOctetString est de longueur variable et codée comme petit-boutiste. L'octet le plus bas, ou le moins important, est le premier octet, puis l'ordre séquentiel est suivi, jusqu'à l'octet le plus haut, ou le plus important, qui est le dernier octet.

5.2 FALPDU type C – encodage des éléments

5.2.1 FALARHeader

5.2.1.1 arFType

Ce champ indique les types de PDU décrits dans le Tableau 4.

Tableau 4 – afFType

Valeur	Description
0x00	Connect-PDU
0x01	ConnectAck-PDU
0x02	Scan-PDU
0x03	Collect-PDU
0x04	Select-PDU
0x05	Launch-PDU
0x06	PDU Token
0x07..0x09	Réservé
0x10	Persuasion-PDU
0x11	TestData-PDU
0x12	TestDataAck-PDU
0x13	Setup-PDU
0x14	SetupAck-PDU
0x15	Token-PDU
0x16..0x1F	Réservé
0x20	MyStatus-PDU
0x22	Transient1-PDU
0x23	Réservé
0x24	Dummy-PDU
0x25	Transient2-PDU
0x26..0x2E	Réservé
0x2F	NTNTest-PDU
0x30..0x7F	Réservé
0x80	CyclicDataW-PDU
0x81	CyclicDataB-PDU
0x82..0x8B	Réservé
0x8C	CyclicDataOut1-PDU
0x8D	CyclicDataOut2-PDU
0x8E	CyclicDataIn1-PDU
0x8F	CyclicDataIn2-PDU

5.2.1.2 priorité

Ce champ indique les priorités décrites dans le Tableau 5.

Tableau 5 – priority

Valeur	Description
0x00	Select-PDU, Launch-PDU, Token-PDU, Dummy-PDU, CyclicDataW-PDU, CyclicDataB-PDU, CyclicDataOut1-PDU, CyclicDataOut2-PDU, CyclicDataIn1-PDU, CyclicDataIn2-PDU
0x01	Collect-PDU, Connect-PDU, ConnectAck-PDU, Scan-PDU
0x02	MyStatus-PDU
0x03	Transient1-PDU
0x04	Transient2-PDU, NTNTest-PDU
0x05	Réservé
0x06	Transient1-PDU

5.2.1.3 scanNumber

Ce champ contient un identificateur de trame utilisé dans Scan-PDU. Le numéro est incrémenté à chaque envoi de Scan-PDU. La valeur par défaut de 0x000000 est utilisée dans tous les autres PDU.

5.2.1.4 reserved1

Il est réservé à un usage ultérieur. La valeur est 0x00.

5.2.1.5 srcNodeNumber

Ce champ contient le numéro identificateur du nœud source.

5.2.1.6 reserved2

Il est réservé à un usage ultérieur. La valeur est 0x0000.

5.2.1.7 hec

Ce champ est un code de vérification d'erreur de DestAddr (DLPDU) jusqu'à reserved2 (FALARHeader). Les définitions DLPDU sont les suivantes.

```

DLPDU ::= SEQUENCE {
    préambule          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
== Unsigned16
FCS ::= OctetString SIZE(4)
    
```

Le polynôme générateur est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$.

5.2.2 Connect-PDU

5.2.2.1 falArHeader

Voir 5.2.1.

5.2.2.2 portChoice

Ce champ indique les types de ports décrits dans le Tableau 6.

Tableau 6 – portChoice

Valeur	Description
0x0	Côté "entrée"
0x1	Côté "sortie"
0xF	Côté "sortie", pendant l'essai entre les nœuds

5.2.2.3 Remplissage

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.2.4 dcs

Ce champ est un code de vérification d'erreur à partir de DestAddr (DLPDU). Pour connaître les définitions des DLPDU, voir 5.2.1.7.

Le polynôme générateur est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$.

5.2.3 ConnectAck-PDU

5.2.3.1 falArHeader

Voir 5.2.1.

5.2.3.2 portCheckResult

Ce champ indique les résultats de vérification de la connexion In-Out. Les valeurs sont spécifiées dans le Tableau 7.

Tableau 7 – portCheckResult

Valeur	Description
0x0	OK
0x1	NG
0xE	Essai entre les nœuds OK
0xF	Essai entre les nœuds pas OK

5.2.3.3 dstPortInfo

Ce champ indique les types de ports de destination. Les valeurs sont spécifiées dans le Tableau 8.

Tableau 8 – dstPortInfo

Valeur	Description
0	Côté "entrée"
1	Côté "sortie"

5.2.3.4 Remplissage

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.3.5 dcs

Voir 5.2.2.4.

5.2.4 Scan-PDU

5.2.4.1 falArHeader

Voir 5.2.1.

5.2.4.2 scanState

Ce champ indique l'état du chemin de transmission. Les valeurs sont spécifiées dans le Tableau 9.

Tableau 9 – scanState

Valeur	Description
0	Etat de traversée
1	État du rebouclage du côté "entrée"
2	État du rebouclage du côté "sortie"

5.2.4.3 sendTime

Ce champ indique le temps envoyé.

5.2.4.4 Remplissage

Ce champ concerne un bourrage. La valeur est 0x00.

5.2.4.5 dcs

Voir 5.2.2.4.

5.2.5 Collect-PDU

5.2.5.1 falArHeader

Voir 5.2.1.

5.2.5.2 vendorCode

Ce champ indique le code du fournisseur. Pour connaître les codes des fournisseurs, voir les règles de mise en œuvre.

5.2.5.3 nodeType

Ce champ indique les types de nœuds décrits dans le Tableau 10.

Tableau 10 – nodeType

Valeur	Description
0x00	Management node
0x01	Nœud de réserve (pour une extension ultérieure)
0x02	Nœud normal
0x10	Nœud de réserve (pour une extension ultérieure)
0x12	Nœud de réserve (pour une extension ultérieure)
0x20	Nœud de réserve (pour une extension ultérieure)
0x21	Nœud principal dans l'essai entre nœuds

5.2.5.4 netNumber

Ce champ contient l'identificateur du réseau auquel le nœud appartient. La plage est 1..239.

5.2.5.5 sendTime

Voir 5.2.4.3.

5.2.5.6 loopState

Ce champ indique l'état de boucle du nœud. Les valeurs sont spécifiées dans le Tableau 11.

Tableau 11 – loopState

Valeur	Description
0x00	Through
0x12	Rebouclage vers le côté "Entrée"; côté "Sortie" déconnecté
0x13	Retour de boucle sur le côté "Entrée", erreur de vérification d'entrée/sortie côté "Sortie"
0x14	Rebouclage vers le côté "entrée"; vérification d'entrée/sortie en cours du côté "sortie"
0x21	Côté "Entrée" déconnecté, retour de la boucle sur le côté "Sortie"
0x31	Erreur de vérification d'entrée/sortie du côté "Entrée"; rebouclage vers le côté "sortie"
0x41	Vérification d'entrée/sortie en cours du côté "Entrée"; rebouclage vers le côté "sortie"

5.2.5.7 parmTypeCyclicStatus

Ce champ est divisé de la façon suivante:

Bit 0..5	Etat cyclique – suivre les valeurs du Tableau 12
Bit 6..7	Mode paramétrage – suivre les valeurs du Tableau 13

Tableau 12 – État cyclique

Valeur	Description
0x00	La transmission cyclique n'est pas exécutée.
0x01	La transmission cyclique est exécutée.
0x02	Paramètres communs non reçus
0x03	Réception des paramètres communs en cours
0x04	Erreur dans les paramètres communs
0x05	Réservé
0x06	Le numéro de nœud est interdit
0x07	Paramétrage du nœud de réserve
0x08	Instruction d'arrêt de la transmission cyclique
0x09	Exécution d'essai hors ligne
0x0A	Expiration du temporisateur de surveillance
0x0B	Numéro de nœud non défini
0x0C	Erreur d'UC de nœud
0x0D	Le numéro de nœud est en double.
0x0E	Duplication de nœud de gestion
0x0F	Duplication de numéro de nœud et de nœud de gestion
0x10	Erreur de numéro de réseau

Tableau 13 – Mode paramétrage

Valeur	Description
0x0	Paramètre commun
0x1	Réservé

5.2.5.8 commonParamId

5.2.5.8.1 date

Les bits de ce champ sont définis comme suit:

Bit 28..31	Année (chiffre des centaines)
Bit 24..27	Année (chiffre des milliers)
Bit 20..23	Année (chiffre des unités)
Bit 16..19	Année (chiffre des dizaines)
Bit 12..15	Mois (chiffre des unités)
Bit 8..11	Mois (chiffre des dizaines)
Bit 4..7	Jour (chiffre des unités)
Bit 0..3	Jour (chiffre des dizaines)

Si tous les bits sont égaux à 0, il n'existe pas de paramètre commun.

5.2.5.8.2 timeNodeId

Les bits de ce champ sont définis comme suit:

Bit 28..31	Heures (chiffre des unités)
Bit 24..27	Heures (chiffre des dizaines)
Bit 20..23	Minutes (chiffre des unités)
Bit 16..19	Minutes (chiffre des dizaines)
Bit 12..15	Secondes (chiffre des unités)
Bit 8..11	Secondes (chiffre des dizaines)
Bit 0..7	Numéro de nœud du réglage de référence

Si tous les bits sont égaux à 0, il n'existe pas de paramètre commun.

5.2.5.8.3 Somme de contrôle

Ce champ contient la somme de contrôle "date" et "time_nodeId" dans commandParmlId. Si tous les bits sont égaux à 0, il n'existe pas de paramètre commun.

5.2.5.9 Remplissage

Ce champ comprend 8 octets de longueur. La valeur est 0x00 pour les 8 octets.

5.2.5.10 dcs

Voir 5.2.2.4.

5.2.6 Select-PDU

5.2.6.1 falArHeader

Voir 5.2.1.

5.2.6.2 Remplissage

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.6.3 dcs

Voir 5.2.2.4.

5.2.7 Launch-PDU**5.2.7.1 falArHeader**

Voir 5.2.1.

5.2.7.2 Remplissage

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.7.3 dcs

Voir 5.2.2.4.

5.2.8 PDU Token**5.2.8.1 falArHeader**

Voir 5.2.1.

5.2.8.2 Remplissage

Ce champ concerne le bourrage. La valeur est 0x00.

5.2.8.3 dcs

Voir 5.2.2.4.

5.2.9 MyStatus-PDU**5.2.9.1 falArHeader**

Voir 5.2.1.

5.2.9.2 availableFuncs

Ce champ représente la disponibilité du mode étendu du point de transmission..

- 0x01: le mode de transmission étendu est disponible.
- 0x00: le mode de transmission étendu n'est pas disponible.

5.2.9.3 nodeType

Voir 5.2.5.3.

5.2.9.4 netNumber

Voir 5.2.5.4.

5.2.9.5 reserved2

Il est réservé à un usage ultérieur. La valeur est 0x0000.

5.2.9.6 loopState

Voir 5.2.5.6.

5.2.9.7 parmTypeCyclicStatus

Voir 5.2.5.7.

5.2.9.8 commonParamId

Voir 5.2.5.8.

5.2.9.9 inFarNodeMACAddr

Ce champ indique l'adresse MAC du nœud connecté au côté "Entrée". Si le côté "Entrée" n'est pas connecté correctement, la valeur est 0.

5.2.9.10 inFarNodeNumber

Ce champ indique le numéro de nœud du nœud connecté au côté "Entrée".

5.2.9.11 reserved3

Il est réservé à un usage ultérieur. La valeur est 0x00.

5.2.9.12 outFarNodeMACAddr

Ce champ indique l'adresse MAC du nœud connecté au côté "Sortie". Si le côté "Sortie" n'est pas connecté correctement, la valeur est 0.

5.2.9.13 outFarNodeNumber

Ce champ indique le numéro de nœud du nœud connecté au côté "Sortie".

5.2.9.14 reserved4

Il est réservé à un usage ultérieur. La valeur est 0x00.

5.2.9.15 opState

Ce champ indique l'état d'un nœud. Les valeurs sont spécifiées dans le Tableau 14.

Tableau 14 – opState

Valeur	Description
0	Contrôleur inexistant
1	Contrôleur arrêté
2	Contrôleur en cours de fonctionnement

5.2.9.16 errorState

Ce champ indique l'état d'erreur d'un nœud. Les valeurs sont spécifiées dans le Tableau 15.

Tableau 15 – errorState

Valeur	Description
0	Pas d'erreur
1	Erreur mineure
2	Erreur majeure
3	Erreur grave

5.2.9.17 errorCode

Ce champ indique les codes d'erreur des erreurs survenues au niveau d'un nœud. Les codes d'erreur et leur signification ne sont pas définis ici.

5.2.9.18 vendorCode

Voir 5.2.5.2.

5.2.9.19 DeviceType

Ce champ spécifie le type d'appareil. Pour connaître les types d'appareils, voir le profil de l'appareil.

5.2.9.20 unitTypeName

Ce champ contient la chaîne de caractères du nom du modèle.

5.2.9.21 unitTypeCode

Ce champ contient le code du nom du modèle.

5.2.9.22 reserved5

Il est réservé à un usage ultérieur. La valeur est 0x0000.

5.2.9.23 nodeInfo

Ce champ indique l'état d'un nœud défini par l'utilisateur.

5.2.9.24 dcs

Voir 5.2.2.4.

5.2.10 Transient1-PDU**5.2.10.1 falArHeader**

Voir 5.2.1.

5.2.10.2 destinationGroup

Ce champ spécifie le groupe de destination. Chaque bit représente une adresse de groupe. Le Bit 0 indique l'adresse du groupe 1. Le Bit 31 indique l'adresse du groupe 32. Si aucun groupe de destination n'est spécifié, les valeurs de tous les bits sont 0.

5.2.10.3 seqNumber

La signification de chaque bit de ce champ est la suivante:

Bit 0..6	Indique le numéro des fragments de données.
Bit 7	Indique si les données constituent le dernier fragment. La valeur 0 indique qu'il ne s'agit pas de la dernière PDU. La valeur 1 indique qu'il s'agit de la dernière PDU du dernier fragment.

5.2.10.4 dataId

Ce champ contient le numéro d'identification des données transitoires. La plage de valeurs est 0x00..0xFF. Tous les fragments d'une transmission de données transitoires possèdent le même numéro d'identification.

5.2.10.5 wholeDataSize

Ce champ spécifie la quantité de données transitoires en octets.

5.2.10.6 offsetAddr

Ce champ spécifie l'adresse de décalage. Dans la PDU initiale, la valeur est 0x0. Dans les PDU suivantes, la position au sein de l'ensemble des données transitoires est indiquée sous la forme d'une adresse de décalage par rapport à la PDU initiale.

5.2.10.7 dataSize

Ce champ spécifie la taille des données transitoires fragmentées en octets. La plage de valeurs est 0x0000..0x05BA.

5.2.10.8 DataType

Ce champ spécifie les types de données décrits dans le Tableau 16.

Tableau 16 – Data type

Valeur	Description
0x0000	Réservé
0x0001	Remise de paramètres
0x0002..0xFFFF	Réservé

5.2.10.9 données

5.2.10.9.1 Vue d'ensemble

Le champ de données contient les données transitoires. La structure dépend du type de données transitoires. Si la valeur wholeDataSize dépasse 1 466 octets, il s'agit des données transitoires comprises entre le décalage décrit dans offsetAddr et la taille des données décrite dans dataSize. Si les données transitoires comptent moins de 16 octets, le bourrage est rempli avec 0x00

5.2.10.9.2 Structure de la remise de paramètres

Lorsque le type de données est la remise de paramètres, les données sont sélectionnées parmi CPW, CPWC ou CPWCR. La structure de CPW est décrite dans le Tableau 17, celle de CPWC dans le Tableau 18 et celle de CPWCR dans le Tableau 19. Chaque champ est codé comme LOctetString.

Tableau 17 – CPW

Nom du champ	Taille (octets)	Description
cpfType	4	Type de trame. La valeur est 0.
rcvNodeList	32	Liste des nœuds de réception. Chaque bit représente un nœud. Le bit de poids faible (LSB) de l'octet le plus bas représente le nœud 1 et le bit de poids fort (MSB) de l'octet le plus bas représente le nœud 8. La valeur 1 signifie qu'il s'agit d'un nœud de réception et la valeur 0 signifie que le nœud n'est pas un nœud de réception.
commonParamId	12	Voir 5.2.5.8.
cmParam	–	Voir Tableau 20.

Tableau 18 – CPWC

Nom du champ	Taille (octets)	Description
cpfType	4	Type de trame. La valeur est 1.
commonParamId	12	Voir 5.2.5.8.

Tableau 19 – CPWCR

Nom du champ	Taille (octets)	Description
cpfType	4	Type de trame. La valeur est 2.
checkResult	4	Résultat de la vérification des paramètres communs 0: Paramètres communs non reçus 1: Vérification des paramètres communs 2: Vérification OK 3: Echec de la vérification
errorCode	4	Code d'erreur en cas d'échec de la vérification
srcNodeNumber	1	Source node number
Remplissage	3	Remplissage

Tableau 20 – cmParam

Nom du champ	Taille (octets)	Description
Nom de paramètre	8	Spécification par l'utilisateur
Total Size	2	Taille des données du bloc de début au bloc de fin en octets
Sum Check Enabled (Contrôle de somme activé)	1	0: Exécution du contrôle de la totalisation 1: Non-exécution du contrôle de la totalisation (par défaut)
Réservé	1	Réservé
Sum Check Value (Valeur de somme de contrôle)	4	Valeur de somme de contrôle du bloc de début au bloc de fin
Create Time (Date et heure de création)	12	Année/mois/jour/heure/minute/seconde de création

Nom du champ	Taille (octets)	Description	
		Taille (octets)	Description
		1	Deux derniers chiffres de A.D. (année)
		1	Deux premiers chiffres de A.D. (année)
		2	Mois
		2	Jour
		2	Heure
		2	Minute
		2	Seconde
Begin Marker (Marqueur de début)	4	0x5047532d	
Param Area (zone de paramètres)	0..5 924	Voir Tableau 21.	
End Marker (Marqueur de fin)	4	0x5047452d	

Tableau 21 – Détails de param area

Nom du champ	Taille (octets)	Description
Parameter Availability	2	Bit 0: LB/LW Common Memory Area Bit 1: LB/LW Common Memory Additional Area Bit 8: Zone mémoire commune LX/LY Bit 9: Zone mémoire commune LX/LY supplémentaire 0 signifie qu'aucun réglage n'est disponible; 1 signifie que des réglages sont disponibles
Version de paramètre commun	1	Version de paramètre commun
Réservé	1	Réservé
LB/LW CM Area Offset	2	Décalage de zone mémoire commune LB/LW
LB/LW CM Additional Area Offset	2	Décalage de zone mémoire commune LB/LW supplémentaire
LX/LY CM 1 Area Offset	2	Décalage de zone mémoire commune 1 LX/LY
LX/LY CM 2 Area Offset	2	Décalage de zone mémoire commune 2 LX/LY
Réservé	24	Réservé
Application parameters	80	Voir Tableau 22
LB/LW CM Area	0 ou 1 452	Voir Tableau 23. Si la valeur Bit 0 de Parameter Availability est 0, la taille est 0; si la valeur est 1, la taille est 1 452
LB/LW CM Additional Area	0 ou 1 452	Voir Tableau 23. Si la valeur Bit 1 de Parameter Availability est 0, la taille est 0; si la valeur est 1, la taille est 1 452
LX/LY CM 1 Area	0 ou 1 452	Voir Tableau 24. Si la valeur Bit 8 de Parameter Availability est 0, la taille est 0; si la valeur est 1, la taille est 1 452
LX/LY CM 2 Area	0 ou 1 452	Voir Tableau 24. Si la valeur Bit 9 de Parameter Availability est 0, la taille est 0; si la valeur est 1, la taille est 1 452

Tableau 22 – Détails des paramètres d'application

Nom du champ	Taille (octets)	Description
Control Block	4	0x0
Reserved1	1	Reserved 1
Reserved2	1	Reserved 2
Total Nodes	1	Nombre total de nœuds
Network Type	1	Network type
Acyclic Times	2	Temps de transition
Supervisory Period	2	Temps de surveillance (en ms)
Reserved3	20	Reserved 3
Reserved4	16	Reserved 4
Reserved5	16	Reserved 5
Reserved6	16	Reserved 6

Tableau 23 – Détails de LB/LW CM area et de LB/LW CM additional area

Nom du champ	Taille (octets)	Description
LW CM Head Address	4	Adresse relative de début de LW
LW CM Total Size	4	Nombre total de mots dans la plage de réglage LW
LB CM Head Address	2	Adresse relative de début de LB
LB CM Total Size	2	Taille totale de la plage de réglage LB (en unités de 2 octets)
LB/LW CM Table List	1 440 (12 × 120)	Liste des paramètres LB et LW
LW CM Head Address Of Node	4	Adresse relative de début de LW dans chaque nœud
LW CM Size	4	Taille de LW dans chaque nœud (en unités de 2 octets)
LB CM Head Address Of Node	2	Adresse relative de début de LB dans chaque nœud
LB CM Size	2	Taille de LB dans chaque nœud (en unités de 2 octets)

Tableau 24 – Détails de LX/LY CM 1 area et de LX/LY CM 2 area

Nom du champ	Taille (octets)	Description
Master Node Number	1	Master node number
Réservé	3	Réservé
LY CM Head Address	2	Adresse relative de début de LY
LY CM Total Size	2	Nombre total de mots dans la plage de réglage LY
LX CM Head Address	2	Adresse relative de début de LX
LX CM Total Size	2	Nombre total de mots dans la plage de réglage LX
LX/LY CM Table List	1 440 (12 × 120)	Liste des paramètres LX/LY
LY CM Head Address Sent	2	Adresse relative de début de LY envoyée par chaque nœud
LY CM Size	2	Taille de LY envoyée par chaque nœud (en unités de 2 octets)

Nom du champ		Taille (octets)	Description
	LX CM Head Address Master Received	2	Adresse relative de début de LX reçue par le nœud principal
	LX CM Head Address Received	2	Adresse relative de début de LX reçue par chaque nœud
	LX CM Size	2	Taille de LX reçue par chaque nœud (en unités de 2 octets)
	LY CM Head Address Master Sent	2	Adresse relative de début de LY envoyée par le nœud principal

5.2.10.10 evenPadding

Ce champ n'est utilisé que si le champ de données est un nombre impair d'octets. La valeur est 0x00.

5.2.10.11 dcs

Voir 5.2.2.4.

5.2.11 Dummy-PDU

5.2.11.1 falArHeader

Voir 5.2.1.

5.2.11.2 dummyData

Ce champ contient des données fictives. La taille est comprise entre 28 octets et 1 482 octets. La valeur n'a aucune signification.

5.2.11.3 dcs

Voir 5.2.2.4.

5.2.12 Transient2-PDU

5.2.12.1 falArHeader

Voir 5.2.1.

5.2.12.2 l

Ce champ spécifie la longueur des données de fno à data (en octets).

5.2.12.3 gcnt

Ce champ représente le temps maximal de retransmission sous la forme d'un nombre de portes. La valeur par défaut est 0x07. La valeur est décrémentée au passage de chaque relais.

5.2.12.4 typeSeqF

Ce champ est divisé de la façon suivante:

Bit 7..4	Indique le type. La valeur est 0x00.
Bit 3..0	Représente le numéro de séquence.

5.2.12.5 fno

Ce champ est divisé de la façon suivante:

Bit 7	Indique l'identification du support. Si la valeur est 0, cela signifie que la trame est une trame sans début. Si la valeur est 1, cela signifie que la trame est une trame support.
Bit 6..0	Indique le numéro de trame divisée. La valeur 0 indique que la trame n'a pas été divisée. Les valeurs 1 à 7 indiquent le numéro de trame divisée. Le numéro de trame divisée commence par la même valeur que le numéro de division et est soustrait selon une séquence. Dans une trame divisée en 3, par exemple, l'ordre des numéros de trames est: 3, 2, 1.

5.2.12.6 dt

Ce champ est divisé de la façon suivante:

Bit 7	Indique la priorité. Si la valeur est 0, cela signifie que la priorité est faible. Si la valeur est 1, cela signifie que la priorité est élevée.
Bit 6	Indique la présence de trames de réponse. Si la valeur est 0, cela signifie qu'une trame de réponse est exigée. Si la valeur est 1, cela signifie qu'aucune trame de réponse n'est nécessaire.
Bit 5..0	Réservé à un usage futur.

5.2.12.7 da

Ce champ contient le numéro de nœud pour un nœud de relais. Si un nœud de destination se trouve dans le même réseau, ce champ identifie le numéro de nœud de destination.

5.2.12.8 sa

Ce champ contient le numéro de nœud pour un nœud de relais. Si un nœud de destination se trouve dans le même réseau, ce champ identifie le numéro de nœud source.

5.2.12.9 dat

Ce champ contient le type d'application cible. La valeur est fixée à 0x22.

5.2.12.10 sat

Ce champ contient le type d'application source. La valeur est fixée à 0x22.

5.2.12.11 dmf

Ce champ contient l'indicateur de module de destination. Les valeurs sont décrites dans le Tableau 25.

Tableau 25 – Indicateur de module de destination

Nom du module	Description
0	A l'intérieur du module de réseau
1	A l'intérieur du contrôleur

5.2.12.12 smf

Ce champ contient l'indicateur de module source. Les valeurs sont décrites dans le Tableau 25.

5.2.12.13 dna

Ce champ contient le numéro de réseau du nœud de destination.

5.2.12.14 ds

Ce champ contient le numéro de nœud du nœud de destination.

5.2.12.15 did

Ce champ est divisé de la façon suivante:

Bit 10..15	Zone de contrôle (stocke le numéro du nœud de destination)
Bit 0..9	Numéro d'identification de cible. La valeur est fixée à 0x3FF.

5.2.12.16 sna

Ce champ contient le numéro de réseau du nœud source.

5.2.12.17 ss

Ce champ contient le numéro de nœud du nœud source.

5.2.12.18 sid

Ce champ est divisé de la façon suivante:

Bit 10..15	Zone de contrôle (stocke le numéro du nœud source)
Bit 0..9	Numéro d'identification de source. La valeur est fixée à 0x3FF.

5.2.12.19 l1

Ce champ spécifie la longueur des données de ct à data (en octets).

5.2.12.20 ct

Ce champ contient le type de commande tel qu'indiqué dans le Tableau 26.

Tableau 26 – Types de commandes

Type de commande	Description
0x00	Non disponible
0x01	Réservé
0x02	Réservé
0x03	Réservé
0x04	Obtention d'informations sur l'accès à la mémoire
0x05..0x07	Réservé
0x08	Run (Exécution)
0x09	STOP (Arrêt=
0x0A..0x0E	Réservé
0x0F	Réservé
0x10	Lire mémoire

Type de commande	Description
0x11	Réservé
0x12	Ecriture dans la mémoire
0x13..0x5F	Réservé
0x60..0x7F	Propre au fournisseur

5.2.12.21 rsv

Il est réservé à un usage ultérieur. La valeur est 0x0.

5.2.12.22 aps

Ce champ est divisé de la façon suivante:

Bit 8..15	Numéro de tâche. La plage de valeurs est 0..255.
Bit 0..7	Numéro d'identification d'application source. La plage de valeurs est 0..255.

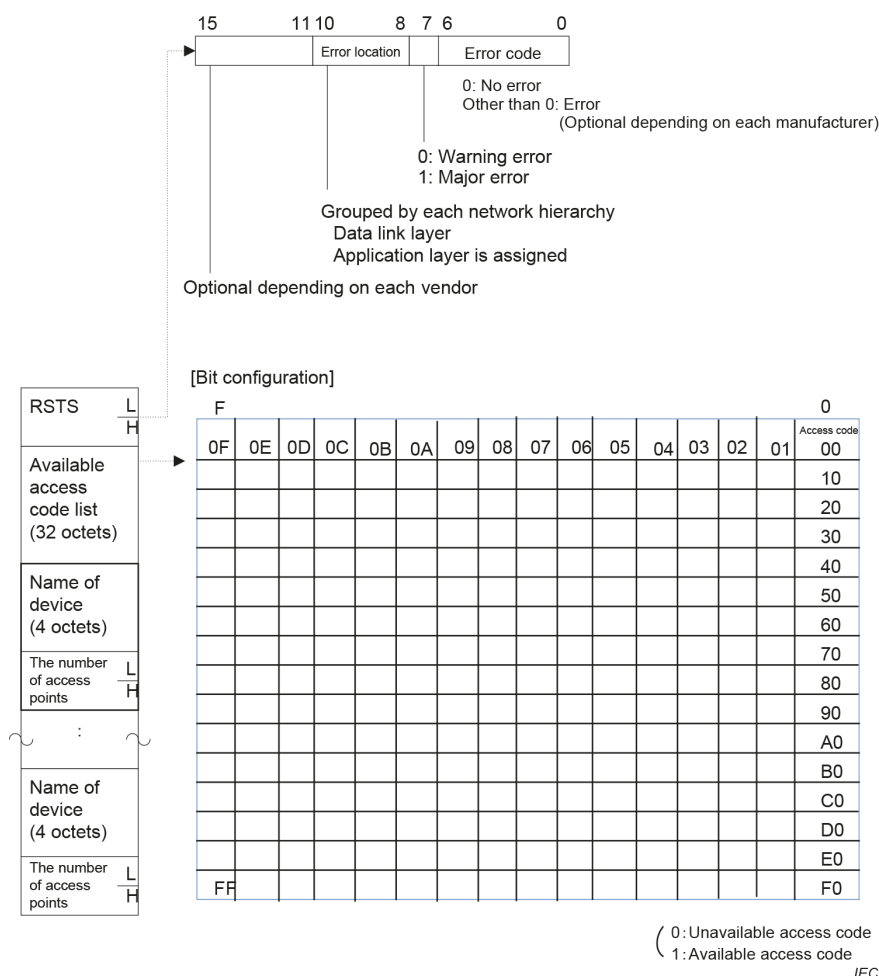
5.2.12.23 données

5.2.12.23.1 Vue d'ensemble

Ce champ contient les données transitoires. La structure des données transitoires dépend de ct (voir 5.2.12.20).

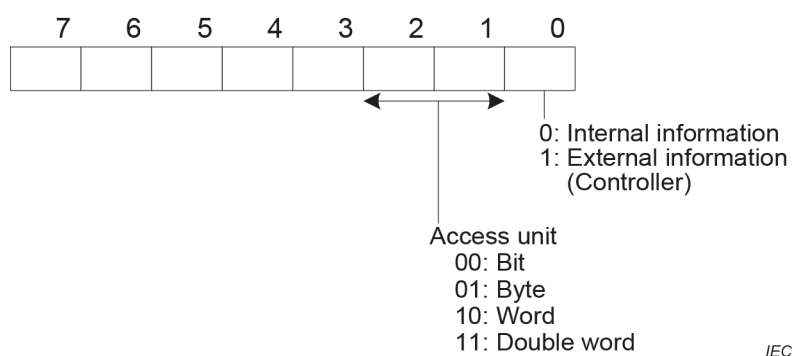
5.2.12.23.2 Obtention d'informations sur l'accès à la mémoire

En cas de demande d'obtention d'informations sur l'accès à la mémoire, cette zone n'est pas utilisée. La structure de la réponse est indiquée à la Figure 2. La définition de l'attribut est indiquée à la Figure 3. La définition du code d'accès est indiquée à la Figure 4.



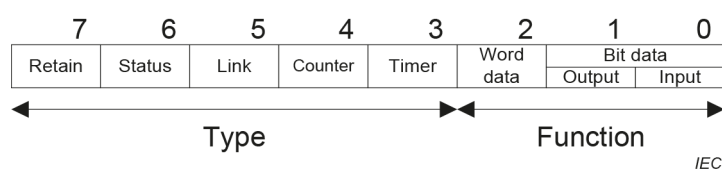
Anglais	Français
Error location	Error location
Error code	Code d'erreur
0: No error	0: Pas d'erreur
Other than 0: Error (Optional depending on each manufacturer)	Autre que 0: Erreur (facultatif, en fonction du fabricant)
0: Warning error	0: Warning error
1: Major error	1: Erreur majeure
Grouped by each network hierarchy	Groupe par hiérarchie de réseau
Data link layer	Couche liaison de données
Application layer is assigned	La couche application est attribuée
Optional depending on each vendor	Facultatif, en fonction du fournisseur
Bit configuration	Configuration de bit
Access code	Code d'accès
RSTS	RSTS
L/H	L/H
Available access code list (32 octets)	Liste de codes d'accès disponibles (32 octets)
Name of device (4 octets)	Nom de l'appareil (4 octets)
The number of access points	Nombre de points d'accès
0: Unavailable access code	0: Code d'accès indisponible
1: Available access code	1: Code d'accès disponible

Figure 2 – Structure de réponse à une demande de récupération d'informations sur l'accès à la mémoire



Anglais	Français
0: Internal information	0: Informations internes
1: External information (controller)	1: Informations externes (contrôleur)
Access unit	Unité d'accès
00: Bit	00: Bit
01: Byte	01: Octet
10: Word	10: Mot
11: Double word	11: Double mot

Figure 3 – Définitions des attributs



Anglais	Français
Retain	Conserver
Status	Statut
Link	Liaison
Counter	Compteur
Timer	Temporisateur
Word data	Données de mot
Bit data	Données de bit
Output	Sortie
Input	Entrée
Type	Type
Function	Fonction

Figure 4 – Définitions des codes d'accès

Un exemple de code d'accès à la mémoire du module de réseau est décrit dans le Tableau 27.
Un exemple de code d'accès à la mémoire du contrôleur est décrit dans le Tableau 28.

Tableau 27 – Codes d'accès à la mémoire du module de réseau

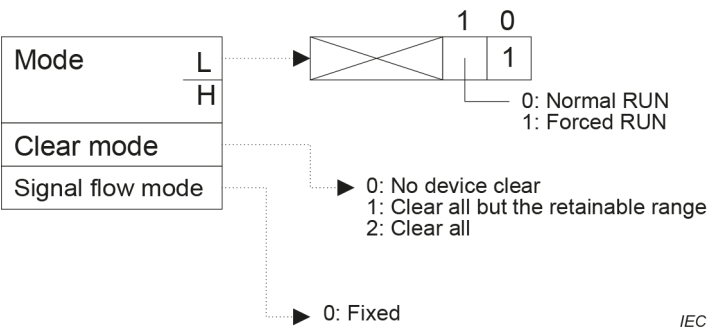
Contenu de la mémoire		Code d'accès
Tampon	Tampon normalisé	0x00
Tampon d'état	Nœud d'appareil intelligent Tampon à rafraîchissement automatique	0x40
Tampon de liaison	Tampon à accès aléatoire	0x20
Appareil de liaison	Entrée de liaison	0x21
	Sortie de liaison	0x22
	Registre de liaison	0x24
	Relais spécial de liaison	0x63
	Registre spécial de liaison	0x64
NOTE Les champs et les valeurs de propriété spécifiques dépendent du fournisseur.		

Tableau 28 – Codes d'accès à la mémoire du contrôleur

Contenu de la mémoire	Code d'accès	Type	
		B	E
Relais d'entrée	0x01	X	
Relais de sortie	0x02	X	
Relais spécial	0x43	X	
Registre spécial	0x44		X
Relais interne	0x03	X	
Relais à verrouillage	0x83	X	
Temporisateur (contact)	0x09	X	
Temporisateur (bobine)	0x0A	X	
Temporisateur (valeur courante)	0x0C		X
Temporisateur cumulatif (contact)	0x89	X	
Temporisateur cumulatif (bobine)	0x8A	X	
Temporisateur cumulatif (valeur courante)	0x8C		X
Compteur (contact)	0x11	X	
Compteur (bobine)	0x12	X	
Compteur (valeur courante)	0x14		X
Registre de données	0x04		X
Registre	0x84		X
Relais de liaison	0x23	X	
Registre de liaison	0x24		X
Relais spécial de liaison	0x63	X	
Registre spécial de liaison	0x64		X
NOTE Le nom, le numéro et la plage d'accès à la mémoire de l'appareil dépendent du PLC.			

5.2.12.23.3 Marche

La structure en cas de demande RUN est indiquée à la Figure 5.



Anglais	Français
Mode	communication
Clear mode	Clear mode
Signal flow mode	Mode de flux de signaux
Normal RUN	EXÉCUTION normale
Forced RUN	EXÉCUTION forcée
0: No device clear	0: Aucun effacement d'appareil
1: Clear all but the retainable range	1: Tout effacer sauf la plage à conserver
2: Clear all	2: Tout effacer
0: Fixed	0: Fixe

Figure 5 – Structure en cas de demande RUN

La structure en cas de réponse est indiquée à la Figure 6.

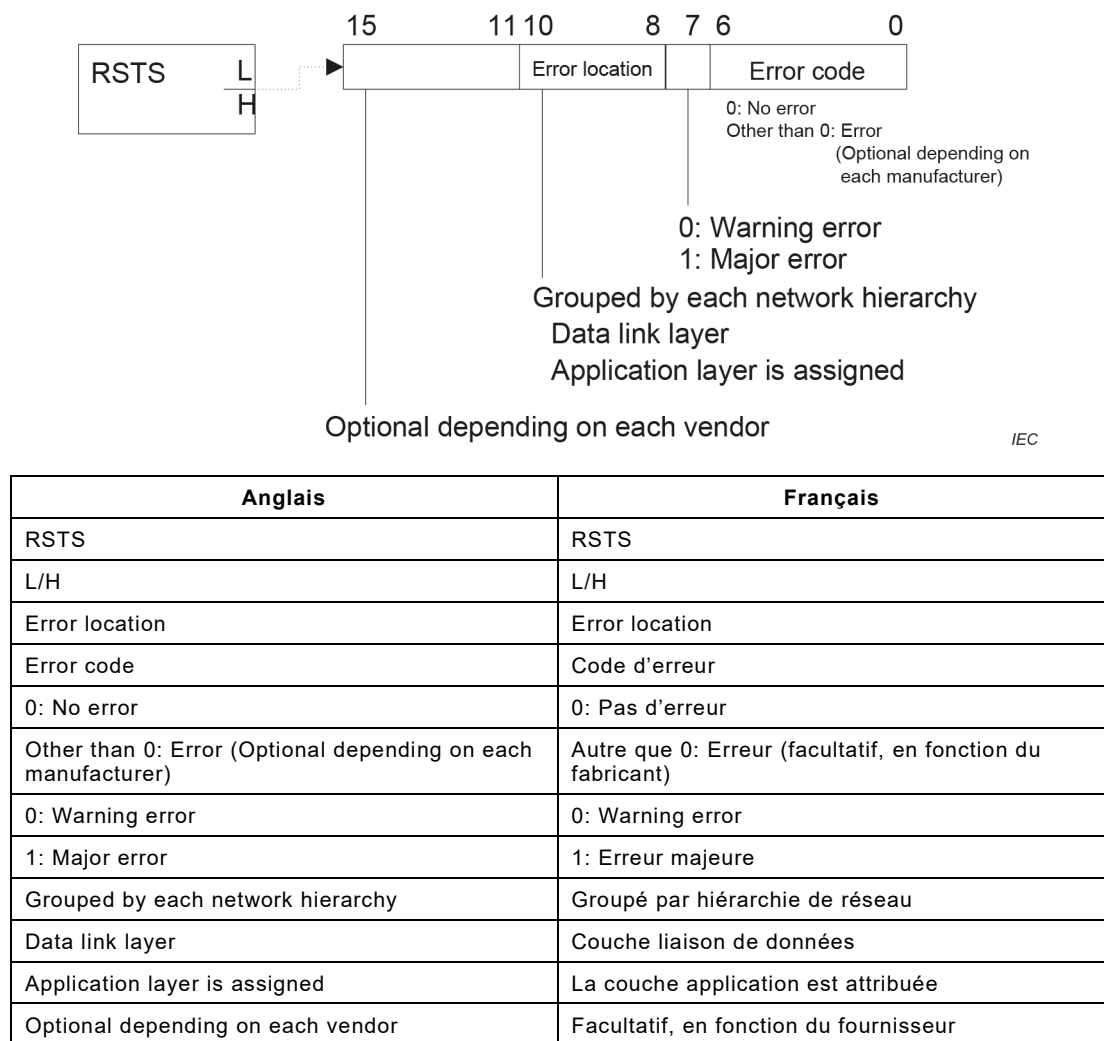


Figure 6 – Structure en cas de réponse à une demande RUN

5.2.12.23.4 Arrêter

La structure en cas de demande STOP est indiquée à la Figure 7. La structure en cas de réponse est indiquée à la Figure 8.

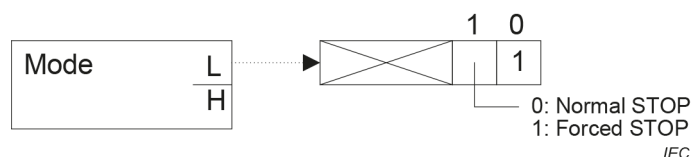
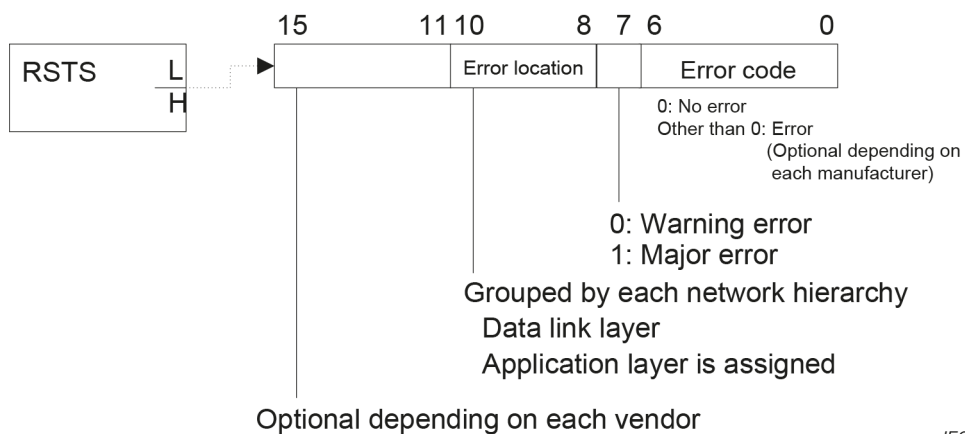


Figure 7 – Structure en cas de demande STOP



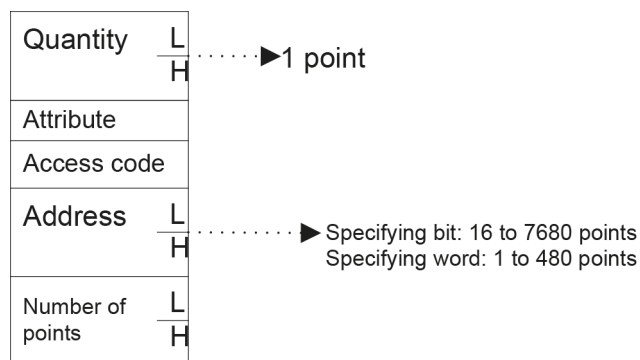
IEC

Anglais	Français
RSTS	RSTS
L/H	L/H
Error location	Error location
Error code	Code d'erreur
0: No error	0: Pas d'erreur
Other than 0: Error (Optional depending on each manufacturer)	Autre que 0: Erreur (facultatif, en fonction du fabricant)
0: Warning error	0: Warning error
1: Major error	1: Erreur majeure
Grouped by each network hierarchy	Groupé par hiérarchie de réseau
Data link layer	Couche liaison de données
Application layer is assigned	La couche application est attribuée
Optional depending on each vendor	Facultatif, en fonction du fournisseur

Figure 8 – Structure en cas de réponse à une demande STOP

5.2.12.23.5 Lire mémoire

La structure en cas de demande de lecture de mémoire en lot est indiquée à la Figure 9. La structure en cas de réponse est indiquée à la Figure 10.



IEC

Anglais	Français
Quantity	Quantité
L/H	L/H
1 point	1 point
Attribute	Attribut
Access code	Code d'accès
Address	Adresse
Number of points	Number of points
Specifying bit: 16 to 7680 points	Spécification de bit: 16 à 7 680 points
Specifying word 1 to 480 points	Spécification de mot de 1 à 480 points

Figure 9 – Structure en cas de demande de lecture de mémoire par lots

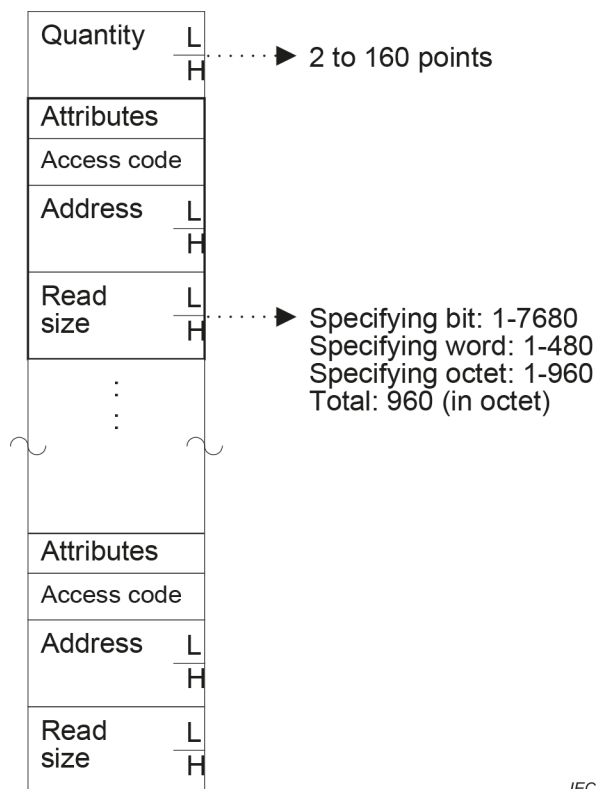


IEC

Anglais	Français
Data area	Zone de données

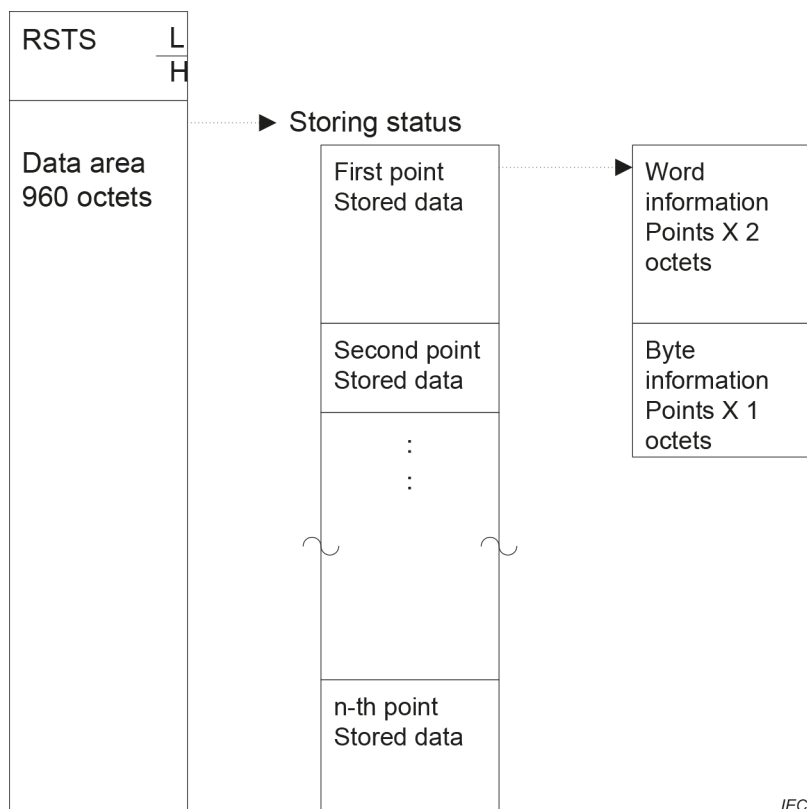
Figure 10 – Structure en cas de réponse à une demande de lecture de mémoire par lots

La structure en cas de demande de lecture de mémoire aléatoire est indiquée à la Figure 11.
La structure en cas de réponse est indiquée à la Figure 12.



Anglais	Français
Quantity	Quantité
2 to 160 points	2 à 160 points
Attributes	Attributs
Access code	Code d'accès
Address	Adresse
Read size	Taille de lecture
Specifying bit: 1-7680	Spécification de bit: 1-7680
Specifying word: 1-480	Spécification de mot: 1-480
Specifying octet: 1-960	Spécification d'octet: 1-960
Total: 960 (in octet)	Total: 960 octets

Figure 11 – Structure en cas de demande de lecture de mémoire aléatoire

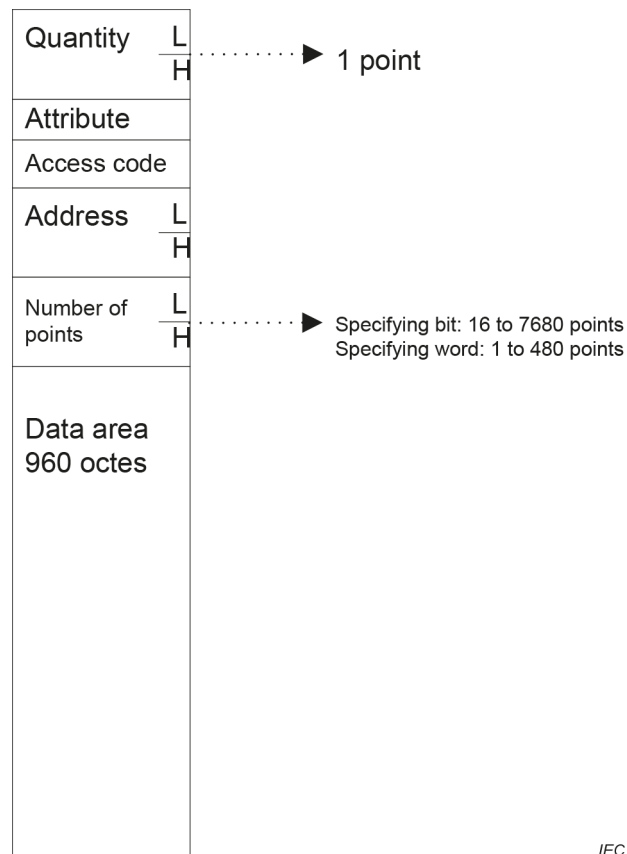


Anglais	Français
Data area	Zone de données
960 octets	960 octets
Storing status	Statut de stockage
First point	Premier point
Stored data	Données stockées
Second point	Deuxième point
n-th point	n-ième point
Word information Points X 2 octets	Points d'information de mot X 2 octets
Byte information Points X 1 octets	Points d'information d'octet X 1 octet

Figure 12 – Structure en cas de réponse à une demande de lecture de mémoire aléatoire

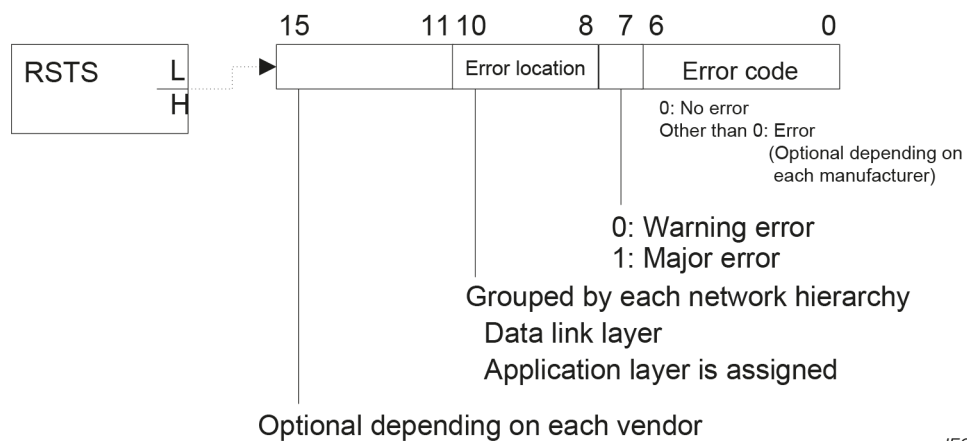
5.2.12.23.6 Écrire en mémoire

La structure en cas de demande d'écriture dans la mémoire en lot est indiquée à la Figure 13. La structure en cas de réponse est indiquée à la Figure 14.



Anglais	Français
Quantity	Quantité
1 point	1 point
Attribute	Attribut
Access code	Code d'accès
Address	Adresse
Number of points	Number of points
Specifying bit: 16 to 7680 points	Spécification de bit: 16 à 7 680 points
Specifying word 1 to 480 points	Spécification de mot de 1 à 480 points
Data area 960 octets	Zone de données 960 octets

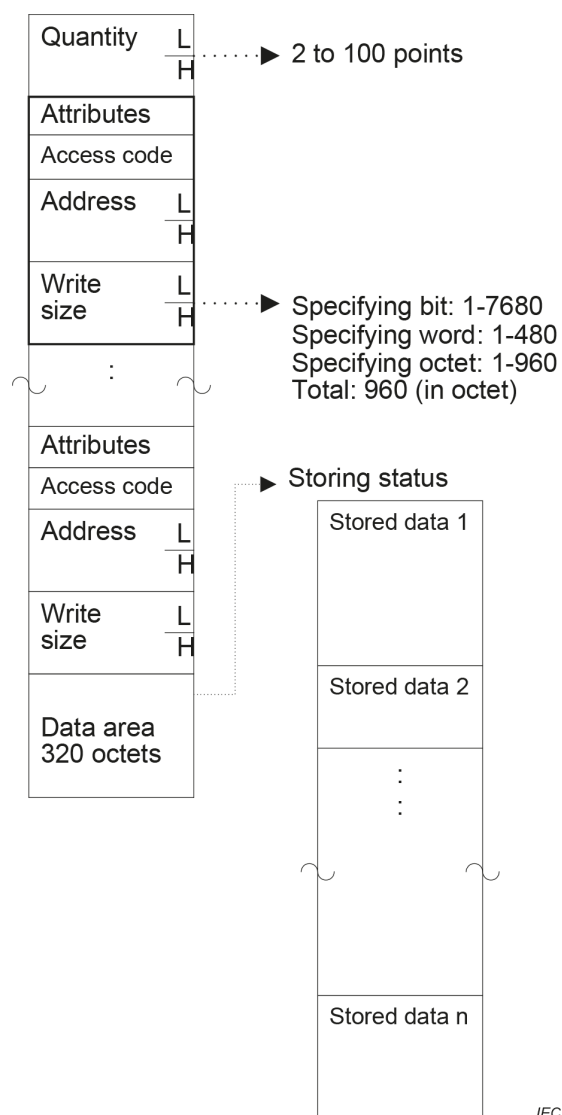
Figure 13 – Structure en cas de demande d'écriture mémoire par lots



Anglais	Français
Error location	Error location
Error code	Code d'erreur
0: No error	0: Pas d'erreur
Other than 0: Error (Optional depending on each manufacturer)	Autre que 0: Erreur (facultatif, en fonction du fabricant)
0: Warning error	0: Warning error
1: Major error	1: Erreur majeure
Grouped by each network hierarchy	Groupé par hiérarchie de réseau
Data link layer	Couche liaison de données
Application layer is assigned	La couche application est attribuée
Optional depending on each vendor	Facultatif, en fonction du fournisseur

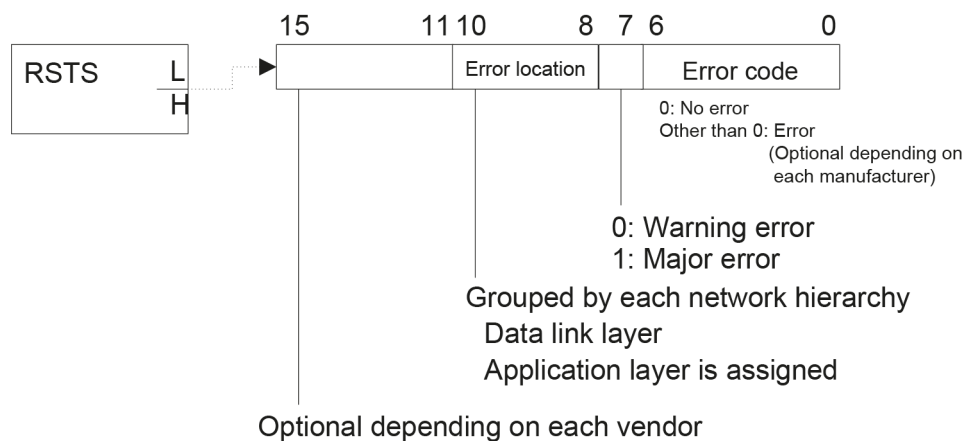
Figure 14 – Structure en cas de réponse à une demande d'écriture mémoire par lots

La structure en cas de demande d'écriture dans la mémoire aléatoire est indiquée à la Figure 15.
La structure en cas de réponse est indiquée à la Figure 16.



Anglais	Français
Quantity	Quantité
2 to 100 points	2 à 100 points
Attributes	Attributs
Access code	Code d'accès
Address	Adresse
Write size	Write size
Specifying bit: 1-7680	Spécification de bit: 1-7680
Specifying word: 1-480	Spécification de mot: 1-480
Specifying octet: 1-960	Spécification d'octet: 1-960
Total: 960 (in octet)	Total: 960 octets
Data area 320 octets	Zone de données 320 octets
Storing status	Statut de stockage
Stored data 1	Données stockées 1
Stored data 2	Données stockées 2
Stored Data n	Données stockées n

Figure 15 – Structure en cas de demande d'écriture mémoire aléatoire



IEC

Anglais	Français
Error location	Error location
Error code	Code d'erreur
0: No error	0: Pas d'erreur
Other than 0: Error (Optional depending on each manufacturer)	Autre que 0: Erreur (facultatif, en fonction du fabricant)
0: Warning error	0: Warning error
1: Major error	1: Erreur majeure
Grouped by each network hierarchy	Groupé par hiérarchie de réseau
Data link layer	Couche liaison de données
Application layer is assigned	La couche application est attribuée
Optional depending on each vendor	Facultatif, en fonction du fournisseur

Figure 16 – Structure en cas de réponse à une demande d'écriture mémoire aléatoire

5.2.12.24 evenPadding

Ce champ n'est utilisé que si le champ de données contient un nombre impair d'octets. La valeur est 0x00.

5.2.12.25 dcs

Voir 5.2.2.4.

5.2.13 NTNTest-PDU

5.2.13.1 falArHeader

Voir 5.2.1.

5.2.13.2 ntnTestData

Ce champ contient les données d'essai fictives. La taille est comprise entre 28 octets et 1 482 octets. La valeur n'a aucune signification.

5.2.13.3 dcs

Voir 5.2.2.4.

5.2.14 CyclicDataW-PDU

5.2.14.1 falArHeader

Voir 5.2.1.

5.2.14.2 seqNumber

Voir 5.2.10.3.

5.2.14.3 byteValidity

Ce champ spécifie si les 4 premiers octets et les 4 derniers octets de wData sont reflétés sur la mémoire partagée. Les définitions des bits sont indiquées dans le Tableau 29. Pour chaque bit, 1 = valide et 0 = invalide.

Tableau 29 – byteValidity

Bit	Octet de wData
0	premier octet
1	deuxième octet
2	troisième octet
3	quatrième octet
4	quatrième octet à partir de la fin
5	troisième octet à partir de la fin
6	deuxième octet à partir de la fin
7	dernier octet

5.2.14.4 dataSize

Ce champ spécifie la taille des données cycliques. La plage pour cette valeur est 0x0..0x16F. La taille est spécifiée par le nombre de groupes de 4 octets; les 4 premiers octets et les 4 derniers octets avec byteValidity sont également inclus dans la taille.

5.2.14.5 offsetAddr

Ce champ spécifie l'adresse de décalage de LW en octets. La plage pour cette valeur est 0x0..0x3FFFC; seuls les multiples de 4 sont autorisés. En d'autres termes, la valeur du Bit 0..1 est 0.

5.2.14.6 exSeqNumber

Ce champ identifie le numéro de séquence associé aux données fragmentées. La plage pour cette valeur est 0x0..0xFFFF. Lorsque ce champ est utilisé, la valeur seqNumber est 0x7F.

5.2.14.7 réservé

Il est réservé à un usage ultérieur. La valeur est 0x0.

5.2.14.8 WData

Ce champ contient les données LW. Si la valeur de données contient moins de 16 octets de longueur, le bourrage est de 0x00.

5.2.14.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de longueur de wData est impair. La valeur est 0x00.

5.2.14.10 dcs

Voir 5.2.2.4.

5.2.15 CyclicDataB-PDU**5.2.15.1 falArHeader**

Voir 5.2.1.

5.2.15.2 seqNumber

Voir 5.2.14.2.

5.2.15.3 byteValidity

Voir 5.2.14.3.

5.2.15.4 dataSize

Voir 5.2.14.4.

5.2.15.5 offsetAddr

Ce champ spécifie l'adresse de décalage de LB en octets. La plage pour cette valeur est 0x0..0x3FFFC; seuls les multiples de 4 sont autorisés. En d'autres termes, la valeur du Bit 0..1 est 0.

5.2.15.6 reserved1

Il est réservé à un usage ultérieur. La valeur est 0x0.

5.2.15.7 reserved2

Il est réservé à un usage ultérieur. La valeur est 0x0.

5.2.15.8 bData

Ce champ contient les données LB. Si les données comptent moins de 16 octets, le bourrage est de 0x00.

5.2.15.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de longueur de bData est impair. La valeur est 0x00.

5.2.15.10 dcs

Voir 5.2.2.4.

5.2.16 CyclicDataOut1-PDU**5.2.16.1 falArHeader**

Voir 5.2.1.

5.2.16.2 seqNumber

Voir 5.2.14.2.

5.2.16.3 byteValidity

Ce champ n'est pas utilisé. La valeur est 0x00.

5.2.16.4 dataSize

Voir 5.2.14.4.

5.2.16.5 offsetAddr

Ce champ n'est pas utilisé. La valeur est 0x00.

5.2.16.6 reserved1

Ce champ est réservé. La valeur est 0x0.

5.2.16.7 reserved2

Ce champ est réservé. La valeur est 0x0.

5.2.16.8 out1Data

Ce champ contient les données LY1 envoyées par le nœud principal à tous les nœuds de réception. Si la valeur des données contient moins de 16 octets de longueur, le bourrage est de 0x00.

5.2.16.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de longueur de out1Data est impair. La valeur est 0x00.

5.2.16.10 dcs

Voir 5.2.2.4.

5.2.17 CyclicDataOut2-PDU

5.2.17.1 falArHeader

Voir 5.2.1.

5.2.17.2 seqNumber

Voir 5.2.14.2.

5.2.17.3 byteValidity

Ce champ n'est pas utilisé. La valeur est 0x00.

5.2.17.4 dataSize

Voir 5.2.14.4.

5.2.17.5 offsetAddr

Ce champ n'est pas utilisé. La valeur est 0x00.

5.2.17.6 reserved1

Ce champ est réservé. La valeur est 0x0.

5.2.17.7 reserved2

Ce champ est réservé. La valeur est 0x0.

5.2.17.8 out2Data

Ce champ contient les données LY2 envoyées par le nœud principal à tous les nœuds de réception. Si la valeur de ces données contient moins de 16 octets de longueur, le bourrage est de 0x00.

5.2.17.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de longueur de out2Data est impair. La valeur est 0x00.

5.2.17.10 dcs

Voir 5.2.2.4.

5.2.18 CyclicDataIn1-PDU**5.2.18.1 falArHeader**

Voir 5.2.1.

5.2.18.2 seqNumber

Voir 5.2.14.2.

5.2.18.3 byteValidity

Voir 5.2.14.3.

5.2.18.4 dataSize

Voir 5.2.14.4.

5.2.18.5 offsetAddr

Ce champ spécifie l'adresse de décalage de LX en octets. La plage pour cette valeur est 0x0..0x3FFFC; seuls les multiples de 4 sont autorisés. En d'autres termes, la valeur du Bit 0..1 est 0.

5.2.18.6 reserved1

Ce champ est réservé. La valeur est 0x0.

5.2.18.7 reserved2

Ce champ est réservé. La valeur est 0x0.

5.2.18.8 in1Data

Ce champ contient les données LX1 envoyées par chaque nœud d'émission au nœud principal. Si la valeur des données contient moins de 16 octets de longueur, le bourrage est de 0x00.

5.2.18.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de longueur de in1Data est impair. La valeur est 0x00.

5.2.18.10 dcs

Voir 5.2.2.4.

5.2.19 CyclicDataIn2-PDU

5.2.19.1 falArHeader

Voir 5.2.1.

5.2.19.2 seqNumber

Voir 5.2.14.2.

5.2.19.3 byteValidity

Voir 5.2.14.3.

5.2.19.4 dataSize

Voir 5.2.14.4.

5.2.19.5 offsetAddr

Ce champ spécifie l'adresse de décalage de LX en octets. La plage pour cette valeur est 0x0..0x3FFFC; seuls les multiples de 4 sont autorisés. En d'autres termes, la valeur du Bit 0..1 est 0.

5.2.19.6 reserved1

Ce champ est réservé. Il est paramétré sur 0x0.

5.2.19.7 reserved2

Ce champ est réservé. Il est paramétré sur 0x0.

5.2.19.8 in2Data

Ce champ contient les données LX2 envoyées par chaque nœud d'émission au nœud principal. Si la valeur des données contient moins de 16 octets de longueur, le bourrage est de 0x00.

5.2.19.9 evenPadding

Ce champ n'est utilisé que si le nombre d'octets de longueur de in2Data est impair. La valeur est 0x00.

5.2.19.10 dcs

Voir 5.2.2.4.

5.3 FALPDU type F – encodage des éléments

5.3.1 FALARHeader

5.3.1.1 arFType

Ce champ indique les types de PDU décrits dans le Tableau 30.

Tableau 30 – afFType

Valeur	Description
0x00..0x06	Utilisation avec le type C
0x07..0x09	Réservé à un usage ultérieur
0x10	persuasion-PDU
0x11	testData-PDU
0x12	testDataAck-PDU
0x13	setup-PDU
0x14	setupAck-PDU
0x15	token-PDU
0x16..0x1B	Réservé à un usage ultérieur
0x1C	timer-PDU
0x1D..0x1F	Réservé à un usage ultérieur
0x20	myStatus-PDU
0x21	Réservé à un usage ultérieur
0x22	transient1-PDU
0x23	transientAck-PDU
0x24	Utilisation avec le type C
0x25	transient2-PDU
0x26..0x27	Réservé à un usage ultérieur
0x28	paramCheck-PDU
0x29	parameter-PDU
0x2A..0x3F	Réservé à un usage ultérieur
0x40	measure-PDU
0x41	measureAck-PDU
0x42	offset-PDU
0x43	update-PDU
0x44..0x7F	Réservé à un usage ultérieur
0x80..0x81	Utilisation avec le type C
0x82	cyclicDataRWw-PDU
0x83	cyclicDataRY-PDU
0x84	cyclicDataRWr-PDU
0x85	cyclicDataRX-PDU
0x86..0x8B	Réservé à un usage ultérieur
0x8C..0x8F	Utilisation avec le type C
0x90..0xFF	Réservé à un usage ultérieur

5.3.1.2 DataType

Ce champ indique le type de données, dont les valeurs sont classées par arFtype comme décrit dans le Tableau 31.

Tableau 31 – dataType

arFtype	Valeur	Description
0x10..0x15	0x00	Réservé à un usage ultérieur
	0x01	Contrôle de transmission
	0x02..0xFF	Réservé à un usage ultérieur
0x1C	0x00	Réservé à un usage ultérieur
	0x01	Transmission transitoire (spécifique au type F)
	0x02..0xFF	Réservé à un usage ultérieur
0x20	0x00..0x01	Réservé à un usage ultérieur
	0x02	Contrôle de transmission (diagnostic)
	0x03..0xFF	Réservé à un usage ultérieur
0x22	0x00..0x06	Réservé à un usage ultérieur
	0x07	Transmission transitoire (spécifique au type F)
	0x08	Transmission transitoire (spécifique à l'entraînement)
	0x09	Transmission transitoire (spécifique à la sécurité)
	0x10..0xFF	Réservé à un usage ultérieur
0x23	0x00..0xFF	Réponse à la transmission transitoire
0x25	0x00..0x03	Réservé à un usage ultérieur
	0x04	Transmission transitoire (compatible CP8/1, CP8/2)
	0x05..0xFF	Réservé à un usage ultérieur
0x28 à 0x29	0x00..0x02	Réservé à un usage ultérieur
	0x03	Réglage de transmission cyclique
	0x04..0xFF	Réservé à un usage ultérieur
0x82 à 0x85	0x00	Transmission cyclique
	0x01..0xFF	Réservé à un usage ultérieur

5.3.1.3 varField

5.3.1.3.1 Vue d'ensemble

Ce champ contient les champs énumérés dans le Tableau 32, classés par arFtype comme décrit.

Tableau 32 – varField

arFtype	Champ utilisé
persuasion-PDU testDataAck-PDU	persPriority nodeType
testData-PDU setupAck-PDU timer-PDU	reserved1
setup-PDU 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

Ce champ indique le niveau de priorité lorsque le gestionnaire de contrôle de transmission est sélectionné. La plage pour cette valeur est 0x0..0xFFFFFF. La valeur 0x0 indique que le nœud n'est pas le gestionnaire de contrôle de transmission.

5.3.1.3.3 nodeType

Ce champ indique le type de nœud. Les valeurs sont spécifiées dans le Tableau 33.

Tableau 33 – nodeType

Valeur	Description
0x00..0x2F	Utilisation avec le type C
0x30	Station principale
0x31	Réservé à un usage ultérieur
0x32	Station locale
0x33	Station d'appareil intelligent
0x34	Station d'appareil distant
0x35	Station d'E/S distante
0x36..0xFF	Réservé à un usage ultérieur

5.3.1.3.4 reserved1

Réservé à un usage futur. La valeur de chaque octet est 0x00.

5.3.1.3.5 nodeld

Ce champ contient l'identificateur de nœud. Le gestionnaire de contrôle de transmission détermine les identificateurs des nœuds autres que le gestionnaire de contrôle de transmission et définit les valeurs à l'aide de setup-PDU. La plage de valeurs est 0..255. Si un numéro de nœud n'a pas été défini, la valeur est 255.

5.3.1.3.6 reserved2

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.1.3.7 syncFlag

La signification de chaque bit est la suivante:

Bits 7..1	Réservé à un usage futur. La valeur est 0.
Bit 0	Définit l'indicateur de synchronisation. La valeur est 1 si la trame fournit des informations relatives à la séquence de synchronisation; dans le cas contraire, la valeur est 0.

5.3.1.3.8 connectionInfo

Ce champ identifie la valeur f-transientData-PDU à transmettre si un jeton est maintenu. Si un jeton est maintenu et si une PDU de transmission non cyclique en double présentant les mêmes valeurs destaddr et connectionInfo est reçue, le nœud de réception procède au rejet.

5.3.1.3.9 reserved4

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.1.3.10 srcNodeNumber

Ce champ contient le numéro d'identification du nœud source.

5.3.1.3.11 protocolVerType

La signification de chaque bit est la suivante:

Bits 7..4:	Version du protocole. La valeur est 0.
Bits 3..0:	Type de protocole. Les valeurs utilisées sont conformes au Tableau 34.

Tableau 34 – ProtocolVerType

Valeur	Description
0x0	type C
0x1	type F
0x2..0xF	Réservé à un usage ultérieur

5.3.1.3.12 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.1.3.13 hec

Ce champ contient un code de détection d'erreur qui cible l'élément DestAddr de la DLPDU. Les définitions DLPDU sont les suivantes.

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

```

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

```

Le polynôme générateur est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$.

5.3.2 Persuasion-PDU

5.3.2.1 falArHeader

Voir 5.3.1.

5.3.2.2 reserved1

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.2.3 myPorts

Ce champ indique le nombre de ports de communication physiques détenus par le nœud.

5.3.2.4 vendorCode

Ce champ contient le code du fournisseur.

5.3.2.5 modelCode

Ce champ contient le code du modèle spécifique au fournisseur.

5.3.2.6 reserved2

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.2.7 dcs

Ce champ contient le code de détection d'erreur qui cible l'élément destaddr de la DLPDU dans le champ de dcs précédent. Pour connaître les définitions des DLPDU, voir 5.3.1.3.13.

Le polynôme générateur est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$.

5.3.3 TestData-PDU

5.3.3.1 falArHeader

Voir 5.3.1.

5.3.3.2 **tmMacAddr**

Ce champ contient l'adresse MAC du gestionnaire de contrôle de transmission.

5.3.3.3 **srcPort**

Ce champ contient le numéro de port source de transmission de testData-PDU. La valeur 0 est invalide.

5.3.3.4 **réservé**

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.3.5 **dcs**

Voir 5.3.2.7.

5.3.4 **TestDataAck-PDU**

5.3.4.1 **falArHeader**

Voir 5.3.1.

5.3.4.2 **tdSrcMacAddr**

Ce champ contient l'adresse MAC du nœud source de transmission de testData-PDU inclus comme valeur srcaddr dans la DLPDU du testData-PDU reçu. La cible est testData-PDU qui servait d'impulsion pour la transmission de testDataAck-PDU.

5.3.4.3 **tdSrcPort**

Ce champ contient le numéro de port source du nœud source de transmission de testData-PDU inclus comme valeur srcPort dans le testData-PDU reçu.

5.3.4.4 **tdRcvPort**

Ce champ contient le numéro de port de son propre nœud qui a reçu le testDataAck-PDU.

5.3.4.5 **reserved1**

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.4.6 **myPorts**

Voir 5.3.2.3.

5.3.4.7 **tokenKeepTime**

Ce champ spécifie la durée maximale pendant laquelle le nœud maintient un jeton après le début du passage de jeton. Le nœud avertit le gestionnaire de contrôle de transmission de la durée de maintien du jeton par l'intermédiaire de ce champ.

La signification de chaque bit est la suivante:

Bit 15:	Réservé à un usage futur. La valeur est 0.
Bits 14..0:	Indique la durée définie. L'unité est 1 µs. La plage de valeurs est 1..32 767.

NOTE Les points de départ et de fin de la durée de maintien du jeton sont identiques à ceux de token-PDU. Si le point de départ est le début de la réception du token-PDU, le point final est le début de la transmission de token-PDU. Si le point de départ est défini comme la fin de la réception de token-PDU, le point final est défini comme la fin de la transmission de token-PDU.

5.3.4.8 reserved2

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.4.9 myConnectStatus

Ce champ indique l'état de chaque port. Le champ est utilisé pour indiquer l'état des 24 ports, au maximum, à l'aide de quatre bits pour chaque port. Le champ indique l'état du port 1 avec les bits 3..0 et l'état du port 2 avec les bits 7..4 du premier octet. Le champ indique l'état du port 3 avec les bits 3..0 et l'état du port 4 avec les bits 7..4 du deuxième octet. De la même façon, le champ utilise ensuite jusqu'à 12 octets pour indiquer les états des 24 ports. La signification de chaque bit d'un octet est la suivante:

La signification de chaque bit est la suivante:

Bits 7..6:	Réservé à un usage futur. La valeur est 0.
Bits 5..4:	Les valeurs sont conformes au Tableau 35.
Bits 3..2:	Réservé à un usage futur. La valeur est 0.
Bits 1..0:	Les valeurs sont conformes au Tableau 35.

Tableau 35 – Etat de liaison

Valeur	Description
00b	Liaison descendante
01b	Liaison établie (1 Gbit/s)
10b	Réservé à un usage ultérieur
11b	Réservé à un usage ultérieur

5.3.4.10 dcs

Voir 5.3.2.7.

5.3.5 Setup-PDU

5.3.5.1 falArHeader

Voir 5.3.1.

5.3.5.2 tokenDstMacAddr

Ce champ spécifie l'adresse MAC du nœud suivant qui maintient le jeton sur le chemin de passage du jeton. Le gestionnaire de contrôle de transmission transmet le setup-PDU qui a défini l'adresse MAC de la destination de transmission du token-PDU de ce champ au nœud qui sert d'origine de transmission du token-PDU. L'adresse MAC spécifiée dans ce champ est maintenue dans le nœud qui a reçu le setup-PDU.

Une fois le passage du jeton commencé, le gestionnaire de contrôle de transmission devient le premier nœud à détenir le jeton. Le nœud qui détient le jeton transmet ses propres données, puis transmet le token-PDU avec l'adresse MAC spécifiée dans ce champ dans le tokenDstMacAddr. Le nœud qui reçoit le token-PDU détermine si sa propre adresse MAC de nœud et la valeur de tokenDstMacAddr du token-PDU correspondent. Si tel est le cas, il évalue si le token-PDU est adressé à sa propre adresse de nœud. Un nœud qui reçoit un token-PDU qui lui est adressé devient le nœud qui détient le jeton.

5.3.5.3 reserved1

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.5.4 leaveTimerValue

Ce champ spécifie la valeur de réglage de LeaveTimer. Le gestionnaire de contrôle de transmission entre la valeur LeaveTimer à régler dans un nœud autre que le gestionnaire de contrôle de transmission dans ce champ du setup-PDU à transmettre. La valeur de ce champ est définie dans le nœud qui reçoit le setup-PDU comme valeur LeaveTimer. LeaveTimer est utilisé pour mesurer la durée de la détection de la déconnexion de son propre nœud.

La signification des bits de ce champ est la suivante:

Bit 15:	Réservé à un usage futur. La valeur est 0.
Bits 14..0:	Valeur LeaveTimer. La plage de valeurs est 1..32 767. L'unité est 400 µs.

5.3.5.5 portUsage

Ce champ indique le réglage d'activation/de désactivation de la fonction transmission/réception du port associé au nœud. La désactivation d'un port signifie que le port n'utilise pas le service DL. Les valeurs utilisées sont conformes au Tableau 36.

Dans un nœud avec deux ports qui reçoivent des valeurs de réglage, l'état activé/désactivé des ports du nœud est déterminé conformément au Tableau 36. Un nœud disposant de trois ports ou plus ayant reçu des valeurs de réglage active tous les ports sans utiliser les valeurs de réglage spécifiées.

Tableau 36 – Spécification d'activation/de désactivation de port

Valeur	Description
0x00	Activation des deux ports
0x01	Activation du port 1 uniquement et désactivation de tous les autres ports.
0x02	Activation du port 2 uniquement et désactivation de tous les autres ports.

5.3.5.6 reserved2

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.5.7 netBehaviour

5.3.5.7.1 multipleTransmit

Ce champ spécifie le nombre de fois où le nœud qui détient le jeton procède à la transmission des FAL-PDU autres que le token-PDU. La plage de valeurs est 1..255. 0 n'est pas valide.

Si ce champ contient la valeur 1, le nœud qui détient le jeton transmet myStatus-PDU, f-cyclicData-PDU et f-transientData-PDU, puis transmet enfin le token-PDU. Par conséquent, le nœud ne détient plus le jeton. Dans le cas contraire, le nœud ignore f-transientData-PDU. Le nœud qui détient le jeton transmet myStatus-PDU, f-cyclicData-PDU et f-transientData-PDU le nombre de fois spécifié dans ce champ, après quoi il transmet le token-PDU.

NOTE Par exemple, si une station à appareil intelligent, qui spécifie 2 dans ce champ, devient le nœud qui détient le jeton, la station transmet myStatus-PDU, cyclicRWr-PDU, cyclicRX-PDU et transient1-PDU, puis transmet à nouveau myStatus-PDU, cyclicRWr-PDU, cyclicRX-PDU et transient1-PDU, suivis par le token-PDU.

5.3.5.7.2 frameInterval

Ce champ spécifie l'intervalle entre la réception de token-PDU et la transmission de myStatus-PDU. La plage de valeurs est 1..255. 0 n'est pas valide. 1 indique que la transmission doit être effectuée dans un intervalle équivalent à l'écart entre les trames, ce qui permet d'obtenir le plus petit intervalle de transmission de trame Ethernet. 2 indique que la transmission doit être effectuée dans un intervalle d'une trame ajoutée à token-PDU plus l'écart entre les trames avant et après cette trame. Si la valeur spécifiée est égale ou supérieure à 2, elle indique que la transmission doit être effectuée, par rapport aux token-PDU, dans un intervalle d'un certain nombre de trames équivalent au nombre spécifié moins un, plus l'écart entre les trames avant et après chaque trame.

5.3.5.7.3 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.5.7.4 multipleTokens

Ce champ spécifie le nombre de fois où la transmission de token-PDU a lieu au cours d'une période de détention de jeton. La plage de valeurs est 1..255. 0 n'est pas valide et ne doit pas être utilisé.

NOTE Par exemple, si une station à appareil intelligent, qui spécifie 2 dans ce champ, devient le nœud qui détient le jeton, la station transmet myStatus-PDU, cyclicRWr-PDU, cyclicRX-PDU, transient1-PDU et token-PDU, puis transmet à nouveau le token-PDU.

5.3.5.8 reserved3

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.5.9 dcs

Voir 5.3.2.7.

5.3.6 SetupAck-PDU

5.3.6.1 falArHeader

Voir 5.3.1.

5.3.6.2 **slaveNodeInfo**

La signification de chaque bit est la suivante:

Bits 7..2:	Réservé à un usage futur. La valeur est 0.
Bits 1..0:	Spécifie le type d'E/S. 00 = mixte (l'entrée et la sortie partagent la même adresse) 01 = entrée 10 = sortie 11 = composite (entrée et sortie avec adresses uniques)

5.3.6.3 **fwVersion**

Ce champ spécifie la version de micrologiciel ("firmware").

5.3.6.4 **DeviceType**

Ce champ spécifie le type d'appareil.

5.3.6.5 **reserved1**

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.6.6 **vendorCode**

Voir 5.3.2.4.

5.3.6.7 **modelCode**

Voir 5.3.2.5.

5.3.6.8 **rySize**

Ce champ spécifie la taille RY (nombre d'octets).

5.3.6.9 **rwwSize**

Ce champ spécifie la taille RWw (nombre de mots).

5.3.6.10 **rxSize**

Ce champ spécifie la taille RX (nombre d'octets).

5.3.6.11 **rwrSize**

Ce champ spécifie la taille RWr (nombre de mots).

5.3.6.12 **reserved2**

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.6.13 availableFuncs

La signification de chaque bit est la suivante:

- Bits 7..2: Réservé à un usage futur. La valeur est 0.
- Bit 1: Indique la présence de la fonction de réglage du numéro de nœud dans parameter-PDU. 0 indique que la fonction n'est pas disponible; 1 indique que la fonction est disponible. Pour le réglage du numéro de nœud, parameter-PDU ne doit pas être envoyé à un nœud qui ne dispose pas de la fonction nécessaire.
- Bit 0: Indique la présence de la fonction de réception transitoire. 0 indique que la fonction n'existe pas; 1 indique que la fonction existe.

5.3.6.14 reserved3

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.6.15 dcs

Voir 5.3.2.7.

5.3.7 F-Token-PDU

5.3.7.1 falArHeader

Voir 5.3.1.

5.3.7.2 tokenDstMacAddr

Ce champ contient l'adresse MAC du nœud de destination de transmission de token-PDU. Chaque nœud évalue si tokenDstMacAddr et l'adresse MAC de son propre nœud correspondent lorsque token-PDU est reçu et, si tel est le cas, évalue si token-PDU est adressé à son propre nœud et devient le nœud détenteur du jeton.

5.3.7.3 tokenSeqNumber

Ce champ contient le numéro de séquence de token-PDU. La plage de valeurs s'étend de 1 à 255.

Le gestionnaire de contrôle de transmission attribue un numéro de séquence différent à chaque passage de jeton. Chaque nœud rejette le deuxième token-PDU et les suivants dont les valeurs tokenSeqNumber sont identiques si des token-PDU présentant la même valeur tokenSeqNumber sont reçus. Dans les nœuds autres que le gestionnaire de contrôle de transmission, les valeurs ne doivent pas être modifiées. Dans les nœuds autres que le gestionnaire de contrôle de transmission, le token-PDU présentant la même valeur que la valeur tokenSeqNumber du token-PDU reçu est transmis.

5.3.7.4 reserved1

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.7.5 tokenHopCounter

Ce champ contient la valeur du compteur de passage de jeton.

La valeur tokenHopCounter du token-PDU à transmettre par le gestionnaire de contrôle de transmission, devenu le nœud détenteur du jeton, doit être 1. Un nœud autre que le gestionnaire de contrôle de transmission, devenu le nœud détenteur de jeton, doit ajouter 1 à la valeur tokenHopCounter du token-PDU reçu, puis procéder à la transmission.

5.3.7.6 traAvailHopCounter

Ce champ spécifie la valeur minimale du compteur de passage de jeton.

Le nœud détenteur du jeton utilise ce champ et la valeur tokenHopCounter pour évaluer si une transmission transitoire est possible. Le nœud détenteur du jeton est capable de procéder à une transmission transitoire si $\text{tokenHopCounter} \geq \text{traAvailHopCounter}$ et $\text{traAllows} > 0$.

Après le début du passage du jeton, le gestionnaire de contrôle de transmission, qui devient le nœud détenteur de jeton pour la première fois, transmet le token-PDU qui spécifie le premier nœud à devoir procéder à une transmission transitoire dans traAvailHopCounter. Si le gestionnaire de contrôle de transmission devient le nœud détenteur de jeton pour la deuxième fois ou plus, le gestionnaire de contrôle de transmission transmet le token-PDU qui définit la même valeur que le traLastHopCounter du token-PDU reçu lors du précédent passage de jeton dans le traAvailHopCounter du token-PDU.

Un nœud autre que le gestionnaire de contrôle de transmission, devenu le nœud détenteur de jeton, transmet le token-PDU qui présente la même valeur que la valeur de traAvailHopCounter du token-PDU reçu.

5.3.7.7 traLastHopCounter

Ce champ indique le dernier compteur de passage de jeton de transmission transitoire. Il s'agit de la valeur tokenHopCounter du nœud détenteur de jeton pour lequel traAllows a été défini sur 0 après l'exécution de la transmission transitoire.

Si le nœud détenteur de jeton procède à une transmission transitoire et si traAllows passe à 0, le nœud doit transmettre le token-PDU dans lequel la valeur tokenHopCounter du token-PDU reçu est définie dans traLastHopCounter.

5.3.7.8 traAllows

Ce champ spécifie le nombre de fois où la transmission transitoire peut avoir lieu. Le nombre est équivalent au nombre de transmissions de trames de transmission transitoire autorisé par le nœud détenteur du jeton.

Le nœud détenteur de jeton ne doit pas transmettre le transientData-PDU si la valeur traAllows du token-PDU reçu est 0. Après la transmission de la valeur transientData-PDU, le nœud détenteur de jeton doit définir, dans l'élément traAllows du token-PDU à transmettre, une valeur équivalente à la valeur moins le nombre de fois où transientData-PDU a été transmis.

5.3.7.9 reserved2

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.7.10 dcs

Voir 5.3.2.7.

5.3.8 F-Measure-PDU

5.3.8.1 falArHeader

Voir 5.3.1.

5.3.8.2 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.8.3 dcs

Voir 5.3.2.7.

5.3.9 F-Offset-PDU**5.3.9.1 falArHeader**

Voir 5.3.1.

5.3.9.2 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.9.3 SyncOffset

Ce champ contient la valeur mesurée du retard associé au chemin de transmission.

5.3.9.4 reserved2

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.9.5 dcs

Voir 5.3.2.7.

5.3.10 F-Update-PDU**5.3.10.1 falArHeader**

Voir 5.3.1.

5.3.10.2 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.10.3 syncOffset

Voir 5.3.9.3.

5.3.10.4 reserved2

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.10.5 dcs

Voir 5.3.2.7.

5.3.11 F-MyStatus-PDU**5.3.11.1 falArHeader**

Voir 5.3.1.

5.3.11.2 seqNumber

Ce champ contient un numéro de séquence qui représente l'ordre de transmission des éléments myStatus-PDU et f-cyclicData-PDU à transmettre une fois par le nœud. La signification de chaque bit est la suivante:

- Bit 7: Identification de la dernière trame. 0 indique qu'il existe des trames suivantes; 1 indique qu'il n'existe pas de trame suivante.
- Bits 6..0: Numéro de séquence. La valeur de myStatus-PDU est 0.

5.3.11.3 netNumber

Ce champ contient le numéro de réseau du nœud. La plage est 1..239.

5.3.11.4 masterCmd

La signification de chaque bit est la suivante:

- Bits 15..12: Réserve à un usage futur. La valeur est 0.
- Bit 11: Indique l'exigence de remise de table d'adresses. 0 indique que la remise est exigée. Ce champ est utilisé uniquement dans myStatus-PDU. Pour les stations principales, cette valeur est 0.
- Bit 10..8: Numéro de séquence de transient1-PDU pour la remise de table d'adresses. Ce champ est utilisé uniquement pour myStatus-PDU transmis par les stations subordonnées, à l'exclusion des stations locales. Pour les stations principales et locales, cette valeur est 0.
- Bit 7: Réserve à un usage futur. La valeur est 0.
- Bit 6: Réserve à un usage futur. La valeur est 0.
- Bit 5: Etat d'erreur d'application. 0 indique l'absence d'erreur; 1 indique l'existence d'une erreur. Utilisation uniquement avec myStatus-PDU transmis par la station principale. Pas d'utilisation et définition sur 0 pour les stations subordonnées.
- Bit 4: Etat de fonctionnement d'application. 0 indique l'arrêt et 1 l'exécution. Utilisation uniquement avec myStatus-PDU transmis par la station principale. La station subordonnée utilise 0.
- Bit 3: Instruction de fonctionnement cyclique. 0 indique une instruction d'exécution et 1 une instruction d'arrêt. Utilisation uniquement avec MyStatus-PDU transmis par la station principale. Utilisation par la station principale pour attribuer une instruction d'exécution cyclique aux stations subordonnées. Une station subordonnée qui reçoit l'instruction d'arrêt rejette cyclicDataRWw-PDU et cyclicDataRY-PDU reçus et arrête la transmission de cyclicDataRWw-PDU et cyclicDataRX-PDU. Les stations subordonnées n'utilisent pas ce champ.
- Bits 2 ..1: Réserve à un usage futur. La valeur est 0.
- Bit 0: Identification de la station principale. 0 indique que le nœud n'est pas une station principale; 1 indique que le nœud est une station principale.

5.3.11.5 cyclicStatus

Si la valeur du bit 8 est 1 et la valeur des bits 9 et 10 est 0, la transmission cyclique et la réception sont exécutées. La signification de chaque bit est la suivante:

- Bit 15: Etat d'instruction de fonctionnement cyclique, spécifique au nœud. 0 indique l'exécution; 1 indique l'arrêt. Utilisation uniquement avec myStatus-PDU transmis par la station subordonnée. La station subordonnée reflète le bit 19 de la commande parameter-PDU reçue. La station principale n'utilise pas ce champ.
- Bit 14: Etat d'instruction de fonctionnement cyclique, tous les nœuds. 0 indique le fonctionnement; 1 indique l'arrêt. Utilisation uniquement avec myStatus-PDU transmis par la station subordonnée. La station subordonnée reflète le bit 3 de la commande myStatus-PDU masterCmd reçue de la station principale. La station principale n'utilise pas ce champ.

- Bit 13: Etat du réglage du nœud réservé. 0 indique que le nœud n'est pas un nœud réservé; 1 indique que le nœud est un nœud réservé. Utilisation uniquement avec myStatus-PDU transmis par la station subordonnée. La station subordonnée reflète la valeur de réglage du bit 16 de la commande parameter-PDU reçue. La station principale n'utilise pas ce champ.
- Bit 12: Etat du réglage du numéro de nœud. 0 indique que le numéro est compris dans la plage; 1 indique que le numéro est hors de la plage. Utilisation uniquement avec myStatus-PDU transmis par la station subordonnée. La station subordonnée reflète le bit 18 de la commande myStatus-PDU reçue. La station principale n'utilise pas ce champ.
- Bit 11: Etat de confirmation du paramètre de transmission cyclique. 0 indique que l'état est confirmé; 1 indique que la confirmation est en cours.
- Bits 10 ..8: Etat de maintien du paramètre de transmission cyclique. Les valeurs utilisées sont conformes au Tableau 37.
- Bit 7: Etat d'arrêt pour raisons propres. 0 indique l'absence d'arrêt; 1 indique l'arrêt de la transmission cyclique pour des raisons autres que celles mentionnées ci-avant.
- NOTE Par exemple, si une demande d'arrêt est reçue de la couche application ou au démarrage.
- Bit 6: Etat de déconnexion. 0 indique l'absence de déconnexion; 1 indique la déconnexion. La station principale n'utilise pas ce champ.
- Bit 5: Réservé à un usage futur. Utilise 0
- Bit 4: Etat invalide du type/numéro de nœud. 0 indique que l'état est valide; 1 indique que l'état est invalide. Utilisation uniquement avec myStatus-PDU transmis par la station subordonnée. Absence d'utilisation et définition sur 0 pour la station principale.
- Bit 3: Etat de duplication de la station principale. 0 indique l'absence de duplication; 1 indique la duplication. Utilisation uniquement avec myStatus-PDU transmis par la station subordonnée. La station subordonnée n'utilise pas ce champ.
- Bit 2: Etat de duplication du numéro de nœud. 0 indique l'absence de duplication; 1 indique la duplication. Utilisation uniquement avec myStatus-PDU transmis par la station subordonnée. La station subordonnée reflète le bit 17 de la commande parameter-PDU reçue. La station principale n'utilise pas ce champ.
- Bit 1: Erreur d'impossibilité de poursuivre la transmission cyclique. 0 indique l'absence d'erreur; 1 indique l'existence d'une erreur qui empêche de poursuivre la transmission cyclique.
- NOTE Par exemple, une erreur matériel ou une erreur micrologiciel sur son propre nœud.
- Bit 0: Réservé à un usage futur. La valeur est 0.

Tableau 37 – État de maintien du paramètre de transmission cyclique

Valeur	Description
001b	Reçu. Paramètre normal.
010b	Non reçu ou absence de correspondance d'ID.
011b	Confirmation en cours.
100b	Reçu. Erreur de paramètre.

5.3.11.6 nodeStatus

La signification de chaque bit est la suivante:

- Bits 15..12: Pour usage futur. La valeur du bit 15 et des bits 13..12 est 0.
- Bits 11..10: Etat d'erreur d'application détaillé. Les valeurs sont définies dans le Tableau 39.
- Bits 9..8: Etat de fonctionnement d'application détaillé. La station principale et la station locale utilisent les valeurs conformément au Tableau 38. Les bits sont en option pour les stations subordonnées (à l'exception de la station locale) et doivent être définis sur 0 en cas de non-utilisation.
- Bit 7: Réservé à un usage futur.
- Bit 6: Etat d'activation/désactivation de réception transitoire. 0 indique la désactivation de la réception; 1 indique l'activation de la réception.
- Bits 5..4: Réservé à un usage futur. La valeur est 0.

- Bit 3: Etat d'erreur de taille. 0 indique que la taille est normale; 1 indique une erreur de taille. Non utilisé et toujours défini sur 0 pour la station principale et la station locale. Les stations subordonnées(à l'exception de la station locale) indiquent une erreur de valeur si les données adressées à leur propre nœud ne sont pas incluses dans le CyclicDataRY-PDU ou le CyclicDataRWw-PDU reçu. La valeur est toujours définie sur 0 pour les stations subordonnées(à l'exception de la station locale) qui ne présentent pas de RY ou de RWw.
- Bits 2..0: Réserve à un usage futur. La valeur est 0.

Tableau 38 – État de fonctionnement d'application détaillé

Valeur	Description
0	Notification d'état de fonctionnement d'application détaillé non prise en charge
1	Application arrêtée
2	Application en cours d'exécution
3	Utilisateur de l'application inexistant

Tableau 39 – État de détection d'erreur

Valeur	Description
0	Pas d'erreur
1	Erreur mineure
2	Erreur majeure
3	Erreur grave

5.3.11.7 errorCode

Ce champ contient les codes des erreurs survenues au niveau du nœud.

5.3.11.8 portStatus

Ce champ indique l'état des quatre ports à l'aide de quatre bits chacun. Les bits 3..0 et les bits 7..4 du premier octet indiquent l'état du premier port et l'état du deuxième port, respectivement; les bits 3..0 et les bits 7..4 du deuxième octet indiquent l'état du troisième port et l'état du quatrième port, respectivement. Le premier port est spécifié dans portIndex. La signification de chaque bit du premier octet est la suivante:

- Bits 7..6: Réserve à un usage futur. La valeur est 0.
- Bits 5..4: Les valeurs utilisées sont conformes au Tableau 35.
- Bits 3..2: Réserve à un usage futur. La valeur est 0.
- Bits 1..0: Les valeurs utilisées sont conformes au Tableau 35.

5.3.11.9 portStatistics

Ce champ contient les informations statistiques des quatre ports, avec quatre bits chacun. Les bits 3..0 et les bits 7..4 du premier octet indiquent l'état du premier port et l'état du deuxième port, respectivement; les bits 3..0 et les bits 7..4 du deuxième octet indiquent l'état du troisième port et l'état du quatrième port, respectivement. Le premier port est spécifié dans portIndex. La signification de chaque bit du premier octet est la suivante. La valeur des informations statistiques d'un port non existant est 0.

- Bit 7, 3: Réservé à un usage futur. La valeur est 0.
- Bit 6, 2: Indique la présence d'une erreur de temporisation de la réception d'une token-PDU. 0 indique l'absence d'erreur; 1 indique une erreur. Si le nœud reçoit une token-PDU adressé à son propre nœud, commence la transmission tout en maintenant le jeton, puis reçoit une token-PDU présentant la même valeur tokenSeqNumber, une erreur de temporisation de réception a lieu. L'erreur est annulée si une token-PDU avec une valeur tokenSeqNumber différente est reçu ou si le nœud passe à un état indéterminé de ChannelGroup.
- Bit 5, 1: Indique la détection d'une duplication d'autorité de transmission. 0 indique l'absence de détection; 1 indique une détection. Une détection a lieu si le nœud détient le jeton et si une PDU autre que persuasion-PDU, testData-PDU, testDataAck-PDU, setup-PDU, setupAck-PDU et token-PDU est reçue.
- Bit 4, 0: Indique la présence d'une erreur de réception. 0 indique l'absence d'erreur; 1 indique une erreur. Une erreur est indiquée si une erreur FCS, une erreur de taille inférieure à la réception ou une erreur de taille supérieure à la réception a lieu.

5.3.11.10 portIndex

Ce champ spécifie le numéro de port du premier port de portStatus et portStatistics.

5.3.11.11 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.11.12 cyclicSequenceNumber

Ce champ est utilisé uniquement avec myStatus-PDU transmis par les stations locales. La station principale et la station subordonnée (pas la station locale) n'utilisent pas les bits et les définissent sur 0. Les bits sont définis comme suit:

- Bit 7: Identificateur de début. 1 indique le début de la transmission cyclique. 0 indique que la transmission cyclique est en cours.
- Bits 6..0: Diviseur de numéro de séquence. Pour la valeur 0, le numéro de séquence n'est pas divisé. Un numéro de séquence divisé est déterminé par le biais d'une soustraction séquentielle, en commençant par la valeur du nombre de divisions. Par exemple, un diviseur de 3 donne la séquence de numéros suivante: 3, 2, 1.

5.3.11.13 SlaveSpfEventInfo1

Ce champ est utilisé avec slaveSpfEventInfo2 si une station subordonnée avertit la station principale d'un événement survenu spécifique aux stations subordonnées. Dans myStatus-PDU transmis par la station principale, le champ indique l'état de réception de l'événement spécifique aux stations subordonnées. Dans myStatus-PDU transmis par une station subordonnée, le champ indique le nombre de fois où l'événement spécifique aux stations subordonnées est survenu.

When the master station transmits the field, the value used is in accordance with Tableau 40. La valeur est 0x00 au démarrage et 0x01 une fois l'initialisation terminée. Si myStatus-PDU est reçu d'une station subordonnée, une vérification est menée à bien pour déterminer si la valeur slaveSpfEventInfo1 reçue est différente de la valeur précédemment reçue. Si tel est le cas, la valeur passe à 0x02. Après l'enregistrement du code détaillé de l'événement spécifique aux stations subordonnées, indiqué par slaveSpfEventInfo2 dans le myStatus-PDU reçu, la valeur passe à 0x01.

Tableau 40 – État de réception d'événement spécifique à une station subordonnée

Valeur	Description
0x00	Etat initial
0x01	En attente de réception
0x02	Réception/enregistrement en cours
0x03..0xFF	Utilisation indisponible

Si une station subordonnée transmet le champ, la valeur est 0x01..0xFF. Si la valeur slaveSpfEventInfo1 du myStatus-PDU reçu de la station principale est 0x01 et si la valeur slaveSpfEventInfo2 a changé par rapport à la valeur précédente, la station subordonnée transmet de nouvelles informations sur l'événement spécifique aux stations subordonnées à la station principale. Si les informations sur l'événement spécifique aux stations subordonnées sont transmises à la station principale, le numéro incrémenté est à nouveau attribué à l'événement spécifique aux stations subordonnées à transmettre et enregistrer. La station subordonnée transmet la valeur myStatus-PDU contenant le nombre d'occurrences attribué à slaveSpfEventInfo1 et le code détaillé de l'événement spécifique aux stations subordonnées à enregistrer dans slaveSpfEventInfo2.

5.3.11.14 SlaveSpfEventInfo2

Ce champ est utilisé avec slaveSpfEventInfo1 si une station subordonnée avertit la station principale d'un événement survenu spécifique aux stations subordonnées.

Dans myStatus-PDU transmis par la station principale, le champ contient la valeur du compteur de réception d'événements spécifiques aux stations subordonnées. La valeur est 0x0000 au démarrage et 0x0001 une fois l'initialisation terminée. La valeur est incrémentée dès la fin du processus de réception et d'enregistrement de l'événement spécifique aux stations subordonnées.

Dans myStatus-PDU transmis par une station subordonnée, la valeur spécifie le code détaillé de l'événement spécifique aux stations subordonnées.

Pour la méthode d'utilisation, voir 5.3.11.13.

5.3.11.15 vendorSpfNodeInfo

Ce champ contient les informations sur le nœud spécifique au fournisseur.

5.3.11.16 dcs

Voir 5.3.2.7.

5.3.12 F-CyclicData-PDU

5.3.12.1 Vue d'ensemble

L'arFType indique si la PDU stocke RWw, RY, RWr ou RX. Dans les descriptions suivantes de RWw, RY, RWr et RX, un arFType de 0x82 (CyclicDataRWw-PDU) fait référence à RWw, 0x83 (CyclicDataRY-PDU) à RY, 0x84 (CyclicDataRWr-PDU) à RWr et 0x85 (CyclicDataRX-PDU) à RX.

5.3.12.2 falArHeader

Voir 5.3.1.

5.3.12.3 seqNumber

Voir 5.3.11.2.

5.3.12.4 bothEndsValidity

Ce champ indique si les quatre premiers octets et les quatre derniers octets de cycData doivent être reflétés dans la mémoire partagée. La signification de chaque bit est la suivante:

Bit 7:	Informations relatives à la validité du dernier octet. 1 indique la validité; 0 indique l'invalidité.
Bit 6:	Informations relatives à la validité du deuxième octet par rapport au dernier. 1 indique la validité; 0 indique l'invalidité.
Bit 5:	Informations relatives à la validité du troisième octet par rapport au dernier. 1 indique la validité; 0 indique l'invalidité.
Bit 4:	Informations relatives à la validité du quatrième octet par rapport au dernier. 1 indique la validité; 0 indique l'invalidité.
Bit 3:	Informations relatives à la validité du quatrième octet par rapport au premier. 1 indique la validité; 0 indique l'invalidité.
Bit 2:	Informations relatives à la validité du troisième octet par rapport au premier. 1 indique la validité; 0 indique l'invalidité.
Bit 1:	Informations relatives à la validité du deuxième octet par rapport au premier. 1 indique la validité; 0 indique l'invalidité.
Bit 0:	Informations relatives à la validité du premier octet. 1 indique la validité; 0 indique l'invalidité.

5.3.12.5 cycDataSize

La signification de chaque bit est la suivante:

Bits 15..12:	Réservé à un usage futur.
Bits 11..0:	Indique la taille de RWw, RY, RWr ou RX. Spécifie la taille en unités de quatre octets. Les quatre premiers octets et les quatre derniers octets à ne pas écrire dans bothEndsValidity sont aussi inclus dans la taille.

5.3.12.6 offsetAddr

Ce champ spécifie l'adresse de décalage à partir du début de RWw, Ry, RWr ou RX. Il est spécifié par une valeur en unités de quatre octets. En d'autres termes, la valeur des bits 1..0 est 0.

5.3.12.7 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.12.8 cycData

Ce champ contient les données de RWw, RY, RWr ou RX. Si les 16 octets ne sont pas remplis, le bourrage est de 0x00.

5.3.12.9 dcs

Voir 5.3.2.7.

5.3.13 Transient1-PDU

5.3.13.1 falArHeader

Voir 5.3.1.

5.3.13.2 traMsgHeader

5.3.13.2.1 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.13.2.2 seqNumber

La signification de chaque bit de ce champ est la suivante:

- Bit 7: Indique si la PDU est la dernière PDU. Une valeur de 0 indique que la PDU n'est pas la dernière PDU. Une valeur de 1 indique que la PDU est la dernière PDU.
- Bits 6..0: Indique le nombre en cas de division des données.

5.3.13.2.3 dataId

Ce champ contient le numéro d'identification des données transitoires. La plage est 0x00..0xFF. Les données transitoires divisées possèdent le même numéro d'identification.

5.3.13.2.4 wholeDataSize

Ce champ spécifie la quantité de données transitoires en unités d'octets.

5.3.13.2.5 offsetAddr

Ce champ spécifie l'adresse de décalage. Le champ est utilisé pour assembler les données transitoires transmises après division. Dans la première PDU, la valeur est 0x00. Dans les PDU suivantes, la position des données vis-à-vis de l'ensemble du volume des données transitoires est spécifiée par une adresse de décalage par rapport au début.

5.3.13.2.6 dataSize

Ce champ spécifie la taille des données transitoires en unités d'octets.

5.3.13.2.7 dataSubType

Ce champ spécifie le sous-type de données. Les valeurs sont spécifiées dans le Tableau 41. Voir 5.3.1.2.

Tableau 41 – dataSupType de dataType (0x07)

Valeur	Description
0x0000	Non applicable
0x0001	Réservé à un usage ultérieur
0x0002	Spécifique à un système
0x0003..0xFFFF	Réservé à un usage ultérieur

5.3.13.3 données

5.3.13.3.1 Vue d'ensemble

Ce champ contient les données transitoires. La structure et la taille diffèrent en fonction du dataType en 5.3.1.2 et du dataSubType en 5.3.13.2.7. Si la valeur wholeDataSize en 5.3.13.2.4 dépasse 1 466 octets, la taille des données transitoires est indiquée par la valeur dataSize en 5.3.13.2.6 à partir du décalage indiqué par la valeur offsetAddr de 5.3.13.2.5. Si les 16 octets ne sont pas remplis, le bourrage est de 0x00.

5.3.13.3.2 FieldSpecificTransient

5.3.13.3.2.1 opHeader

La structure de opHeader est indiquée dans le Tableau 42.

Tableau 42 – FieldSpecificTransient opHeader

Champ	Description
instruction	spécifie le type de commande spécifique au champ. Les valeurs utilisées pour chaque dataType et dataSubType sont conformes au Tableau 43. Pour le dataType, voir 5.3.1.2 et pour le dataSubType, voir 5.3.13.2.7.
subCommand	spécifie le type de sous-commande. Les valeurs pour chaque commande sont conformes au Tableau 44.
rtn	non utilisé, contient une valeur de 0x0000 si le type de sous-commande est "demande" (request). Le champ contient une valeur de retour si le type de sous-commande est "réponse" (response).
reserved1	Réservé à un usage futur. La valeur de chaque octet est 0x00.
destNetNumber	Spécifie le numéro de réseau destination. 0 indique une diffusion. Pour la remise des informations relatives au nœud, la valeur 0 est utilisée.
destNodeNumber	Spécifie le numéro de nœud destination. 0xFFFF indique une diffusion. Pour la remise des informations relatives au nœud, la valeur 0xFFFF est utilisée.
reserved2	Réservé à un usage futur. La valeur de chaque octet est 0x00.
srcNetNumber	Spécifie le numéro de réseau source de transmission.
srcNodeNumber	Spécifie le numéro du nœud source de transmission.
reserved3	Réservé à un usage futur. La valeur de chaque octet est 0x00.

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

Valeur	Description
0x00	Non applicable
0x01	Remise des informations relatives au nœud (nodeInfoDist)
0x02	Réservé à un usage ultérieur
0x03	Obtention d'informations statistiques (Statistiques)
0x04	Acquisition d'informations détaillées relatives au nœud (nodeInfoDetail)
0x05..0xFF	Réservé à un usage ultérieur

Tableau 44 – Type subCommand pour chaque type de commande

Valeur de Command	Valeur	Description
0x01	0x00	Demande
0x03	0x00	Demande
	0x80	Réponse
0x04	0x00	Demande
	0x80	Réponse

5.3.13.3.2.2 fSTraData

La structure de ce champ diffère en fonction de dataType et de dataSubType. Pour le type de données, voir 5.3.1.2. Pour le sous-type de données, voir 5.3.13.2.7.

La structure de la remise des informations relatives au nœud (Deliver node information) est indiquée dans le Tableau 45.

Tableau 45 – Structure de la "Remise des informations relatives au nœud"

Champ	Description
seqNumber	contient le numéro de séquence de remise.
masterNetNumber	contient le numéro de réseau de la station principale.
masterDeviceType	contient le numéro d'appareil de la station principale.
masterModelCode	contient le numéro de modèle de la station principale. Le code du modèle est spécifique au fournisseur.
masterVendorCode	contient le code du fournisseur de la station principale. Pour le code du fournisseur, voir l'édition sur le profil de l'appareil.
masterNodeType	contient le type de nœud de la station principale.
reserved1	Réservé à un usage futur.
masterMacAddress	contient l'adresse MAC de la station principale.
reserved2	Réservé à un usage futur.
dataNum	contient le nombre de remises d'informations relatives au nœud.
message	contient les informations relatives au nœud telles que spécifiées dans le Tableau 46. Les informations relatives au nœud dont la grandeur est indiquée dans dataNum sont continues.

Tableau 46 – Structure de la remise des informations relatives au nœud – message

Champ	Description
nodeNumber	contient le numéro de nœud.
reserved1	Réservé à un usage futur. La valeur de chaque octet est 0x00.
availableFuncs	Voir 5.3.6.13.
reserved2	Réservé à un usage futur. La valeur de chaque octet est 0x00.
netNumber	contient le numéro de réseau.
DeviceType	contient le type d'appareil.
modelCode	Voir 5.3.2.5.
vendorCode	Voir 5.3.2.4.
nodeType	contient le type de nœud.
reserved3	Réservé à un usage futur.

Champ	Description
MACAddress	contient l'adresse MAC.
reserved4	Réservé à un usage futur. La valeur de chaque octet est 0x00.

Il n'existe aucun champ pour la demande d'obtention d'informations statistiques. La structure de l'obtention d'informations statistiques (Get statistical information) est indiquée dans le Tableau 47.

Tableau 47 – Structure de la réponse à une demande d'obtention d'informations statistiques (Get statistical information response)

Champ	Description
port1Mib1	Indique le nombre de trames en erreur HEC dans le port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib2	indique le nombre de trames d'erreur DCS/FCS du port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib3	indique le nombre de trames d'erreur de taille inférieure (28 octets) du port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib4	indique le nombre de trames transférées du port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib5	indique le nombre de trames remises par le port 1 à la couche supérieure. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib6	indique le nombre de trames rejetées en raison d'un tampon de transfert plein dans le port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port1Mib7	indique le nombre de trames remises à la couche supérieure et rejetées en raison d'un tampon plein dans le port 1. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
réservé	Réservé à un usage futur. La valeur de chaque octet est 0x00.
port2Mib1	Indique le nombre de trames en erreur HEC dans le port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib2	indique le nombre de trames d'erreur DCS/FCS du port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib3	indique le nombre de trames d'erreur de taille inférieure (28 octets) du port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib4	indique le nombre de trames transférées du port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib5	indique le nombre de trames remises par le port 2 à la couche supérieure. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib6	indique le nombre de trames rejetées en raison d'un tampon de transfert plein dans le port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
port2Mib7	indique le nombre de trames remises à la couche supérieure et rejetées en raison d'un tampon plein dans le port 2. Le nombre est le nombre accumulé depuis les précédentes acquisitions.
healthStatusNum	indique le nombre de données d'état de robustesse. La plage de valeurs est 0..128.
healthStatus	indique l'état de robustesse. Les informations sur l'état de robustesse dont le nombre est indiqué dans healthStatusNum sont continues. Cette valeur est spécifique à un fournisseur.

Il n'existe aucun champ pour la demande d'acquisition des détails du nœud. Les champs pour la réponse à la demande d'acquisition des détails du nœud (Acquisition of node details response) sont spécifiés dans le Tableau 48.

**Tableau 48 – Structure de la réponse à une demande d'acquisition des détails du nœud
(Acquisition of node details response)**

Champ	Description
rySize	Voir 5.3.6.8.
rwwSize	Voir 5.3.6.9.
rxSize	Voir 5.3.6.10.
rwrSize	Voir 5.3.6.11.
reserved1	Réservé à un usage futur. La valeur de chaque octet est 0x00.
ports	Indique le nombre de ports.
tokenKeepTime	Voir 5.3.4.7.
netBehaviour	Voir 5.3.5.7.
nodeInfo	Voir 5.3.6.2.
fwVersion	Indique la version du micrologiciel ("firmware") du réseau.
DeviceType	contient le type d'appareil.
modelCode	contient le code du modèle.
vendorCode	contient le code du fournisseur.
reserved2	Réservé à un usage futur. La valeur de chaque octet est 0x00.
modelName	contient le nom du modèle.
vendorName	contient le nom du fournisseur.
contInfo	contient l'indicateur d'informations. 1 indique que les informations pertinentes sont contenues dans les champs qui suivent. 0 indique qu'aucune information pertinente ne suit.
contFwVersion	contient la version du micrologiciel ("firmware") du contrôleur.
contDeviceType	contient le type d'appareil du contrôleur.
contModelCode	contient le code du modèle de contrôleur.
contVendorCode	contient le code du fournisseur du contrôleur.
reserved3	Réservé à un usage futur. La valeur de chaque octet est 0x00.
contModelName	contient le nom du modèle du contrôleur.
contVendorName	contient le nom du fournisseur du contrôleur.
contVendorSpecificInfo	contient les informations sur l'appareil spécifiques à un fournisseur.
contAilableFuncs	contient la disponibilité du mode étendu de transmission – 0x00: le mode de transmission étendu n'est pas disponible. – 0x01: le mode de transmission étendu est disponible.

5.3.14 TransientAck-PDU

5.3.14.1 falArHeader

Voir 5.3.1.

5.3.14.2 acks

Ce champ spécifie le nombre de ackData. La valeur est 1.

5.3.14.3 ackData

5.3.14.3.1 nodeNumber

Ce champ contient le numéro de nœud.

5.3.14.3.2 reserved1

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.14.3.3 connectionInfo

Ce champ contient les informations de connexion du f-transientData-PDU reçu. La plage de valeurs est 0x01..0xFF.

5.3.14.3.4 dataSubType

Si le type de trame est 0x22, le sous-type de données reçu est défini. Voir 5.3.13.2.7. Si le type de trame est 0x25, la valeur 0x0000 est fixée.

5.3.14.3.5 ret

Ce champ indique le résultat de la réponse à la demande du f-transientData-PDU reçu. 0 indique un résultat normal; une valeur autre que 0 indique le code d'erreur en cas d'anomalie.

5.3.14.4 dcs

Voir 5.3.2.7.

5.3.15 Transient2-PDU

5.3.15.1 falArHeader

Voir 5.3.1.

5.3.15.2 l

Ce champ spécifie la longueur des données de fno à data (en octets).

5.3.15.3 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.15.4 tp

La signification de chaque bit de ce champ est la suivante:

Bits 7..4: Indique le type. La valeur est 0.

Bits 3..0: Indique le numéro de séquence.

5.3.15.5 fno

La signification de chaque bit de ce champ est la suivante:

- Bit 7: Indique l'identification de la première trame. Une valeur de 0 indique que la trame est une trame sans début. Une valeur de 1 indique que la trame est une trame support.
- Bits 6..0: Indique le numéro de trame divisée. Une valeur de 0 indique l'absence de division. Une valeur de 1 à 7 indique le numéro de trame divisée. Le numéro de trame divisée diminue de façon séquentielle, en commençant par le même numéro que le nombre de divisions.
Exemple Si une trame est divisée en trois, les numéros de trame divisée sont envoyés dans l'ordre suivant: 3, 2, 1.

5.3.15.6 dt

La signification de chaque bit de ce champ est la suivante:

- Bit 7: Indique la priorité. 0 indique une priorité basse; 1 indique une priorité haute.
- Bit 6: Indique la présence d'une trame de réponse. Si la valeur est 0, cela signifie qu'une trame de réponse est exigée. Si la valeur est 1, cela signifie qu'aucune trame de réponse n'est nécessaire.
- Bits 5 à 0 Réserve à un usage futur.

5.3.15.7 da

Ce champ spécifie le numéro de nœud du nœud qui procède au transfert dans le réseau local si le nœud de destination se trouve dans un réseau différent.

5.3.15.8 sa

Ce champ indique le numéro de nœud du nœud source.

5.3.15.9 dat

Ce champ spécifie le type d'application de destination. La valeur est fixée à 0x22.

5.3.15.10 sat

Ce champ indique le type d'application source. La valeur est fixée à 0x22.

5.3.15.11 dmf

Ce champ spécifie la destination du module d'exécution. Les valeurs utilisées sont conformes au Tableau 49.

Tableau 49 – Spécification du module d'exécution

Valeur	Description
0x00	Au sein du module
0x01	Au sein du contrôleur
0x02	Réserve à un usage ultérieur
0x03..0xFF	Au sein d'un autre module

5.3.15.12 smf

Ce champ spécifie la source du module d'exécution. Les valeurs utilisées sont conformes au Tableau 49.

5.3.15.13 dna

Ce champ spécifie le numéro de réseau du nœud de destination. La plage de valeurs est 0x00..0xEF et 0xFE. 0x00 indique qu'aucun réseau n'est spécifié; 0xFE indique le réseau par défaut.

5.3.15.14 ds

Ce champ spécifie le numéro de nœud du nœud de destination cible de l'autre réseau pendant le transfert.

5.3.15.15 did

La signification de chaque bit de ce champ est la suivante:

Bits 15..10:	Zone de spécification du système. Stocke le numéro du nœud de destination cible.
Bits 9..0:	Indique le numéro d'identification de destination cible.

5.3.15.16 sna

Ce champ indique le numéro de réseau du nœud source de démarrage. La plage de valeurs est 0x00..0xEF et 0xFE. 0x00 indique qu'aucun réseau n'est spécifié; 0xFE indique la spécification du réseau par défaut.

5.3.15.17 ss

Ce champ indique le numéro de nœud du nœud source de transmission de l'autre réseau pendant le transfert.

5.3.15.18 sid

La signification de chaque bit de ce champ est la suivante:

Bits 15..10:	Zone de spécification du système. Stocke le numéro du nœud source de démarrage.
Bits 9..0:	Indique le numéro d'identification source de démarrage.

5.3.15.19 l1

Ce champ spécifie la longueur des données (en unités d'octets) de ct à data.

5.3.15.20 ct

Ce champ spécifie le type de commande. Les valeurs utilisées sont conformes au Tableau 50.

Tableau 50 – Type de commande

Valeur	Description
0x00	Non applicable
0x01..0x03	Réservé à un usage ultérieur
0x04	Obtenir info d'accès mémoire
0x05..0x07	Réservé à un usage ultérieur
0x08	Run (Exécution)
0x09	STOP (Arrêt=
0x0A..0x0F	Réservé à un usage ultérieur

Valeur	Description
0x10	Lire mémoire
0x11	Réservé à un usage ultérieur
0x12	Ecriture dans la mémoire
0x13..0x5F	Réservé à un usage ultérieur
0x60..0x7F	Propre au fournisseur

5.3.15.21 rsv

Réservé à un usage futur.

5.3.15.22 aps

La signification de chaque bit de ce champ est la suivante:

- Bits 15..11: Indique le numéro de tâche attribué à la tâche. La plage de valeurs est 0..255.
- Bits 8..0: Indique le numéro d'identification de l'application source de démarrage. La plage de valeurs est 0..255.

5.3.15.23 rsts

Ce champ est utilisé pour les réponses uniquement. La signification de chaque bit est la suivante:

- Bits 15..12: Définition du fournisseur
- Bits 11..8: Emplacement d'occurrence d'erreur
- Bit 7: Criticité d'erreur. 0 indique une erreur d'avertissement; 1 indique une erreur majeure.
- Bits 6..0: Code d'erreur

5.3.15.24 données

Voir 5.2.12.23.

5.3.15.25 dcs

Voir 5.3.2.7.

5.3.16 ParamCheck-PDU

5.3.16.1 falArHeader

Voir 5.3.1.

5.3.16.2 reserved1

Il est réservé à un usage ultérieur. La valeur des premier, deuxième et quatrième octets est 0x00. La valeur des bits 0..3 du troisième octet est 0.

5.3.16.3 paramId

5.3.16.3.1 Vue d'ensemble

Ce champ contient l'identification du paramètre. Une valeur de 0 indique une instruction de rejet de paramètre.

5.3.16.3.2 date

La signification de chaque bit des éléments de ce champ est la suivante:

Bits 31..28	Jour (dizaines)
Bits 27..24	Jour (unités)
Bits 23..20	Mois (dizaines)
Bits 19..16	Mois (unités)
Bits 15..12	Année (dizaines)
Bits 11..8	Année (unités)
Bits 7..4	Année (milliers)
Bits 3..0	Année (centaines)

5.3.16.3.3 timeNodeId

La signification de chaque bit des éléments de ce champ est la suivante:

Bits 31..24	Numéro de nœud de source de réglage
Bits 23..20	Secondes (dizaines)
Bits 19..16	Secondes (unités)
Bits 15..12	Minutes (dizaines)
Bits 11..8	Minutes (unités)
Bits 7..4	Heure (dizaines)
Bits 3..0	Heure (unités)

5.3.16.3.4 Somme de contrôle

Ce champ contient la valeur sum-check des paramètres communs détenus par la station principale.

5.3.16.4 reserved3

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.16.5 dcs

Voir 5.3.2.7.

5.3.17 Parameter-PDU

5.3.17.1 falArHeader

Voir 5.3.1.

5.3.17.2 paramSetFlag

Ce champ contient la spécification du paramètre. La signification de chaque bit est la suivante:

Bits 7..3:	Réservé à un usage futur. La valeur est 0.
Bit 2:	Réglage du paramètre commun. 0 indique que le paramètre n'est pas valide; 1 indique que le paramètre est valide.
Bit 1:	Réglage de la commande de fonctionnement. 0 indique que le paramètre n'est pas valide; 1 indique que le paramètre est valide.
Bit 0:	Réglage du numéro de nœud et du numéro de réseau. 0 indique que le paramètre n'est pas valide; 1 indique que le paramètre est valide.

5.3.17.3 addressOrder

5.3.17.3.1 Vue d'ensemble

Ce champ est utilisé si le bit 0 de paramSetFlag indique un paramètre valide. Le champ n'est pas utilisé et la valeur est 0 si paramSetFlag indique un paramètre invalide.

5.3.17.3.2 assignedNetNumber

Ce champ spécifie la valeur de réglage du numéro de réseau. La plage est 1..239.

5.3.17.3.3 assignedNodeNumber

Ce champ spécifie la valeur de réglage du numéro de nœud.

5.3.17.4 cmdOrder

5.3.17.4.1 Vue d'ensemble

Ce champ est utilisé si le bit 1 de paramSetFlag indique un paramètre valide. Le champ n'est pas utilisé et la valeur est 0 si paramSetFlag indique un paramètre invalide.

5.3.17.4.2 cmd

La signification de chaque bit des éléments de ce champ est la suivante:

Bits 23..20:	Réservé à un usage futur. La valeur est 0.
Bit 19:	Instruction d'arrêt de transmission cyclique causée par l'évaluation de la station principale. 0 indique l'activation; 1 indique la désactivation. La station principale envoie cette instruction aux stations subordonnées. La station subordonnée reflète l'état d'instruction dans le bit 15 du cyclicStatus du myStatus-PDU.
Bit 18:	Instruction d'arrêt de transmission cyclique causée par un numéro de nœud invalide. 0 indique l'activation; 1 indique la désactivation. La station principale envoie cette instruction aux stations subordonnées. La station subordonnée reflète l'état d'instruction dans le bit 3 du cyclicStatus du myStatus-PDU.
Bit 17:	Instruction d'arrêt de transmission cyclique causée par un numéro de nœud dupliqué. 0 indique l'activation; 1 indique la désactivation. La station principale envoie cette instruction aux stations subordonnées. La station subordonnée reflète l'état d'instruction dans le bit 13 du cyclicStatus du myStatus-PDU.
Bit 16:	Réglage de nœud réservé. 0 indique que le nœud est un nœud réservé; 1 indique que le nœud n'est pas un nœud réservé. La station principale utilise ce réglage en cas de spécification d'une station subordonnée comme nœud réservé. Une station subordonnée spécifiée comme nœud réservé ne procède pas à une transmission cyclique. La station subordonnée reflète l'état de ce réglage dans le bit 2 du cyclicStatus du myStatus-PDU.
Bits 15..5:	Réservé à un usage futur. La valeur est 0.

- Bit 4: Indique si le type de nœud est valide, tel que déterminé par comparaison des champs relatifs au type de nœud, à savoir le champ `nodeType` et le type de station de la station. 0 indique que le type de nœud est invalide et que la transmission cyclique n'est pas effectuée.
- Bits 3..0: Réserve à un usage futur. La valeur est 0.

5.3.17.4.3 `nodeType`

Voir 5.3.1.3.3.

5.3.17.5 `cyclicParameter`

5.3.17.5.1 Vue d'ensemble

Ce champ est utilisé si le bit 2 de `paramSetFlag` indique un réglage. Le champ n'est pas utilisé et la valeur est 0 si `paramSetFlag` indique une absence de réglage.

5.3.17.5.2 `paramId`

Voir 5.3.16.3.

5.3.17.5.3 `reserved1`

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.4 `masterStatus`

La signification de chaque bit des éléments de ce champ est la suivante:

- Bits 15..1: Réserve à un usage futur. La valeur est 0.
- Bit 0: Fonction de garantie de nœud. 0 indique que le nœud n'est pas garanti; 1 indique que le nœud est garanti. Utilisation par la station principale pour indiquer à la station subordonnée si la station principale possède une fonction de garantie de nœud.

5.3.17.5.5 `rySeqNumber`

La signification de chaque bit des éléments de ce champ est la suivante:

- Bit 7: Réserve à un usage futur. La valeur est 0.
- Bits 6..0: Numéro de séquence de `cyclicDataRY-PDU` à recevoir. La plage de valeurs est 1..127.

5.3.17.5.6 `ryBothEndsValidity`

Ce champ indique s'il convient de refléter dans la mémoire partagée les quatre premiers octets et les quatre derniers octets de la zone spécifiée par `ryDataSize` et `ryOffset` dans les valeurs `cycData` du `cyclicDataRY-PDU` présentant le numéro de séquence spécifié par `rySeqNumber`. Si les valeurs `ryDataSize` et `ryOffset` sont spécifiées de sorte que les données chevauchent plusieurs `cyclicDataRY-PDU`, les quatre derniers octets sont inclus dans le dernier `cyclicDataRY-PDU`.

La signification de chaque bit est la suivante:

Bit 7:	Informations du dernier octet activées. 1 indique l'activation; 0 indique la désactivation.
Bit 6:	Informations du deuxième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 5:	Informations du troisième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 4:	Informations du quatrième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 3:	Informations du quatrième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 2:	Informations du troisième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 1:	Informations du deuxième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 0:	Informations de l'octet initial activées. 1 indique l'activation; 0 indique la désactivation.

5.3.17.5.7 ryDataSize

Ce champ spécifie la taille des données à refléter dans la mémoire partagée parmi les valeurs cycData du cyclicDataRY-PDU spécifiées dans rySeqNumber, depuis l'adresse précisée dans ryOffset. La valeur est spécifiée en unités de quatre octets. Les quatre octets initiaux et les quatre derniers octets spécifiés dans ryBothEndsValidity sont aussi inclus dans la taille.

En fonction des valeurs de ryDataSize et ryOffset, la valeur reflétée dans la mémoire partagée, avec le cyclicDataRY-PDU spécifié dans rySeqNumber, il est possible que le début chevauche plusieurs cyclicDataRY-PDU.

5.3.17.5.8 ryOffset

Ce champ spécifie le début de la zone parmi les valeurs cycData du cyclicDataRY-PDU à refléter dans la mémoire partagée, en octets depuis le début de cycData. La valeur est spécifiée en unités de quatre octets.

En fonction des valeurs de ryDataSize et ryOffset, la valeur reflétée dans la mémoire partagée, avec le cyclicDataRY-PDU spécifié dans rySeqNumber, il est possible que le début chevauche plusieurs cyclicDataRY-PDU.

5.3.17.5.9 reserved2

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.10 rwwSeqNumber

Ce champ est défini comme suit:

Bit 7:	Réservé à un usage futur. La valeur est 0.
Bits 6..0:	Numéro de séquence du cyclicDataRww-PDU à recevoir. La plage de valeurs est 1..127.

5.3.17.5.11 reserved3

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.12 rwwDataSize

Ce champ spécifie la taille parmi les valeurs cycData du cyclicDataRY-PDU présentant le numéro de séquence spécifié dans rwwSeqNumber à refléter dans la mémoire partagée à partir de l'adresse spécifiée par rwwOffset. La valeur est spécifiée en unités de quatre octets.

En fonction des valeurs de rwwDataSize et rwwOffset, la valeur reflétée dans la mémoire partagée avec le cyclicDataRWw-PDU spécifié dans rwwSeqNumber, il est possible que le début chevauche plusieurs cyclicDataRWw-PDU.

5.3.17.5.13 rwwOffset

Ce champ spécifie le début de la zone à refléter dans la mémoire partagée parmi les valeurs cycData du cyclicDataRWw-PDU présentant le numéro de séquence spécifié dans rwwSeqNumber, avec un décalage par rapport au début de cycData. La valeur est spécifiée en unités de quatre octets.

En fonction des valeurs de rwwDataSize et rwwOffset, la valeur reflétée dans la mémoire partagée avec le cyclicDataRWw-PDU spécifié dans rwwSeqNumber, il est possible que le début chevauche plusieurs cyclicDataRWw-PDU.

5.3.17.5.14 reserved4

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.15 rxBothEndsValidity

Ce champ spécifie si les quatre premiers octets et les quatre derniers octets de la zone spécifiée par rxDataSize et rxOffset parmi les valeurs cycData du cyclicDataRX-PDU constituent la zone reflétée depuis la mémoire partagée.

La signification de chaque bit est la suivante:

Bit 7:	Informations du dernier octet activées. 1 indique l'activation; 0 indique la désactivation.
Bit 6:	Informations du deuxième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 5:	Informations du troisième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 4:	Informations du quatrième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 3:	Informations du quatrième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 2:	Informations du troisième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 1:	Informations du deuxième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 0:	Informations de l'octet initial activées. 1 indique l'activation; 0 indique la désactivation.

5.3.17.5.16 rxDataSize

Ce champ spécifie la taille des données de cycData du cyclicDataRY-PDU reflétées par la mémoire partagée à l'aide de l'adresse spécifiée dans rxOffset comme début. Les quatre premiers octets et les quatre derniers octets spécifiés dans rxBothEndsValidity sont aussi inclus dans la taille.

5.3.17.5.17 rxOffset

Ce champ spécifie le début de la zone parmi les valeurs cycData du cyclicDataRX-PDU reflété par la mémoire partagée, en octets depuis le début de cycData. La valeur est spécifiée en unités de quatre octets. La valeur des bits 1..0 est 0.

5.3.17.5.18 reserved5

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.19 rwrDataSize

Ce champ indique la taille parmi les valeurs cycData du cyclicDataRY-PDU reflété par la mémoire partagée à l'aide de l'adresse spécifiée par rwrOffset comme début. La valeur est spécifiée en unités de quatre octets.

5.3.17.5.20 rwrOffset

Ce champ spécifie le début de la zone reflétée par la mémoire partagée parmi les valeurs cycData du cyclicDataRWr-PDU, avec un décalage par rapport au début de cycData. La valeur est spécifiée en unités de quatre octets. La valeur des bits 1..0 est 0.

5.3.17.5.21 reserved6

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.22 masterWatchTimer

Ce champ spécifie la valeur de réglage de MasterWatchTimer.

Bit 15: Réservé à un usage futur. 0 est utilisé.

Bits 14..0: Valeur de réglage de MasterWatchTimer. La plage de valeurs est 1..32767. L'unité est 400 µs.

5.3.17.5.23 reserved7

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.24 cmRyBothEndsValidity

Ce champ spécifie s'il convient de refléter dans la mémoire partagée les quatre premiers octets et les quatre derniers octets de la zone spécifiée par cmRyDataSize et cmRyOffset dans les valeurs RY transmises par la station principale. Le champ est utilisé uniquement avec la station locale. Si les valeurs cmRyDataSize et cmRyOffset sont spécifiées de sorte que les données chevauchent plusieurs cyclicDataRY-PDU, les quatre derniers octets sont inclus dans le dernier cyclicDataRY-PDU.

La signification de chaque bit est la suivante:

- Bit 7: Informations du dernier octet activées. 1 indique l'activation; 0 indique la désactivation.
- Bit 6: Informations du deuxième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
- Bit 5: Informations du troisième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
- Bit 4: Informations du quatrième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
- Bit 3: Informations du quatrième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.

- Bit 2: Informations du troisième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
- Bit 1: Informations du deuxième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
- Bit 0: Informations de l'octet initial activées. 1 indique l'activation; 0 indique la désactivation.

5.3.17.5.25 cmRyDataSize

Ce champ spécifie la taille des données à refléter dans la mémoire partagée parmi les valeurs RY transmises par la station principale, depuis l'adresse précisée dans cmRyOffset. Les quatre premiers octets et les quatre derniers octets spécifiés dans cmRyBothEndsValidity sont aussi inclus dans la taille. Ce champ est utilisé par la station locale uniquement.

5.3.17.5.26 cmRyOffset

Ce champ spécifie le début de la zone parmi les valeurs RY transmises par la station principale à refléter dans la mémoire partagée, en octets depuis le début de RY. La valeur est spécifiée en unités de quatre octets. Ce champ est utilisé par la station locale uniquement.

5.3.17.5.27 reserved8

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.28 cmRwwDataSize

Ce champ spécifie la taille parmi les valeurs RWw transmises par la station principale à refléter dans la mémoire partagée, depuis l'adresse précisée dans cmRwwOffset. La valeur est spécifiée en unités de quatre octets. Le champ est utilisé par la station locale uniquement.

5.3.17.5.29 cmRwwOffset

Ce champ spécifie le début de la zone à refléter dans la mémoire partagée parmi les valeurs RWw transmises par la station principale, avec un décalage par rapport au début de RWw. La valeur est spécifiée en unités de quatre octets. Le champ est utilisé par la station locale uniquement.

5.3.17.5.30 reserved9

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.31 cmRxBothEndsValidity

Ce champ spécifie si les quatre premiers octets et les quatre derniers octets de la zone spécifiée par cmRxDataSize et cmRxOffset parmi les valeurs RX à recevoir par la station principale depuis une station subordonnée et construites au niveau de la station principale doivent être reflétés dans la mémoire partagée. Le champ est utilisé par la station locale uniquement. Si les valeurs cmRyDataSize et cmRyOffset sont spécifiées de sorte que les données chevauchent plusieurs cyclicDataRX-PDU, les quatre derniers octets sont inclus dans le dernier cyclicDataRX-PDU.

La signification de chaque bit est la suivante:

Bit 7:	Informations du dernier octet activées. 1 indique l'activation; 0 indique la désactivation.
Bit 6:	Informations du deuxième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 5:	Informations du troisième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 4:	Informations du quatrième octet à partir de la fin activées. 1 indique l'activation; 0 indique la désactivation.
Bit 3:	Informations du quatrième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 2:	Informations du troisième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 1:	Informations du deuxième octet à partir du début activées. 1 indique l'activation; 0 indique la désactivation.
Bit 0:	Informations de l'octet initial activées. 1 indique l'activation; 0 indique la désactivation.

5.3.17.5.32 cmRxDataSize

Ce champ spécifie la taille des données à refléter dans la mémoire partagée parmi les valeurs RX transmises par une station subordonnée à la station principale et construites au niveau de la station principale, depuis l'adresse précisée dans cmRxyOffset. Les quatre premiers octets et les quatre derniers octets spécifiés dans cmRxBothEndsValidity sont aussi inclus dans la taille. Ce champ est utilisé par la station locale uniquement.

5.3.17.5.33 cmRxOffset

Ce champ spécifie le début de la zone à refléter dans la mémoire partagée parmi les valeurs RX transmises par une station subordonnée à la station principale et construites au niveau de la station principale, en octets depuis le début de RX. Le champ est utilisé par la station locale uniquement. La valeur est spécifiée en unités de quatre octets. La valeur des bits 1..0 est 0.

5.3.17.5.34 reserved10

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.17.5.35 cmRwrDataSize

Ce champ spécifie la taille des données à refléter dans la mémoire partagée parmi les valeurs RWr transmises par une station subordonnée à la station principale et construites au niveau de la station principale, depuis l'adresse précisée par cmRwrOffset. Le champ est utilisé par la station locale uniquement.

5.3.17.5.36 cmRwrOffset

Ce champ spécifie le début de la zone à refléter dans la mémoire partagée parmi les valeurs RWr transmises par une station subordonnée à la station principale et construites au niveau de la station principale, avec un décalage par rapport au début de RWr. Ce champ est utilisé par la station locale uniquement. La valeur est spécifiée en unités de quatre octets. La valeur des bits 1..0 est 0.

5.3.17.6 dcs

Voir 5.3.2.7.

5.3.18 Timer-PDU

5.3.18.1 falArHeader

Voir 5.3.1.

5.3.18.2 temps

Ce champ contient la valeur du temporisateur en unités de 15,258 789 062 5 μ s, en utilisant le 1er janvier 2000, 00:00:00 comme point de référence.

5.3.18.3 réservé

Il est réservé à un usage ultérieur. La valeur de chaque octet est 0x00.

5.3.18.4 dcs

Voir 5.3.2.7.

5.4 FALPDU type T – encodage des éléments

5.4.1 CyclicM-PDU

5.4.1.1 cyclicMSHeader

5.4.1.1.1 frameType

Ce champ spécifie le type de trame. La valeur selon le Tableau 51 est utilisée.

Tableau 51 – frameType

Valeur	Description
0x00 à 0xBF	Utilisé dans le réseau d'appareils de commande CC-Link IE et le réseau de terrain CC-Link IE.
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 à 0xFF	Pour une extension ultérieure

5.4.1.1.2 cycleNo

Ce champ représente le numéro de cycle de la transmission cyclique. La valeur selon le Tableau 52 est utilisée.

Tableau 52 – cycleNo

Bit	Valeur	Description
Bit 7	0	Vérifie le numéro de cycle et, si le numéro de cycle n'est pas trouvé dans la trame reçue, effectue le traitement de rejet de la trame
	1	Effectue le traitement de réception quel que soit le numéro de cycle
Bit 6 à 0	0 à 63	Représente un numéro qui identifie une trame à recevoir pendant les cycles

5.4.1.1.3 sa

Ce champ représente le troisième octet et le quatrième octet de l'adresse IP source. La valeur selon le Tableau 53 est utilisée.

Tableau 53 – sa

Bit	Valeur	Description
Bit 15 à 8	0 à 254	3e octet de l'adresse IP source
	255	Pour une extension ultérieure
Bit 7 à 0	0	Pour une extension ultérieure
	1 à 254	4e octet de l'adresse IP source
	255	Pour une extension ultérieure

5.4.1.1.4 réservé

Pour une extension ultérieure. Chaque octet doit être 0x00.

5.4.1.2 subPayloadHeader**5.4.1.2.1 da**

Ce champ représente le troisième octet et le quatrième octet de l'adresse IP de destination. La valeur selon le Tableau 54 est utilisée.

Tableau 54 – da

Bit	Valeur	Description
Bit 15 à 8	0 à 254	3e octet de l'adresse IP de destination
	255	Indique que cette sous-charge utile est transmise à plusieurs stations au moment de la transmission multidiffusion. Mettre à 255 lorsque les données cycliques sont partagées avec la station locale.
Bit 7 à 0	0	Pour une extension ultérieure
	1 à 254	4e octet de l'adresse IP de destination
	255	Indique que cette sous-charge utile est transmise à plusieurs stations au moment de la transmission multidiffusion. Mettre à 255 lorsque les données cycliques sont partagées avec la station locale.

5.4.1.2.2 commInfo

Ce champ représente les informations relatives à la communication (controlFlag, timingErr, reserved, length). La valeur selon le Tableau 55 est utilisée.

Tableau 55 – commInfo

Nom	Bit	Valeur	Description
controlFlag	Bit 15	0	Transmission cyclique désactivée
		1	Transmission cyclique activée
timingErr	Bit 14	0	Erreur de synchronisation de communication non détectée
		1	Erreur de synchronisation de communication détectée
Réservé	Bit 13 à 11	-	Pour une extension future, 0 est utilisé pour chaque bit.
Length (Longueur)	Bit 10 à 0	0 à 2047	Longueur d'octet des données d'application de la sous-charge utile

5.4.1.2.3 txAsynInfo

Ce champ représente les informations relatives à la transmission transitoire de type de médiation. La valeur selon le Tableau 56 est utilisée.

Tableau 56 – txAsynInfo

Bit	Valeur	Description
Bit 7	0	Il existe une demande de transmission
	1	Il n'existe pas de demande de transmission
Bit 6 à 0	0x00	La transmission transitoire n'est pas autorisée
	0x01 à 0x7F	La transmission transitoire est autorisée

5.4.1.3 subPayloadData**5.4.1.3.1 seqNo**

Ce champ représente un numéro séquentiel. La valeur selon le Tableau 57 est utilisée.

Tableau 57 – seqNo

Bit	Valeur	Description
Bit 15	0	Représente la trame en cours de processus de la division
	1	Représente la trame finale de la division
Bit 14 à 0	0 à 32767	Représente un numéro séquentiel Lors d'un envoi par division en plusieurs trames ou sous charge utile, des numéros de série sont utilisés. Il commence à 0 pour chaque cycle, avec incrémentation de 1 chaque fois qu'il y a 1 sous-charge utile

5.4.1.3.2 diagnosisData

Ce champ représente les informations de diagnostic. L'octet supérieur représente l'état d'erreur d'application (micrologiciel), et les trois octets inférieurs représentent l'état d'erreur d'application (matériel). Les valeurs selon le Tableau 58 et le Tableau 59 sont utilisées.

Tableau 58 – L’octet supérieur de diagnosticData

Bit	Valeur	Description
Bit 7	0	Représente l'absence d'erreur d'état d'erreur WDT
	1	Représente la présence d'erreur d'état d'erreur WDT
Bit 6	0	Représente l'absence de déconnexion de vérification d'erreur N/W (notification de déconnexion de la communication entre appareils subordonnés)
	1	Représente la présence d'une déconnexion de vérification d'erreur N/W (notification de déconnexion de la communication entre appareils subordonnés)
Bit 5	-	Pour une extension ultérieure. 0 est utilisé pour la valeur.
Bit 4	0	Non invalide (les données de commande cycliques sont valides.)
	1	Invalide (les données de commande cycliques sont invalides.)
Bit 3	0	Non spécifié (les données de commande cycliques sont valides.)
	1	Spécifié (les données de commande cycliques sont invalides.)
Bit 2	0	Représente l'absence d'erreur d'état d'erreur de la partie réseau
	1	Représente la présence d'une erreur d'état d'erreur de la partie réseau
Bit 1	0	Représente l'absence d'erreur d'état d'erreur d'application
	1	Représente la présence d'une erreur d'état d'erreur d'application
Bit 0	0	Représente le fait que l'opération d'application est en état d'arrêt
	1	Représente le fait que l'opération d'application est en état de fonctionnement

Tableau 59 – Les trois octets inférieurs de diagnosisData

Bit	Valeur	Description
Bit 23 à 8	0 à 65535	Représente le GROUPE EMG
Bit 7 à 4	0 à 15	Représente le GROUPE GOF
Bit 3	-	Pour une extension ultérieure. 0 est utilisé pour la valeur.
Bit 2	0	Représente le fait que l'ALM n'a pas eu lieu
	1	Représente le fait que l'ALM a eu lieu
Bit 1	0	Représente le fait qu'il n'y a pas de demande d'arrêt de la sous-unité de puissance GOF
	1	Représente le fait qu'il y a une demande d'arrêt de la sous-unité de puissance GOF
Bit 0	0	Représente le fait qu'il n'y a pas d'arrêt d'urgence du système EMG
	1	Représente le fait qu'il y a un arrêt d'urgence du système EMG

5.4.1.3.3 reserved1

Pour une extension ultérieure. Chaque octet doit être 0x00.

5.4.1.3.4 memoryAddress

Ce champ représente une adresse sur l'espace mémoire cyclique de la station de destination. Lors de la réception de la sous-charge utile, la station de réception stocke les données à l'adresse spécifiée dans cette zone de la RAM destinataire.

5.4.1.3.5 applicationData

Ce champ représente les données de commande.

5.4.2 CyclicS-PDU**5.4.2.1 cyclicMSHeader****5.4.2.1.1 frameType**

Voir 5.4.1.

5.4.2.1.2 cycleNo

Voir 5.4.1.

5.4.2.1.3 da

Voir 5.4.1.

5.4.2.1.4 réservé

Pour une extension ultérieure. Chaque octet doit être 0x00.

5.4.2.2 subPayloadHeader**5.4.2.2.1 sa**

Voir 5.4.1.

5.4.2.2.2 commInfo

Voir 5.4.1.

5.4.2.2.3 txAsynInfo

Voir 5.4.1.

5.4.2.3 subPayloadData**5.4.2.3.1 seqNo**

Voir 5.4.1.

5.4.2.3.2 diagnosisData

Ce champ représente les informations de diagnostic. L'octet supérieur représente l'état d'erreur d'application (micrologiciel), et les trois octets inférieurs représentent l'état d'erreur d'application (matériel). Les valeurs selon le Tableau 60 et le Tableau 61 sont utilisées.

Tableau 60 – L'octet supérieur de diagnosticData

Bit	Valeur	Description
Bit 7	0	Représente le fait qu'il n'y a pas d'erreur d'état d'erreur WDT.
	1	Représente le fait qu'il y a une erreur d'état d'erreur WDT.
Bit 6	0	Représente l'absence de déconnexion de vérification d'erreur N/W (notification de déconnexion de la communication entre appareils subordonnés).
	1	Représente la présence d'une déconnexion de vérification d'erreur N/W (notification de déconnexion de la communication entre appareils subordonnés).
Bit 5	-	Pour une extension ultérieure. Utiliser 0 pour la valeur.
Bit 4	0	Non invalide (les données de commande cycliques sont valides.)
	1	Invalide (les données de commande cycliques sont invalides.)

Bit	Valeur	Description
Bit 3	-	Pour une extension ultérieure. Utiliser 0 pour la valeur.
Bit 2	0	Représente l'absence d'erreur d'état d'erreur de la partie réseau.
	1	Représente la présence d'une erreur d'état d'erreur de la partie réseau.
Bit 1	0	Représente l'absence d'erreur d'état d'erreur d'application.
	1	Représente la présence d'une erreur d'état d'erreur d'application.
Bit 0	0	Représente le fait que l'opération d'application est en état d'arrêt.
	1	Représente le fait que l'opération d'application est en état de fonctionnement.

Tableau 61 – Les trois octets inférieurs de diagnosisData

Bit	Valeur	Description
Bit 23 à 8	0 à 65535	Représente le GROUPE EMG.
Bit 7 à 4	0 à 15	Représente le GROUPE GOF.
Bit 3	-	Pour une extension ultérieure. Utiliser 0 pour la valeur.
Bit 2	0	Représente le fait que l'ALM n'a pas eu lieu.
	1	Représente le fait que l'ALM a eu lieu.
Bit 1	0	Représente le fait qu'il n'y a pas de demande d'arrêt de la sous-unité de puissance GOF.
	1	Représente le fait qu'il y a une demande d'arrêt de la sous-unité de puissance GOF.
Bit 0	0	Représente le fait qu'il n'y a pas d'arrêt d'urgence du système EMG.
	1	Représente le fait qu'il y a un arrêt d'urgence du système EMG.

5.4.2.3.3 reserved1

Pour une extension ultérieure. Chaque octet doit être 0x00.

5.4.2.3.4 memoryAddress

Voir 5.4.1.

5.4.2.3.5 applicationData

Voir 5.4.1.

5.4.3 CyclicMs-PDU

5.4.3.1 cyclicMsSsHeader

5.4.3.1.1 frameType

Voir 5.4.1.

5.4.3.1.2 cycleNo

Voir 5.4.1.

5.4.3.1.3 sa

Voir 5.4.1.

5.4.3.1.4 réservé

Pour une extension ultérieure. Chaque octet doit être 0x00.

5.4.3.1.5 hec

Ce champ représente HEC. HEC est un code de détection d'erreur ciblant entre la DestAddr de la DLPDU et les données situées juste avant ce champ (toutefois, l'étiquette VLAN n'est pas incluse dans le calcul).

La définition de la DLPDU est la suivante.

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

Les types de données utilisés dans la syntaxe de DLPDU sont indiqués ci-dessous.

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

Le polynôme générateur est $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$.

Pour Cyclic Ms-PDU, chaque octet de ce champ doit être 0x00.

5.4.3.2 subPayloadHeader**5.4.3.2.1 da**

Voir 5.4.1.

5.4.3.2.2 commInfo

Voir 5.4.1.

5.4.3.2.3 txAsynInfo

Voir 5.4.1.

5.4.3.3 subPayloadData**5.4.3.3.1 seqNo**

Voir 5.4.1.

5.4.3.3.2 diagnosisData

Voir 5.4.1.

5.4.3.3.3 reserved1

Pour une extension ultérieure. Chaque octet doit être 0x00.

5.4.3.3.4 memoryAddress

Voir 5.4.1.

5.4.3.3.5 applicationData

Voir 5.4.1.

5.4.4 CyclicSs-PDU

5.4.4.1 cyclicMsSsHeader

5.4.4.1.1 frameType

Voir 5.4.1.

5.4.4.1.2 cycleNo

Voir 5.4.1.

5.4.4.1.3 da

Voir 5.4.1.

5.4.4.1.4 réservé

Pour une extension ultérieure. Chaque octet doit être 0x00.

5.4.4.1.5 hec

Voir 5.4.3.

5.4.4.2 subPayloadHeader

5.4.4.2.1 sa

Voir 5.4.1.

5.4.4.2.2 commInfo

Voir 5.4.1.

5.4.4.2.3 txAsynInfo

Voir 5.4.1.

5.4.4.3 SubPayloadCheckData

5.4.4.3.1 seqNo

Voir 5.4.1.

5.4.4.3.2 diagnosisData

Voir 5.4.2.

5.4.4.3.3 reserved1

Pour une extension ultérieure. Chaque octet doit être 0x00.

5.4.4.3.4 memoryAddress

Voir 5.4.1.

5.4.4.3.5 applicationData

Voir 5.4.1.

5.4.5 AcyclicPriority-PDU**5.4.5.1 acyclicHeader****5.4.5.1.1 frameType**

Voir 5.4.1.

5.4.5.1.2 réservé

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.5.1.3 hec

Voir 5.4.3.

5.4.5.2 acyclicPriorityData**5.4.5.2.1 srcMAC**

Ce champ représente l'adresse MAC de la station hôte.

5.4.5.2.2 mngPriority

Ce champ représente les informations de priorité lors de la détermination de la station principale de gestion. La valeur selon le Tableau 62 est utilisée.

Tableau 62 – mngPriority

Valeur	Description
0x00	Réglage interdit
0x01 à 0x7F	Représenter la priorité
0x80 à 0xFE	Réglage interdit
0xFF	Représenter la priorité

5.4.5.2.3 reserved1

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.5.2.4 mngMAC

Ce champ représente l'adresse MAC de la station principale de gestion.

5.4.5.2.5 hopCount

Ce champ représente le nombre de sauts provenant de la station principale de gestion. Une valeur comprise entre 0x0000 et 0xFFFF est utilisée.

5.4.5.2.6 KindFlag

Ce champ représente le type et la phase de cette trame. La valeur selon le Tableau 63 est utilisée.

Tableau 63 – KindFlag

Bit	Valeur	Description
Bit 7 à 3	-	Pour une extension ultérieure. 0 est utilisé pour chaque bit.
Bit 2	0	Indique la configuration unique.
	1	Indique une configuration autre que unique.
Bit 1	0	Indique l'initialisation.
	1	Indique la communication de commande.
Bit 0	0	Indique la trame de demande.
	1	Indique la trame de réponse.

5.4.5.2.7 reserved2

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.5.3 dcs

Ce champ représente le DCS. Le DCS est un code de détection d'erreur ciblant entre la DestAddr de la DLPDU et le champ avant le DCS. Pour la définition de DLPDU, voir 5.4.3.1.5.

Le polynôme générateur est $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$.

5.4.6 AcyclicDetection-PDU

5.4.6.1 acyclicDetectionHeader

5.4.6.1.1 frameType

Voir 5.4.1.

5.4.6.1.2 réservé

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.6.2 acyclicDetectionData

5.4.6.2.1 reserved1

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.6.2.2 reserved2

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.6.2.3 protocolVer

Ce champ représente la version de protocole. Une valeur comprise entre 0x00 et 0xFF est utilisée.

5.4.6.2.4 reserved3

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.6.2.5 mngMAC

Ce champ représente l'adresse MAC de la station principale de gestion.

5.4.6.2.6 previousNodeMAC

Ce champ représente l'adresse MAC de la station qui a envoyé ou relayé le plus récemment. Lors de la réception, il s'agit de l'adresse MAC d'une autre station et à la transmission (relais), elle est réécrite à une valeur égale à l'adresse MAC de la station hôte.

5.4.6.2.7 previousNodePort

Ce champ représente le numéro de port de transmission de la station qui a émis ou relayé le plus récemment. Lors de la réception, il s'agit du numéro de port d'une autre station, et à la transmission (relais), il est réécrit à une valeur égale au numéro de port de la station hôte. La valeur selon le Tableau 64 est utilisée.

Tableau 64 – previousNodePort

Valeur	Description
0x00	Ne pas utiliser
0x01 à 0x18	Représente un numéro de port compris entre Port1 et Port24
0x19 à 0xFF	Pour une extension ultérieure

5.4.6.2.8 optionFlag

Ce champ représente un fanion d'option. La valeur selon le Tableau 65 est utilisée.

Tableau 65 – optionFlag

Bit	Valeur	Description
Bit 7 à 1	-	Pour une extension ultérieure. 0 est utilisé pour chaque bit.
Bit 0	0	Détermine la nécessité de DetectionAck en fonction de l'état de la station de réception
	1	Envoie toujours DetectionAck, quel que soit l'état de la station de réception

5.4.6.2.9 hopCount

Voir 5.4.5.

5.4.6.2.10 Adresse du IP

Indique la longueur de l'adresse IPv4, du masque de sous-réseau IPv4, de l'adresse IPv6 et du préfixe du sous-réseau IPv6.

5.4.6.2.11 Send Info

Ce champ représente l'information source de cette trame. La valeur selon le Tableau 66 est utilisée.

Tableau 66 – sendInfo

Bit	Valeur	Description
Bit 7 à 1	-	Pour une extension ultérieure. 0 est utilisé pour chaque bit.
Bit 1	0	Indique la phase d'initialisation.
	1	Indique la communication de commande.
Bit 0	0	Indique la pile.
	1	Indique l'appareil périphérique et le Plug&Play.

5.4.6.2.12 Reserved4

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.7 AcyclicDetectionAck-PDU

5.4.7.1 acyclicDetectionHeader

Voir 5.4.6.

5.4.7.2 acyclicDetectionAckData

5.4.7.2.1 nodeType

Ce champ représente le type de station hôte. Chaque bit est défini dans le Tableau 67.

Tableau 67 – nodeType

Bit	Valeur	Description
Bit 7	0	Station à fonction cyclique et transitoire
	1	Station à fonction transitoire
Bit 6 à 2	-	Pour une extension ultérieure. 0 est utilisé pour chaque bit.
Bit 1 à 0	0	Station principale
	1	Station subordonnée
	2	Commutateur
	3	Utilisation interdite
	4	Station sous-principale
	5 à F	Pour une extension ultérieure

5.4.7.2.2 protocolVer

Voir 5.4.6.

5.4.7.2.3 ipAddressFourthOctet

Ce champ représente la valeur du 4e octet de l'adresse IP, détenue par la station. La valeur selon le Tableau 68 est utilisée.

Tableau 68 – 4e octet de l'adresse IP

Valeur	Description
0x00	Pas de réglage du 4e octet d'adresse IP / paramètre de 4e octet d'adresse IP non valide
0x01 à 0xFE	Valeur de réglage du 4e octet de l'adresse IP
0xFF	Pour une extension ultérieure

5.4.7.2.4 srcMAC

Ce champ représente l'adresse MAC de la station hôte.

5.4.7.2.5 reserved1

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.7.2.6 previousNodeMAC

Voir 5.4.6.

5.4.7.2.7 previousNodePort

Voir 5.4.6.

5.4.7.2.8 detectionRcvPort

Ce champ représente le numéro de port de réception de la station qui a reçu l'AcyclicDetection-PDU. La valeur selon le Tableau 69 est utilisée.

Tableau 69 – detectionRcvPort

Valeur	Description
0x00	Ne pas utiliser
0x01 à 0x18	Représente un numéro de port compris entre Port1 et Port24
0x19 à 0xFF	Pour une extension ultérieure

5.4.7.2.9 myPort

Ce champ représente le nombre de ports appartenant à la station hôte. La valeur selon le Tableau 70 est utilisée.

Tableau 70 – myPort

Valeur	Description
0x00	Ne pas utiliser
0x01 à 0x18	Représente un numéro de port compris entre Port1 et Port24
0x19 à 0xFF	Pour une extension ultérieure

5.4.7.2.10 reserved2

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.7.2.11 myPortLinkStatus

Ce champ représente l'état de liaison de tous les ports appartenant à la station hôte. Quatre bits sont utilisés pour chaque port (Port1 à Port24 à partir du port inférieur). 0x0 est utilisé pour un port qui n'existe pas. Chaque bit a la signification donnée dans le Tableau 71.

Tableau 71 – Quatre bits de myPortLinkStatus

Bit	Valeur	Description
Bit 3	0	Indique le mode duplex intégral
	1	Indique le mode semi-duplex
Bit 2 à 0	0	Indique la déconnexion de liaison
	1	Indique une liaison active à 10 Mbps
	2	Indique une liaison active à 100 Mbps
	3	Indique une liaison active à 10 Mbps
	4	Pour une extension ultérieure. 0 est utilisé pour la valeur.
	5	Pour une extension ultérieure. 0 est utilisé pour la valeur.

5.4.7.2.12 myPortFilterStatus

Ce champ représente l'état de filtrage de tous les ports appartenant à la station hôte. Quatre bits sont utilisés pour chaque port (Port1 à Port24 à partir du port inférieur). 0x0 est utilisé pour un port qui n'existe pas. Chaque bit a la signification donnée dans le Tableau 72.

Tableau 72 – Quatre bits de myPortFilterStatus

Bit	Valeur	Description
Bit 3	-	Pour une extension ultérieure. 0 est utilisé pour la valeur.
Bit 2	0	Indique un port qui n'est pas en boucle
	1	Indique un port en boucle
Bit 1 à 0	0	Indique l'absence de filtre
	1	Indique l'interdiction du relais de trame de diffusion/multidiffusion
	2	Indique l'interdiction du relais autre que la trame de réseau de champ IE CC-Link
	3	Indique l'interdiction du réglage du filtre

5.4.7.2.13 currentManager

Ce champ représente l'adresse MAC de la station principale de gestion actuelle.

5.4.7.2.14 reserved3

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.7.2.15 ipAdd

Ce champ indique une adresse IP.

5.4.7.2.16 performance

Ce champ représente les performances de la station hôte. Chaque bit a la signification donnée dans le Tableau 73.

Tableau 73 – Performance

Bit	Valeur	Description
Bit 7 à 4	-	Pour une extension future, 0 est utilisé pour la valeur
Bit 3	0	Indique qu'il n'y a pas de duplication d'adresse IP.
	1	Indique qu'il y a duplication d'adresse IP.
Bit 2	0	Indique qu'il n'y a pas d'erreur de synchronisation temporelle.
	1	Indique qu'il y a une erreur de synchronisation temporelle.
Bit 1	-	Pour une extension future, 0 est utilisé pour la valeur
Bit 0	0	Indique que le traitement est impossible avec un débit efficace équivalent à la vitesse de liaison active
	1	Indique que le traitement est possible avec un débit efficace équivalent à la vitesse de liaison active

5.4.7.2.17 reserved4

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.7.2.18 gmPriority

Ce champ représente les informations de priorité de la station hôte. Chaque bit a la signification donnée dans le Tableau 74. Pour les stations autres que la grand-mère, tous les octets doivent être 0xFF.

Tableau 74 – gmPriority

Bit	Valeur	Description
Bit 47 à 40	0 à 15	Ne pas utiliser
	16 à 255	Représente Priority1, et plus la valeur est faible, plus la priorité est élevée
Bit 39 à 8	-	Représente la qualité de l'horloge (la valeur utilisée est la même que celle de grandmasterClockQuality figurant dans la synchronisation temporelle Announce)
Bit 7 à 0	0 à 15	Ne pas utiliser
	16 à 255	Représente Priority2, et l'utilisateur définit la valeur

5.4.7.2.19 syncType

Ce champ représente le type de synchronisation temporelle de la station hôte. Chaque bit a la signification donnée dans le Tableau 75.

Tableau 75 – syncType

Bit	Valeur	Description
Bit 7 à 2	-	Pour une extension future, 0 est utilisé pour la valeur
Bit 1	0	Indique que l'IEEE Std 802.1AS n'est pas pris en charge.
	1	Indique que l'IEEE Std 802.1AS est pris en charge.
Bit 0	0	Indique que l'IEEE Std 1588 n'est pas pris en charge.
	1	Indique que l'IEEE Std 1588 est pris en charge.

5.4.7.2.20 Reserved5

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.7.2.21 (P)delay Res Time

Ce champ représente le temps entre le moment où (P)delay_Req est reçu et le moment où la réponse PDelay_Resp_Follow_Up est envoyée entre les stations voisines.

5.4.7.2.22 Delay Set Time

Ce champ représente l'instant à partir duquel le temps de retard de propagation est calculé et PortRole est mis sur MasterPort.

5.4.7.2.23 Announce Relay Time

Ce champ représente le temps entre le moment où Announce est reçu de la station voisine et PortRole paramétré, et le moment où MasterPort est relayé sur Announce.

5.4.7.2.24 Reserved6

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.7.2.25 deviceVer

Ce champ spécifie la version de l'appareil. La valeur est arbitraire.

5.4.7.2.26 vendorCode

Ce champ représente le code du fournisseur. La valeur gérée par CLPA est utilisée.

5.4.7.2.27 modelCode

Ce champ représente le code du nom du modèle. La valeur est arbitraire.

5.4.7.2.28 extensionmodelCode

Ce champ représente le code du modèle d'extension. La valeur est arbitraire. Définir 0x0000 lorsque le code de modèle étendu n'est pas utilisé.

5.4.7.2.29 DeviceType

Ce champ représente le type de modèle. La valeur gérée par CLPA est utilisée.

5.4.7.2.30 memoryAddress

Voir 5.4.2.

5.4.7.2.31 cyclicSize

Ce champ représente le nombre de points cycliques. Chaque bit a la signification donnée dans le Tableau 76.

Tableau 76 – cyclicSize

Bit	Valeur	Description
Bit 96 à 111	0x0000 à 0xFFFF	Transmission de l'appareil subordonné à l'appareil principal (la taille de l'appareil en bits est stockée en longueur d'octet.)
Bit 80 à 95	0x0000 à 0xFFFF	Transmission de l'appareil subordonné à l'appareil principal (la taille de l'appareil en mots est stockée en longueur d'octet.)
Bit 64 à 79	0x0000 à 0xFFFF	Transmission de l'appareil subordonné à l'appareil principal (la taille de l'appareil de sécurité est stockée en longueur d'octet.)
Bit 48 à 63	0x0000 à 0xFFFF	Transmission de l'appareil principal à l'appareil subordonné (la taille de l'appareil en bits est stockée en longueur d'octet.)
Bit 32 à 47	0x0000 à 0xFFFF	Transmission de l'appareil principal à l'appareil subordonné (la taille de l'appareil en mots est stockée en longueur d'octet.)
Bit 16 à 31	0x0000 à 0xFFFF	Transmission de l'appareil principal à l'appareil subordonné (la taille de l'appareil de sécurité est stockée en longueur d'octet.)
Bit 0 à 15	0x0000 à 0xFFFF	Nombre de points d'appareil de notification d'état (la taille de l'appareil en mots est stockée en longueur d'octet.)

5.4.7.2.32 Reserved7

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.7.2.33 function

Ce champ spécifie le type de fonction. Chaque bit a la signification donnée dans le Tableau 77.

Tableau 77 – fonction

Bit	Valeur	Description
Bit 7 à 5	0	Pour une extension ultérieure
Bit 4	0	Fonction de communication de sécurité non utilisée
	1	Fonction de communication de sécurité utilisée
Bit 3	0	Fonction de sauvegarde/restauration non installée
	1	Fonction de sauvegarde/restauration installée
Bit 2	0	Compteur de chien de garde non installé
	1	Compteur de chien de garde installé
Bit 1	0	Indique le rejet de l'écrasement des informations de paramètre telles que les réglages du réseau
	1	Indique l'autorisation d'écrasement d'informations de paramètre telles que les réglages du réseau (toujours à 1 pour une station subordonnée)
Bit 0	0	Indique qu'il ne s'agit pas d'une station locale
	1	Indique qu'il s'agit d'une station locale

5.4.7.2.34 optionInfo

Ce champ représente la présence ou l'absence d'informations d'option. Chaque bit a la signification donnée dans le Tableau 78.

Tableau 78 – optionInfo

Bit	Valeur	Description
Bit 7 à 2	0	Pour une extension ultérieure
Bit 1	0	Indique qu'il n'y a pas d'informations d'option
	1	Indique qu'il existe des informations d'option
Bit 0	0	Indique qu'il n'y a pas de partie d'information de l'appareil de commande
	1	Indique qu'il existe une partie d'information de l'appareil de commande

5.4.7.2.35 stationMode

Ce champ est utilisé pour identifier COMM_IF de CSP+. Une valeur comprise entre 0x0000 et 0xFFFF est utilisée.

5.4.7.2.36 vide

Ce champ représente une zone vide. 0x00 est utilisé pour chaque octet.

5.4.8 AcyclicTestData-PDU

5.4.8.1 acyclicTestDataHeader

5.4.8.1.1 frameType

0x11 est utilisé.

5.4.8.1.2 DataType

0x01 est utilisé.

5.4.8.1.3 persPriority

Ce champ représente la priorité lors de la sélection du gestionnaire de chemin de transmission. La valeur est comprise entre 0x0 et 0FFFFFFF.

Lorsque la valeur est 0x0, ce champ représente le fait que le gestionnaire de chemin de transmission n'est pas sélectionné. La signification de chaque bit est la suivante.

Bits 23 à 22: Réservé. b10 est utilisé.

Bit 21: Pour une extension ultérieure. 0 est utilisé.

Bit 20: Indique le statut. 1: Sous le contrôle d'un jeton, 0: Initialisation.

Bit 19: Indique la spécification principale. 1: Pris en charge par la fonction multiprincipale, 0: Non pris en charge.

Bits 18 à 15: Pour une extension ultérieure. 0 est utilisé pour chaque bit.

Bits 14 à 0: Le nombre de points de données cycliques est utilisé. Pour une station principale, la valeur de $(RX/16 + RY/16 + RWw + RWr)/8$ est utilisée.

Pour une station subordonnée, 0 est utilisé pour chaque bit.

5.4.8.1.4 nodeType

Ce champ représente le type de nœud. 0x30 est utilisé.

5.4.8.1.5 srcNodeNumber

Ce champ représente le numéro de nœud propre.

5.4.8.1.6 protocolVerType

La signification de chaque bit est la suivante:

Bits 7 à 4: Version du protocole. La valeur selon le Tableau 79 est utilisée.

Bits 0 à 3: Type de protocole. La valeur selon le Tableau 80 est utilisée.

Tableau 79 – Version de protocole

Valeur	Description
0x0	Prise en charge par la fonction principale unique du réseau de terrain CC-Link IE ou d'appareils de commande CC-Link IE (non prise en charge par la fonction multiprincipale)
0x1	Prise en charge par la fonction multiprincipale du réseau de terrain CC-Link IE ou d'appareils de commande CC-Link IE
0x2	Prise en charge par la fonction CC-Link IE TSN
0x3	Prise en charge par la fonction CC-Link IE TSN et la fonction principale unique du réseau de terrain CC-Link IE
0x4 à 0xF	Pour une extension ultérieure
NOTE Bit 3 à 0: La valeur selon le type de protocole est utilisée. 0x3 est utilisé.	

Tableau 80 – Type de protocole

Valeur	Description
0x0	Réseau d'appareils de commande CC-Link IE
0x1	Réseau de terrain CC-Link IE
0x2	Pour une extension ultérieure
0x3	Réseau CC-Link IE TSN
0x4 à 0xF	Pour une extension ultérieure

5.4.8.1.7 réservé

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.8.1.8 hec

Voir 5.4.3.

5.4.8.2 acyclicTestDataData**5.4.8.2.1 macAddr**

Ce champ représente l'adresse MAC de la station principale de gestion.

5.4.8.2.2 **srcPort**

Ce champ représente le numéro de port source.

5.4.8.2.3 **reserved1**

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.8.2.4 **sendInf**

Ce champ représente l'information source. La valeur selon le Tableau 81 est utilisée.

Tableau 81 – Informations relatives à la source

Bit	Valeur	Description
Bit 3 à 7	0	Pour une extension ultérieure
Bit 2	0	Le système d'adresses IP est IPv4.
	1	Le système d'adresses IP est IPv6.
Bit 1	0	La phase est la phase d'initialisation.
	1	La phase est dans la communication de commande.
Bit 0	0	La source TestData est la pile.
	1	La source TestData est l'outil technique.

5.4.8.2.5 **reserved2**

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.8.2.6 **hopCount**

Voir 5.4.5.

5.4.8.2.7 **ipv4Address**

Voir 5.4.6.

5.4.8.2.8 **ipv4Subnet**

Voir 5.4.6.

5.4.8.2.9 **reserved3**

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.8.2.10 **ipv6Address**

Voir 5.4.6.

5.4.8.2.11 **hopCount**

Voir 5.4.5.

5.4.8.2.12 **ipv6Subnet**

Voir 5.4.6.

5.4.8.2.13 dcs

Voir 5.4.5.

5.4.9 AcyclicTestDataAck-PDU**5.4.9.1 acyclicTestDataAckHeader****5.4.9.1.1 frameType**

0x12 est utilisé.

5.4.9.1.2 DataType

0x01 est utilisé.

5.4.9.1.3 persPriority

Voir 5.4.8.

5.4.9.1.4 nodeType

Voir 5.4.8.

5.4.9.1.5 srcNodeNumber

Voir 5.4.8.

5.4.9.1.6 protocolVerType

Voir 5.4.8.

5.4.9.1.7 réservé

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.9.1.8 hec

Voir 5.4.8.

5.4.9.2 acyclicTestDataAckData**5.4.9.2.1 macAddr**

Ce champ représente l'adresse MAC du nœud source de acyclicTestData-PDU.

5.4.9.2.2 srcPort

Ce champ représente le numéro de port source de acyclicTestData-PDU qui est inclus dans acyclicTestData-PDU comme srcPort.

5.4.9.2.3 rcvPort

Ce champ représente le numéro de port du nœud propre qui a reçu testDataAck-PDU.

5.4.9.2.4 reserved1

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.9.2.5 **myPorts**

Ce champ indique le nombre de ports de communication physiques détenus par le nœud propre.

5.4.9.2.6 **tokenkeepTime**

Ce champ représente la valeur maximale de la durée pendant laquelle le nœud conserve le jeton après le démarrage de la ronde du jeton.

Le temps de conservation du jeton est calculé en soustrayant le temps d'envoi de f-transientData-PDU du temps entre le moment où token-PDU est reçu par la station hôte et le moment où token-PDU est envoyé au nœud suivant.

Le nœud avertit le gestionnaire de chemin de transmission du temps de maintien du jeton par l'intermédiaire de ce champ.

La signification de chaque bit est la suivante:

Bit 15: Pour une extension ultérieure. 0 est utilisé.

Bits 14 à 0: Indique la durée définie. L'unité est de 1 µs, et la valeur est comprise entre 1 et 32 767.

NOTE Les points de début et de fin du temps de maintien du jeton sont les mêmes que ceux de token-PDU. Si le point de début est défini comme étant le point de début de réception de token-PDU, le point de fin est défini comme étant le point de début d'envoi de token-PDU. Si le point de début est défini comme étant le point de terminaison de réception de token-PDU, le point de fin est défini comme étant le point de terminaison d'envoi de token-PDU.

5.4.9.2.7 **reserved2**

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.9.2.8 **myConnectStatus**

Ce champ représente l'état du port. Chaque ensemble de quatre bits est utilisé pour montrer l'état de 24 ports au maximum.

Les bits 3 à 0 du 1er octet montrent l'état du port 1 et les bits 7 à 4 montrent l'état du port 2. Les bits 3 à 0 du 2e octet montrent l'état du port 3 et les bits 7 à 4 montrent l'état du port 4.

La même chose doit s'appliquer ci-après. Les 1er à 12e octets sont utilisés pour montrer l'état des 24 ports.

La signification de chaque bit d'un octet est la suivante:

Bits 7 à 6: Pour une extension ultérieure. 0 est utilisé pour chaque bit.

Bits 5 à 4: La valeur selon le Tableau 82 est utilisée.

Bits 3 à 2: Pour une extension ultérieure. 0 est utilisé pour chaque bit.

Bits 1 à 0: La valeur selon le Tableau 82 est utilisée.

Tableau 82 – Etat de liaison

Valeur	Description
00b	Déconnexion de liaison
01b	Liaison active
10b	Pour une extension ultérieure
11b	Pour une extension ultérieure

5.4.9.2.9 dcs

Voir 5.4.5.

5.4.10 AcyclicData-PDU**5.4.10.1 acyclicDataHeader****5.4.10.1.1 frameType**

Voir 5.4.1.

5.4.10.1.2 réservé

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.10.1.3 da

Ce champ représente le 4e octet de l'adresse IP de la station de destination ou de la station source.

5.4.10.1.4 reserved1

Pour une extension ultérieure. 0x00 est utilisé pour chaque octet.

5.4.10.2 données

Ce champ représente des données arbitraires à transmettre dans la transmission transitoire.

5.4.11 Ptp-PDU**5.4.11.1 Vue d'ensemble**

Le présent paragraphe décrit uniquement les champs qui utilisent des valeurs spécifiques dans le réseau CC-Link IE TSN. La méthode d'utilisation d'un champ qui n'est pas particulièrement mentionné doit être conforme à chaque norme.

5.4.11.2 transportpecific

0x1 est utilisé. Si PTP MesSAGE avec une valeur autre que 0x1 est reçue, le message est ignoré.

5.4.11.3 messageType

Six types de messages autres que ptpSignalling sont utilisés.

5.4.11.4 versionPtp

0x2 est utilisé.

5.4.11.5 domainNo

0x00 est utilisé.

5.4.11.6 sourcePortIdentity

Le champ ClockIdentity des 8 octets supérieurs représente l'ID d'horloge (adresse MAC). L'adresse MAC de la station d'envoi à proprement parler est utilisée.

Le champ portNumber des 2 octets inférieurs représente le numéro de port. Les valeurs 1 à N sont utilisées pour indiquer une station ayant N ports. Par exemple, la valeur 1 indique une station ayant 1 port.

5.4.11.7 SequenceID

Quand il ne s'agit pas de ptpPdelayResp ou de ptpPdelayRespFollpwUp, c'est un numéro de séquence géré pour chaque port, incrémenté de 1 pour chaque transmission. 0 est utilisé comme valeur initiale. La valeur maximale est 65 535, et lorsqu'elle dépasse la valeur maximale, elle reprend à partir de 0.

ptpPdelayResp et ptpPdelayRespFollpwUp utilisent le numéro de séquence du ptpPdelayReq reçu.

5.4.11.8 commande

0x05 est utilisé.

5.4.11.9 LogMessageInterval

Une valeur comprise entre -127 et 127 est utilisée comme valeur signée et elle est spécifiée avec $2^{\wedge}(\text{logMessageInterval})$ [s].

5.4.11.10 tlvType

0x03 est utilisé.

5.4.11.11 lengthField

28 est utilisé.

5.4.11.12 organizationSubType

1 est utilisé.

5.4.11.13 gmTimeBaseIndicator

En cas de variation de fréquence ou de phase, 0x0001 est utilisé.

5.4.11.14 lastGmPhaseChange

Il est calculé à l'aide de [temps de la grand-mère précédente] – [temps de la grand-mère actuelle]. Il est spécifié comme une valeur signée en unités de $2^{\wedge}(-16)$ nanosecondes. Par exemple, dans le cas de 2,5 nanosecondes, 0x0000_0000_0000_0000_0002 8000 est utilisé.

5.4.11.15 scaledLastGmFreqChange

La valeur avant/après la fréquence est définie.

5.4.11.16 currentUtcOffset

La valeur de décalage de l'UTC et de la TAI est définie.

5.4.11.17 gmPriority1

Cette valeur représente la valeur Priority1 de la grand-mère reconnue par la station d'envoi. Il est utilisé par le BMCA. Plus la valeur est petite, plus la priorité est élevée. Il est interdit d'utiliser 0.

5.4.11.18 gmPriority2

Cette valeur représente la valeur Priority2 de la grand-mère reconnue par la station d'envoi. Chaque utilisateur de station a la priorité définie arbitrairement. Il est interdit d'utiliser 0.

5.4.12 IpData-PDU

5.4.12.1 ipHeader

Ce champ représente l'en-tête IP. Pour plus de détails, voir l'IETF RFC 791.

5.4.12.2 udpHeader

Ce champ représente l'en-tête UDP. Pour plus de détails, voir l'IETF RFC 768.

5.4.12.3 SImp-PDU

Dans le réseau CC-Link IE TSN, la trame SLMP du type MT étendu de numéro de station est utilisée. Pour plus de détails sur chaque PDU, voir la spécification SLMP.

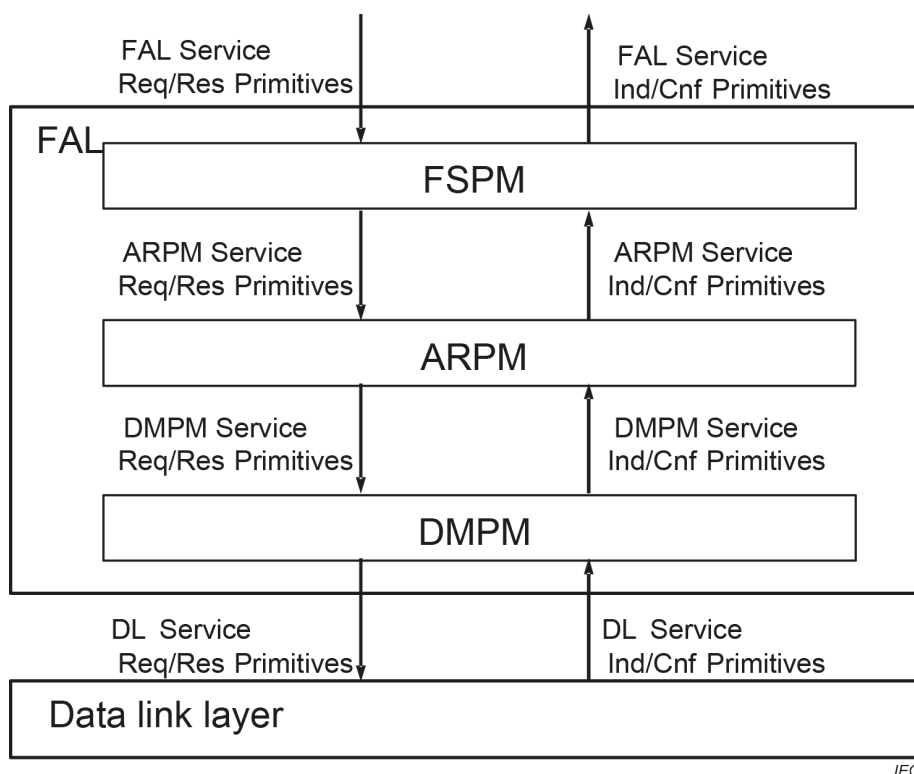
6 Structure du diagramme d'états de protocole FAL

Le diagramme d'états de protocole FAL comprend trois diagrammes d'états de protocole, tel qu'indiqué à la Figure 17. Une machine de protocole comprend, dans l'ordre depuis le côté couche liaison de données, une data link layer mapping protocol machine (DMPM, machine de protocole de mapping de couche DL), une application relationship protocol machine (ARPM, machine protocolaire de relation entre applications) et une FAL service protocol machine (FSPM, machine de protocole de service FAL).

Le rôle du FSPM consiste à recevoir des primitives de service des utilisateurs FAL et à convertir les primitives en primitives internes, à sélectionner un ARPM (diagramme d'états de protocole de mapping d'application), à recevoir les primitives internes de l'ARPM et à convertir ces dernières en primitives de service FAL et à les transmettre aux utilisateurs FAL.

Le rôle de l'ARPM consiste à convertir les primitives de service entre l'ARPM et le DMPM.

Le rôle du DMPM est le mapping dans la couche liaison de données.



Anglais	Français
FAL Service	Service FAL
Req/Res Primitives	Primitives de demande/de réponse
Ind/Cnf Primitives	Primitives Ind/Cnf
ARPM Service	Service ARPM
DMPM Service	Service DMPM
DL Service	Service DL
FAL	FAL
FSPM	FSPM
ARPM	ARPM
DMPM	DMPM
Data Link Layer	Couche liaison de données

Figure 17 – Relations entre les machines de protocole

7 Machine de protocole de service FAL (FSPM)

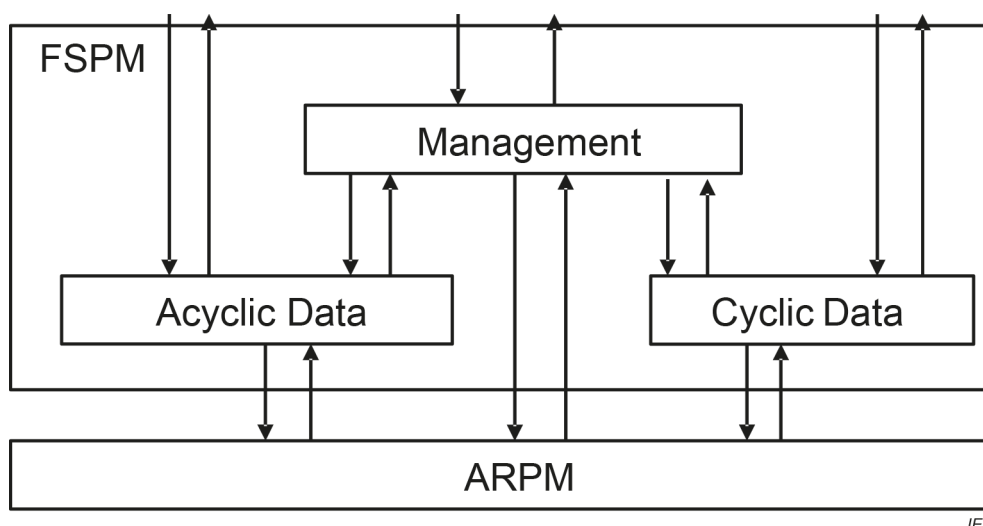
7.1 Vue d'ensemble

Le FSPM offre une interface aux utilisateurs FAL. Il procède au mapping entre les services des utilisateurs FAL et les services FAL internes.

7.2 FSPM – Type C

7.2.1 Vue d'ensemble

Le FSPM comprend trois machines de protocole: données cycliques, données acycliques et gestion. La relation entre les machines de protocole est indiquée à la Figure 18.



IEC

Anglais	Français
Management	Gestion
Acyclic Data	Données acycliques
Cyclic Data	Données cycliques

Figure 18 – Structure du type C de FSPM

Les primitives suivantes sont envoyées par l'utilisateur FAL au FSPM.

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

Les primitives suivantes sont envoyées par le FSPM à l'utilisateur FAL.

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

7.2.2 FSPM

7.2.2.1 Données cycliques

La description détaillée du diagramme d'états des données cycliques est indiquée dans le Tableau 83.

Tableau 83 – Table d'états des données cycliques

#	Etat actuel	Evénement /Condition => Action	Etat suivant
1	ACTIVE	Write Cyclic Data.req => Update BitCM Data and WordCM Data; CT Update.req	ACTIVE
2	ACTIVE	CT Update.ind => Update BitCM Data or WordCM Data.	ACTIVE
3	ACTIVE	Read Cyclic Data.req => Read Cyclic Data.cnf(BitCM Data, WordCM Data)	ACTIVE

7.2.2.2 Données acycliques

La description détaillée du diagramme d'états des données acycliques (Acyclic Data) est indiquée dans le Tableau 84.

Tableau 84 – Table d'états des données acycliques

#	Etat actuel	Evénement /Condition => Action	Etat suivant
1	ACTIVE	Send Parameter 1.req => AC Send.req	ACTIVE
2	ACTIVE	AC Send.ind(Data) / Command == Send Parameter 1 => Send Parameter 1.ind	ACTIVE
3	ACTIVE	Send Parameter 2.req => AC Send.req	ACTIVE
4	ACTIVE	AC Send.ind(Data) / Command == Send Parameter 2 => Send Parameter 2.ind	ACTIVE
5	ACTIVE	Get System Info.req => AC Send.req	ACTIVE
6	ACTIVE	AC Send.ind(Data) / Command == Get System Info => 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

#	Etat actuel	Événement /Condition => Action	Etat suivant
10	ACTIVE	AC Send.ind(Data) / Command == Get Memory Access Info => Get Memory Access Info.ind;	ACTIVE
11	ACTIVE	Get Memory Access Info.rsp => AC Send.rsp	ACTIVE
12	ACTIVE	AC Send.cnf(Data) / Command == Get Memory Access Info => Get Memory Access Info.cnf	ACTIVE
13	ACTIVE	Run.req => AC Send.req	ACTIVE
14	ACTIVE	AC Send.ind(Data) / Command == Run => Run.ind	ACTIVE
15	ACTIVE	Run.rsp => AC Send.rsp	ACTIVE
16	ACTIVE	AC Send.cnf(Data) / Command == Run => Run.cnf	ACTIVE
17	ACTIVE	Stop.req => AC Send.req	ACTIVE
18	ACTIVE	AC Send.ind(Data) / Command == Stop => Stop.ind	ACTIVE
19	ACTIVE	Stop.rsp => AC Send.rsp	ACTIVE
20	ACTIVE	AC Send.cnf(Data) / Command == Stop => Stop.cnf	ACTIVE
21	ACTIVE	Line Test.req => AC Send.req	ACTIVE
22	ACTIVE	AC Send.ind(Data) / Command == Line Test => Line Test.ind	ACTIVE
23	ACTIVE	Line Test.rsp => AC Send.rsp	ACTIVE
24	ACTIVE	AC Send.cnf(Data) / Command == Line Test => 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

#	Etat actuel	Evénement /Condition => Action	Etat suivant
28	ACTIVE	AC Send.cnf(Data) / Command == Read Memory => Read Memory.cnf	ACTIVE
29	ACTIVE	Write Memory.req => AC Send.req	ACTIVE
30	ACTIVE	AC Send.ind(Data) / Command == Write Memory => Write Memory.ind	ACTIVE
31	ACTIVE	Write Memory.rsp => AC Send.rsp	ACTIVE
32	ACTIVE	AC Send.cnf(Data) / Command == Write Memory => Write Memory.cnf	ACTIVE

7.2.2.3 Gestion

La description détaillée du diagramme d'états de gestion (Management) est indiquée dans le Tableau 85.

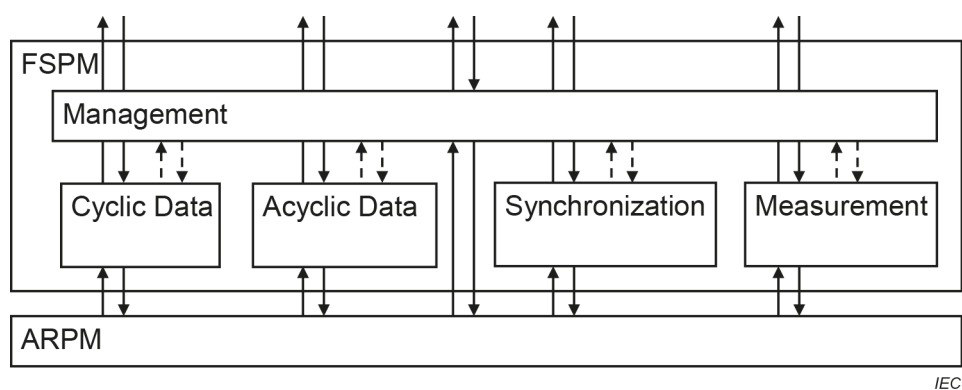
Tableau 85 – Table d'états de gestion

#	Etat actuel	Evénement /Condition => Action	Etat suivant
1	ACTIVE	Get Attribute.req => Get Attribute.ind	ACTIVE
2	ACTIVE	Get Attribute.rsp => Get Attribute.cnf	ACTIVE
3	ACTIVE	Set Attribute.req => Set Attribute.ind	ACTIVE
4	ACTIVE	Set Attribute.rsp => Set Attribute.cnf	ACTIVE

7.3 FSPM – Type F

7.3.1 Vue d'ensemble

Le FSPM comprend cinq machines de protocole: Cyclic data (données cycliques), Acyclic data (données acycliques), Management (gestion), Synchronization (synchronisation) et Measurement (mesure). La relation entre les machines de protocole est indiquée à la Figure 19. La ligne continue représente un problème de service et la ligne en pointillés représente une liaison entre les machines de protocole qui utilisent notamment des paramètres.



Anglais	Français
Management	Gestion
Cyclic Data	Données cycliques
Acyclic Data	Données acycliques
Synchronization	Synchronisation
Measurement	Mesure

Figure 19 – Structure du type F de FSPM

Les primitives suivantes sont envoyées par l'utilisateur FAL au FSPM.

RX_Ld.req
 RX_Set.req
 RX_Reset.req
 RX_Read.req
 RX_Write.req
 RY_Ld.req
 RY_Set.req
 RY_Reset.req
 RY_Read.req
 RY_Write.req
 RWr_Ld.req
 RWr_Set.req
 RWr_Reset.req
 RWr_Read.req
 RWr_Write.req
 RWw_Ld.req
 RWw_Set.req
 RWw_Reset.req
 RWw_Read.req
 RWw_Write.req
 Get Attribute.req
 Set Attribute.req
 Get Memory Access Info.req
 Get Memory Access Info.rsp
 Run.req
 Run.rsp

Stop.req
Stop.rsp
Read Memory.req
Read Memory.rsp
Write Memory.req
Write Memory.rsp
Vendor Command.req
Vendor Command.rsp
Distribute Node Info.req
Get Statistics.req
Get Statistics.rsp
Get Node Info Detail.req
Get Node Info Detail.rsp
AC Data.req
AC Data.rsp
AC Data ND.req
AC Data ND.rsp
Start Measure.req
Get Offset.req

Les primitives suivantes sont envoyées par le FSPM à l'utilisateur FAL.

RX_Ld.cnf
RX_Read.cnf
RY_Ld.cnf
RY_Read.cnf
RWr_Ld.cnf
RWr_Read.cnf
RWw_Ld.cnf
RWw_Read.cnf
Get Attribute.cnf
Set Attribute.cnf
Get Memory Access Info.ind
Get Memory Access Info.cnf
Run.ind
Run.cnf
Stop.ind
Stop.cnf
Read Memory.ind
Read Memory.cnf
Write Memory.ind
Write Memory.cnf
Vendor Command.ind
Vendor Command.cnf
Distribute Node Info.ind
Get Statistics.ind

Get Statistics.cnf

Get Node Info Detail.ind

Get Node Info Detail.cnf

AC Data.ind

AC Data.cnf

AC Data ND.ind

AC Data ND.cnf

Synchroous Triger.ind

Start Measure.cnf

Get Offset.cnf

7.3.2 FSPM

7.3.2.1 Données cycliques

La description détaillée du diagramme d'états des données cycliques est indiquée dans le Tableau 86.

Tableau 86 – Table d'états des données cycliques

#	Etat actuel	Événement/condition => action	Etat suivant
1	ACTIVE	Ld.req(Type, Address) => Ld.cnf(Data)	ACTIVE
2	ACTIVE	Set.req(Type, Address) => met à jour les valeurs RX, RY, RY, RWr ou RWw spécifiées dans le Type. CT Update.req(Type, Address, 1, 1)	ACTIVE
3	ACTIVE	Reset.req(Type, Address) => met à jour les valeurs RX, RY, RY, RWr, ou RWw spécifiées dans le Type. CT Update.req(Type, Address, 1, 1)	ACTIVE
4	ACTIVE	Read.req(Type, Address, Size) => Read.cnf(Data)	ACTIVE
5	ACTIVE	Write.req(Type, Address, Size, Data) => met à jour les valeurs RX, RY, RY, RWr, ou RWw spécifiées dans le Type. CT Update.req(Type, Address, Size, Data)	ACTIVE
6	ACTIVE	CT Update.ind(Type, Offset, Size, Data) => met à jour les valeurs RX, RY, RY, RWr, ou RWw spécifiées dans le Type.	ACTIVE

7.3.2.2 Données acycliques

La description détaillée du diagramme d'états des données acycliques (Acyclic Data) est indiquée dans le Tableau 87.

Tableau 87 – Table d'états des données acycliques

#	Etat actuel	Événement/condition => action	Etat suivant
1	ACTIVE	Get Memory Access Info.req => AC Send.req	ACTIVE
2	ACTIVE	AC Send.ind(Data) / Command == Get Memory Access Info => Get Memory Access Info.ind;	ACTIVE
3	ACTIVE	Get Memory Access Info.rsp => AC Send.rsp	ACTIVE
4	ACTIVE	AC Send.cnf(Data) / Command == Get Memory Access Info => Get Memory Access Info.cnf	ACTIVE
5	ACTIVE	Run.req => AC Send.req	ACTIVE
6	ACTIVE	AC Send.ind(Data) / Command == Run => Run.ind	ACTIVE
7	ACTIVE	Run.rsp => AC Send.rsp	ACTIVE
8	ACTIVE	AC Send.ind(Data) / Command == Run && Request type == Server Response => Run.cnf	ACTIVE
9	ACTIVE	Stop.req => AC Send.req	ACTIVE
10	ACTIVE	AC Send.ind(Data) / Command == Stop && Request type == Client Response => Stop.ind	ACTIVE
11	ACTIVE	Stop.rsp => AC Send.rsp	ACTIVE
12	ACTIVE	AC Send.cnf(Data) / Command == Stop && Request type == Server Response => Stop.cnf	ACTIVE
13	ACTIVE	Read Memory.req => AC Send.req	ACTIVE
14	ACTIVE	AC Send.ind(Data) / Command == Read Memory && Request type == Client Request => Read Memory.ind	ACTIVE

#	Etat actuel	Événement/condition => action	Etat suivant
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	Write Memory.req => AC Send.req	ACTIVE
18	ACTIVE	AC Send.ind(Data) / Command == Write Memory && Request type == Client Request => Write Memory.ind	ACTIVE
19	ACTIVE	Write Memory.rsp => AC Send.rsp	ACTIVE
20	ACTIVE	AC Send.cnf(Data) / Command == Write Memory && Request type == Server Response => Write Memory.cnf	ACTIVE
21	ACTIVE	Vendor Command.req => AC Send.req	ACTIVE
22	ACTIVE	AC Send.ind(Data) / Command == Vendor Command && Request type == Client Request => Vendor Command.ind	ACTIVE
23	ACTIVE	Vendor Command.rsp => AC Send.rsp	ACTIVE
24	ACTIVE	AC Send.cnf(Data) / Command == Vendor Command && Request type == Server Response => Vendor Command.cnf	ACTIVE
25	ACTIVE	Distribute Node Info.req => AC Send.req	ACTIVE
26	ACTIVE	AC Send.ind(Data) / Command == Distribute Node Info => Distribute Node Info.ind	ACTIVE
27	ACTIVE	Get Statistics.req => AC Send.req	ACTIVE
28	ACTIVE	AC Send.ind(Data) / Command == Get Statistics && Request type == Client Request => Get Statistics.ind	ACTIVE

#	Etat actuel	Événement/condition => action	Etat suivant
29	ACTIVE	Get Statistics.rsp => AC Send.rsp	ACTIVE
30	ACTIVE	AC Send.ind(Data) / Command == Get Statistics && Request type == Server Response => Get Statistics.cnf	ACTIVE
31	ACTIVE	Get Node Info Detail.req => AC Send.req	ACTIVE
32	ACTIVE	AC Send.ind(Data) / Command == Get Node Info Detail && Request type == Client Request => Get Node Info Detail.ind	ACTIVE
33	ACTIVE	Get Node Info Detail.rsp => AC Send.rsp	ACTIVE
34	ACTIVE	AC Send.ind(Data) / Command == Get Node Info Detail && Request type == Server Response => Get Node Info Detail.cnf	ACTIVE
35	ACTIVE	AC Data.req => AC Send.req	ACTIVE
36	ACTIVE	AC Data ND.req => AC Send ND.req	ACTIVE
37	ACTIVE	AC Send.ind(Data) / Command == AC Data && Request type == Client Request => AC Data.ind	ACTIVE
38	ACTIVE	AC Send.ind(Data) / Command == AC Send && Request type == Server Response => AC Data.cnf	ACTIVE
39	ACTIVE	AC Data.rsp => AC Send.req	ACTIVE
40	ACTIVE	AC Data ND.rsp => AC Send ND.req	ACTIVE

7.3.2.3 Gestion

La description détaillée du diagramme d'états de gestion (Management) est indiquée dans le Tableau 88.

Tableau 88 – Table d'états de gestion

#	Etat actuel	Événement/condition => action	Etat suivant
1	ACTIVE	Get Attribute.req => Get Attribute.cnf	ACTIVE
2	ACTIVE	Set Attribute.req => Set Attribute.cnf	ACTIVE

7.3.2.4 Synchronisation

La description détaillée du diagramme d'états de synchronisation (Synchronization) est indiquée dans le Tableau 89.

Tableau 89 – Table d'états de Synchronization (synchronisation)

#	Etat actuel	Événement/condition => action	Etat suivant
1	ACTIVE	Synchronous Trigger Internal.ind => Synchronous Trigger.ind	ACTIVE

7.3.2.5 Mesure

La description détaillée du diagramme d'états de mesure (Measurement) est indiquée dans le Tableau 90.

Tableau 90 – Table d'états de Measurement (mesure)

#	Etat actuel	Événement/condition => action	Etat suivant
1	ACTIVE	Start Measure.req => Start Measure Internal.req	ACTIVE
2	ACTIVE	Start Measure Internal.cnf(DATA) => Start Measure.cnf(DATA)	ACTIVE
3	ACTIVE	Get Offset.req => Get Offset Internal.req	ACTIVE
4	ACTIVE	Get Offset Internal.cnf(offset) => Get Offset.cnf(offset)	ACTIVE

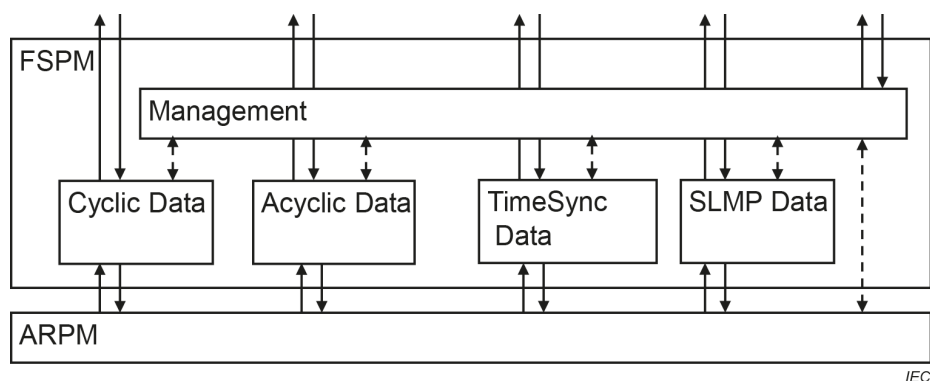
7.4 FSPM – Type T

7.4.1 Vue d'ensemble

La FSPM offre une interface aux utilisateurs FAL. Il procède au mapping entre le services de l'utilisateur FAL et le service FAL interne.

La FSPM comprend cinq machines de protocole: données cycliques, données acycliques, gestion, données de synchronisation temporelle et données SLMP.

Les relations entre les machines de protocole sont indiquées à la Figure 20. Les traits pleins indiquent l'émission d'un service, et les traits discontinus indiquent les liaisons entre les machines de protocole utilisant des variables.



IEC

Anglais	Français
Management	Gestion
Cyclic Data	Données cycliques
Acyclic Data	Données acycliques
imeSync Data	Données imeSync
SLMP Data	Données SLMP

Figure 20 – Structure du type T de FSPM

Les primitives de service fournies par la machine FSPM sont représentées dans le Tableau 91. Pour plus de détails sur chaque service, voir le Service de la couche application.

Tableau 91 – Primitives fournies par la machine FSPM

Nom de primitive	Editeur
Read.req	Utilisateur FAL
Read.cnf	FSPM
Write.req	Utilisateur FAL
Write.cnf	FSPM
Get Attribute.req	Utilisateur FAL
Get Attribute.cnf	FSPM
Set Attribute.req	Utilisateur FAL
Set Attribute. cnf	FSPM
Priority.req	Utilisateur FAL
Priority.ind	FSPM
Detection.req	Utilisateur FAL
Detection.ind	FSPM
DetectionAck.req	Utilisateur FAL
DetectionAck.ind	FSPM
Acyclic Data NRSV.req	Utilisateur FAL
Acyclic Data NRSV.ind	FSPM
Acyclic Data NRSV.rsp	Utilisateur FAL
Acyclic Data NRSV.cnf	FSPM

Nom de primitive	Editeur
Acyclic Data RSV.req	Utilisateur FAL
Acyclic Data RSV.ind	FSPM
Acyclic Data RSV.rsp	Utilisateur FAL
Acyclic Data RSV.cnf	FSPM
TimeSyncMng.req	Utilisateur FAL
TimeSyncMng.ind	FSPM
TimeSyncMng.cnf	FSPM
SLMP Data.req	Utilisateur FAL
SLMP Data.ind	FSPM
SLMP Data.rsp	Utilisateur FAL
SLMP Data.cnf	FSPM

7.4.2 Diagramme d'états FSPM

7.4.2.1 Données cycliques

L'état du diagramme d'états des données cycliques est représenté dans le Tableau 92.

Tableau 92 – Etat des données cycliques (Cyclic Data)

Nom	Description
ACTIVE	Etat de fonctionnement

La description détaillée du diagramme d'états des données cycliques est indiquée dans le Tableau 93.

Tableau 93 – Table d'états des données cycliques

#	Etat actuel	Evénement/Condition => Action	Etat suivant
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

La fonction utilisée pour les données est représentée dans le Tableau 94.

Tableau 94 – Fonction utilisée pour les données cycliques

Nom	Contenu
MemoryUpdate (Address, Size, Data)	Met à jour la mémoire cyclique spécifiée par l'argument

7.4.2.2 Données acycliques

L'état du diagramme d'états des données acycliques est représenté dans le Tableau 95.

Tableau 95 – Etat des données acycliques (Acyclic data)

Nom	Description
ACTIVE	État de fonctionnement

La description détaillée du diagramme d'états des données acycliques (Acyclic Data) est indiquée dans le Tableau 96.

Tableau 96 – Table d'états des données acycliques

#	Etat actuel	Evénement/Condition => Action	Etat suivant
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
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

#	Etat actuel	Événement/Condition => Action	Etat suivant
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 Gestion

L'état du diagramme d'états de gestion est représenté dans le Tableau 97.

Tableau 97 – Etat de la gestion (Management)

Nom	Description
ACTIVE	Etat de fonctionnement

La description détaillée du diagramme d'états de gestion (Management) est indiquée dans le Tableau 98.

Tableau 98 – Table d'états de gestion

#	Etat actuel	Événement/Condition => Action	Etat suivant
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 Données de synchronisation temporelle

L'état du diagramme d'états des données de synchronisation temporelle est représenté dans le Tableau 99.

Tableau 99 – Etat des données de synchronisation temporelle (TimeSync Data)

Nom	Description
ACTIVE	État de fonctionnement

La description détaillée du diagramme d'états des données de synchronisation temporelle est représentée dans le Tableau 100.

Tableau 100 – Table d'états des données de synchronisation temporelle

#	Etat actuel	Evénement/Condition => Action	Etat suivant
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 Données SLMP

L'état du diagramme d'états des données SLMP est représenté dans le Tableau 101.

Tableau 101 – Etat des données SLMP (SLMP Data)

Nom	Description
ACTIVE	État de fonctionnement

La description détaillée du diagramme d'états des données SLMP est représentée dans le Tableau 102.

Tableau 102 – Table d'états des données SLMP

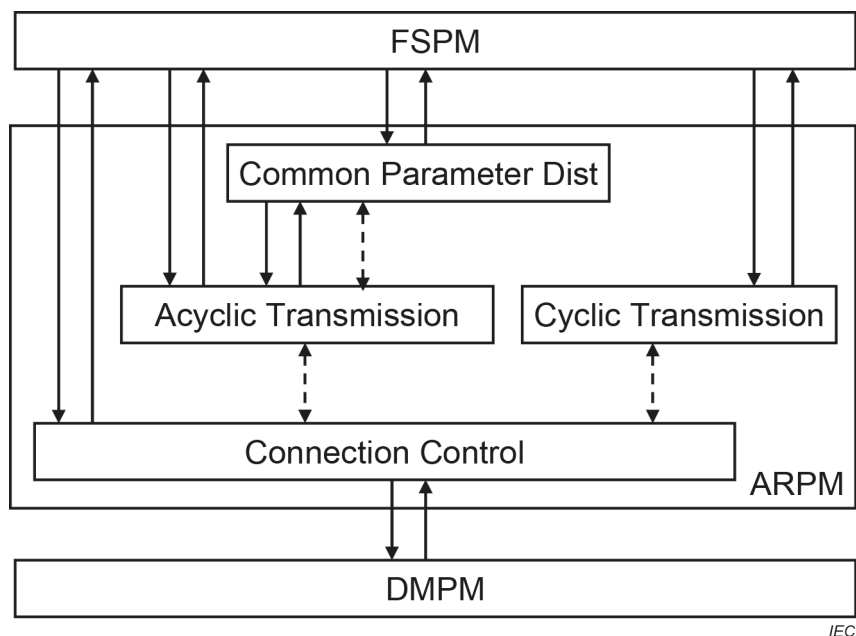
#	Etat actuel	Evénement/Condition => Action	Etat suivant
1	ACTIVE	SLMP Data.req(SlmpData) => SLMPSend.req(SlmpData)	ACTIVE
2	ACTIVE	SLMPSend.ind(SlmpData) / slmpMTHHeader.fType == 0x6800 => SLMPData.ind(SlmpData)	ACTIVE
3	ACTIVE	SLMPSend.ind(SlmpData) / slmpMTHHeader.fType == 0xE800 => SLMPData.cnf(SlmpData)	ACTIVE
4	ACTIVE	SLMPData.rsp(SlmpData) => SLMPSend.req(SlmpData)	ACTIVE

8 Machine de protocole de relation d'application (ARPM)

8.1 ARPM – Type C

8.1.1 Vue d'ensemble

L'ARPM comprend quatre sous-protocoles. La structure de l'ARPM est indiquée à la Figure 21. La ligne continue représente un problème de service et la ligne en pointillés représente une liaison entre les machines de protocole qui utilisent notamment des paramètres.



IEC

Anglais	Français
Common Parameter Dist	Common Parameter Dist
Acyclic Transmission	Acyclic transmission (Transmission acyclique)
Cyclic Transmission	Cyclic transmission (Transmission cyclique)
Connection Control	Connection Control

Figure 21 – Structure du type C de ARPM

8.1.2 Acyclic transmission (Transmission acyclique)

8.1.2.1 Définition de primitive

Le FSPM émet un service AC Send.req pour la transmission acyclique. La distribution des paramètres communs émet un service ACParmSend.req pour la transmission acyclique. La transmission acyclique émet un service ACSend.ind pour le FSPM. La transmission acyclique émet un service ACParmSend.ind pour la distribution des paramètres communs.

8.1.2.2 Diagramme d'états de la transmission acyclique (Acyclic transmission)

La description détaillée du diagramme d'états de la transmission acyclique est indiquée dans le Tableau 103.

Tableau 103 – Table d'états de la transmission acyclique

#	Etat actuel	Événement/condition => action	Etat suivant
1	IDLE	/ ACTicket == TRUE => SendCounter = MaxSend	Expéditeur
2	Expéditeur	/ SendCounter != 0 && length(RemainingData) > 0 && (OutLoopState == Through OutLoopState == Loopback) => Data = CreateTransient1-PDU(RemainingData); OutPort.req(Transient1-PDU(Data)); SendCounter = SendCounter-1	Expéditeur
3	Expéditeur	/ SendCounter != 0 && length(RemainingData) > 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateTransient1-PDU(RemainingData); InPort.req(Transient1-PDU); SendCounter = SendCounter-1	Expéditeur
4	Expéditeur	AC Send.req(Data) / SendCounter != 0 && (OutLoopState == Through OutLoopState == Loopback) => Create Transient1-PDU(Data); OutPort.req(Transient1-PDU); SendCounter = SendCounter-1	Expéditeur
5	Expéditeur	AC Send.req(Data) / SendCounter != 0 && (OutLoopState != Through && OutLoopState != Loopback) => CreateTransient1-PDU(Data); InPort.req(Transient1-PDU); SendCounter = SendCounter-1	Expéditeur
6	Expéditeur	/ SendCounter == 0 => ACTicket = FALSE	IDLE
7	IDLE	ACReceived(Transient1-PDU) => ReassembleData(Transient1-PDU)	IDLE
8	Expéditeur	ACReceived(Transient1-PDU) => ReassembleData(Transient1-PDU)	Expéditeur
9	IDLE	ACReceived(Transient2-PDU) => AC Param Send.ind(Data)	IDLE
10	Expéditeur	ACReceived(Transient2-PDU) => AC Param Send.ind(Data)	Expéditeur
11	IDLE	/ ReceivedDataAvailable == TRUE => AC Send.ind(Data); ReceivedDataAvailable = FALSE	IDLE
12	Expéditeur	/ ReceivedDataAvailable == TRUE => AC Send.ind(Data); ReceivedDataAvailable = FALSE	Expéditeur

8.1.2.3 Fonctions

Les fonctions activées lors de la transmission acyclique sont indiquées dans le Tableau 104.

Tableau 104 – Fonctions de transmission acyclique

Nom	Description
Length	Demande de taille d'argument
CreateTransient1-PDU	Génération de Transient1-PDU. Si les données de l'argument dépassent 1 464 octets, Transient1-PDU est généré à l'aide des 1 464 premiers octets. Les données restantes sont considérées comme des RemainingData
ReassembleData	Réassembler les données divisées qui ont été reçues. dataId est utilisé pour l'identification des données. Si la taille des données reçues est équivalente à wholeDataSize, il est considéré que le réassemblage est terminé et que ReceivedDataAvailable=TRUE.

8.1.3 Transmission cyclique

8.1.3.1 Définition de primitive

Le FSPM émet un service CT Update.req pour la transmission cyclique. La transmission cyclique émet un service CT Update.ind pour le FSPM.

8.1.3.2 Diagramme d'états de la transmission cyclique (Cyclic transmission)

La description détaillée du diagramme d'états de la transmission cyclique est indiquée dans le Tableau 105.

NOTE L'ordre d'envoi n'est pas spécifié pour BitCM, WordCM, OutCM1, OutCM2, InCM1 ni pour InCM2.

Tableau 105 – Table d'états de la transmission cyclique (Cyclic transmission)

#	Etat actuel	Événement/condition => action	Etat suivant
1	IDLE	/ CTicket == TRUE => seqno = 0; UpdateWaiting = FALSE; if length(BitCM) = 0 then BitCMSent = TRUE; if length(WordCM) = 0 then WordCMSent = TRUE; if length(OutCM1) = 0 then OutCM1Sent = TRUE; if length(OutCM2) = 0 then OutCM2Sent = TRUE; if length(InCM1) = 0 then InCM1Sent = TRUE; if length(InCM2) = 0 then InCM2Sent = TRUE;	Expéditeur
2	Any (Any state)	CT Update.req(Data Type, Offset Address, Size, Data) => Update(Data Type, Offset Address, Size, Data)	Any (no change)
3	Any (Any state)	CT Update.req(Data Type, Offset Address, Size, Data) => Update(Data Type, Offset Address, Size, Data)	Any (no change)
4	Expéditeur	/ UpdateWaiting == TRUE => Update(Data Type, Offset Address, Size, Data); UpdateWaiting = FALSE	Expéditeur
5	Expéditeur	/ BitCMSent != TRUE && Cyclic Control == Running && (OutLoopState == Through OutLoopState == Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER
6	Expéditeur	/ BitCMSent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataB-PDU(seqno, BitCM); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	BitCMSENDER

#	Etat actuel	Événement/condition => action	Etat suivant
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	Expéditeur
10	BitCMSENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	BitCMSENDER
11	Expéditeur	/ WordCMSent != TRUE && Cyclic Control == Running && (OutLoopState == Through OutLoopState == Loopback) => CreateCyclicDataW-PDU(seqno, WordCM); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	WordCMSENDER
12	Expéditeur	/ 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	Expéditeur
16	WordCMSENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	WordCMSENDER
17	Expéditeur	/ OutCM1Sent != TRUE&& Cyclic Control == Running && (OutLoopState == Through OutLoopState == Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER
18	Expéditeur	/ OutCM1Sent != TRUE&& Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataOut1-PDU(seqno, OutCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM1SENDER

#	Etat actuel	Événement/condition => action	Etat suivant
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	Expéditeur
22	OutCM1SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	OutCM1SENDER
23	Expéditeur	/ OutCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState == Through OutLoopState == Loopback) => CreateCyclicDataOut2-PDU(seqno, OutCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	OutCM2SENDER
24	Expéditeur	/ 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	Expéditeur
28	OutCM2SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	OutCM2SENDER
29	Expéditeur	/ InCM1Sent != TRUE && Cyclic Control == Running && (OutLoopState == Through OutLoopState == Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER
30	Expéditeur	/ InCM1Sent != TRUE && Cyclic Control == Running && (OutLoopState != Through && OutLoopState != Loopback) => CreateCyclicDataIn1-PDU(seqno, InCM1); InPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM1SENDER

#	Etat actuel	Événement/condition => action	Etat suivant
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	Expéditeur
34	InCM1SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	InCM1SENDER
35	Expéditeur	/ InCM2Sent != TRUE && Cyclic Control == Running && (OutLoopState == Through OutLoopState == Loopback) => CreateCyclicDataIn2-PDU(seqno, InCM2); OutPort.req(CyclicDataB-PDU); seqno = seqno + 1	InCM2SENDER
36	Expéditeur	/ 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	Expéditeur
40	InCM2SENDER	CT Update.req(Data Type, Offset Address, Size, Data) => UpdateWaiting = TRUE	InCM2SENDER
41	Any (Any state)	CReceived(CyclicDataB-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedBitCM = RetrieveBitCM(CyclicDataB-PDU); CT Update.ind(BitCM, Offset Address, Size, ReceivedBitCM) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (no change)
42	Any (Any state)	CReceived(CyclicDataW-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedWordCM = RetrieveWordCM(CyclicDataW- PDU); CT Update.ind(WordCM, Offset Address, Size, ReceivedWordCM) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (no change)

#	Etat actuel	Événement/condition => action	Etat suivant
43	Any (Any state)	CReceived(CyclicDataOut1-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedOutCM1 = RetrieveOutCM1(CyclicDataOut1-PDU); CT Update.ind(OutCM1, Offset Address, Size, ReceivedOutCM1) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (no change)
44	Any (Any state)	CReceived(CyclicDataOut2-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedOutCM2 = RetrieveOutCM2(CyclicDataOut2-PDU); CT Update.ind(OutCM2, Offset Address, Size, ReceivedOutCM2) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (no change)
45	Any (Any state)	CReceived(CyclicDataIn1-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedInCM1 = RetrieveInCM1(CyclicDataIn1-PDU); CT Update.ind(InCM1, Offset Address, Size, ReceivedInCM1) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (no change)
46	Any (Any state)	CReceived(CyclicDataIn2-PDU) => Si l'ordre du numéro de séquence est respecté ReceivedInCM2 = RetrieveInCM2(CyclicDataIn2-PDU); CT Update.ind(InCM2, Offset Address, Size, ReceivedInCM2) Si l'ordre du numéro de séquence n'est pas respecté, rejet	Any (no change)
47	Expéditeur	/ BitCMSent == TRUE && WordCMSent == TRUE && OutCM1Sent == TRUE && OutCM2Sent == TRUE && InCM1Sent == TRUE && InCM2Sent == TRUE => CTicket = FALSE	IDLE

8.1.3.3 Fonctions

Les fonctions activées lors de la transmission cyclique sont indiquées dans le Tableau 106.

Tableau 106 – Fonctions de transmission cyclique (Cyclic transmission)

Nom	Description
Length (Longueur)	Demande de taille donnée par argument.
Mise à jour	Mise à jour des données de BitCM, WordCM, OutCM1, OutCM2, InCM1 et InCM2 que la transmission cyclique utilise pour envoyer.
CreateCyclicDataB-PDU	Génération de CyclicDataB-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataB-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingBitCM. Si la dernière PDU est générée, la valeur de Bit 7 de seqNumber est 1.
CreateCyclicDataW-PDU	Génération de CyclicDataW-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataW-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingWordCM. Si la dernière PDU est générée, la valeur de Bit 7 de seqNumber est 1.
CreateCyclicDataOut1-PDU	Génération de CyclicDataOut1-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataOut1-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingOutCM1. Si la dernière PDU est générée, la valeur de Bit 7 de seqNumber est 1.
CreateCyclicDataOut2-PDU	Génération de CyclicDataOut2-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataOut2-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingOutCM2. Si la dernière PDU est générée, la valeur de Bit 7 de seqNumber est 1.

Nom	Description
CreateCyclicDataIn1-PDU	Génération de CyclicDataIn1-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataIn1-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingInCM1. Si la dernière PDU est générée, la valeur de Bit 7 de seqNumber est fixée à 1.
CreateCyclicDataIn2-PDU	Génération de CyclicDataIn2-PDU. Si les données de l'argument dépassent 1 468 octets, CyclicDataIn2-PDU est généré à l'aide des 1 468 premiers octets. Les données restantes sont considérées comme des RemainingInCM2. Si la dernière PDU est générée, la valeur de Bit 7 de seqNumber est 1.
RetrieveBitCM(PDU)	Récupération de l'adresse de décalage, de la taille et des données BitCM de la PDU.
RetrieveWordCM(PDU)	Récupération de l'adresse de décalage, de la taille et des données WordCM de la PDU.
RetrieveOutCM1(PDU)	Récupération de l'adresse de décalage, de la taille et des données OutCM1 de la PDU.
RetrieveOutCM2(PDU)	Récupération de l'adresse de décalage, de la taille et des données OutCM2 de la PDU.
RetrieveInCM1(PDU)	Récupération de l'adresse de décalage, de la taille et des données InCM1 de la PDU.
RetrieveInCM2(PDU)	Récupération de l'adresse de décalage, de la taille et des données InCM2 de la PDU.

8.1.4 Commande de connexion

8.1.4.1 Diagramme d'états de contrôle de connexion

Le diagramme d'états de contrôle de connexion (Connection control) est décrit ci-dessous du Tableau 107 au Tableau 120.

**Tableau 107 – Diagramme d'états de contrôle de connexion
(Connection control) – Initial**

#	Etat actuel	Événement/condition => action	Etat suivant
1	Initial	InPort.ind(Port_State) / Port_State == LinkUp => Start ConnectTimer InPortState = Checking	Connect
2	Initial	OutPort.ind(Port_State) / Port_State == LinkUp => Start ConnectTimer OutPortState = Checking	Connect
3	Initial	InPort.ind(Port_State) / Port_State == LinkDown =>	Initial
4	Initial	OutPort.ind(Port_State) / Port_State == LinkDown =>	Initial
5	Initial	/ NTNTest == TRUE =>	NTNTestMaster

Tableau 108 – Diagramme d'états de contrôle de connexion (Connection control) – Connect

#	Etat actuel	Événement/condition => action	Etat suivant
1	Connect	InPort.ind(Port_State) / Port_State == LinkUp && InPortState == LinkDown && OutPortState == Checking => InPortState = Checking	Connect
2	Connect	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == Checking && OutPortState == LinkDown => OutPortState = Checking	Connect
3	Connect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == NG && OutPortState == Checking => InPortState = LinkDown	Connect
4	Connect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == Checking && OutPortState != OK => InPortState = LinkDown; Stop ConnectTimer.	Initial
5	Connect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == Checking && OutPortState == NG => OutPortState = LinkDown	Connect
6	Connect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == Checking => OutPortState = LinkDown; Stop ConnectTimer	Initial
7	Connect	ConnectTimer times out. / InPortState == Checking && (OutPortState != OK && OutPortState != Checking) => InPort.req(Connect-PDU(PortChoice=In))	Connect
8	Connect	ConnectTimer times out. / (InPortState != OK && InPortState != Checking) && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	Connect
9	Connect	ConnectTimer times out. / InPortState == Checking && OutPortState == Checking => InPort.req(Connect-PDU(PortChoice=In)) OutPort.req(Connect-PDU(PortChoice=Out))	Connect
10	Connect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	Connect
11	Connect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice != Out => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	Connect
12	Connect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Connect

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

Tableau 109 – Diagramme d'états de contrôle de connexion (Connection control) – Scan

#	Etat actuel	Événement/condition => action	Etat suivant
1	Scan	InPort.ind(Port_State) / Port_State == LinkUp && InPortState == LinkDown && OutPortState == OK && StopConnectFlag != ON => Start ConnectTimer; InPortState = Checking	Scan
2	Scan	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState == LinkDown && StopConnectFlag != ON => Start ConnectTimer; OutPortState = Checking	Scan
3	Scan	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => 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
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

#	Etat actuel	Événement/condition => action	Etat suivant
13	Scan	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	Scan
14	Scan	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Scan
15	Scan	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	Scan
16	Scan	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
17	Scan	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	Scan
18	Scan	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK && 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

#	Etat actuel	Événement/condition => action	Etat suivant
24	Scan	InPort.ind(source_address, Scan-PDU(scanState)) / source_address == my address && InPortState == OK => Start DetectScanTimer.	ScanWait
25	Scan	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
26	Scan	OutPort.ind(source_address, Scan-PDU(scanState)) / source_address == my address && InPortState != OK && OutPortState == OK => Start DetectScanTimer.	ScanWait
27	Scan	OutPort.ind(source_address, Scan-PDU) / source_address == my address && InPortState == OK && OutPortState == OK && Latest(Scan-PDU) == TRUE => InPort.req(Scan-PDU)	Scan
28	Scan	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
29	Scan	InPort.ind(PDU) / PDU != Connect-PDU && PDU != ConnectAck-PDU && PDU != Scan-PDU =>	Scan
30	Scan	OutPort.ind(PDU) / PDU != Connect-PDU && PDU != ConnectAck-PDU && PDU != Scan-PDU =>	Scan
31	Scan	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestNG));	NTNTestSlave
32	Scan	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == NTNTest => InPortState = LinkDown; OutPort.req(ConnectAck-PDU(PortCheckResult = NTNTestOK));	NTNTestSlave

**Tableau 110 – Diagramme d'états de contrôle de connexion
(Connection control) – ScanWait**

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

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

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

**Tableau 111 – Diagramme d'états de contrôle de connexion
(Connection control) – Collect**

#	Etat actuel	Événement/condition => action	Etat suivant
1	Collect	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	Collect
2	Collect	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	Collect
3	Collect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	Collect	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan
5	Collect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	Collect	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	Collect	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	Collect
8	Collect	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	Collect
9	Collect	CollectTimer times out. / InPortState == OK && OutPortState == OK => OutPort.req(Collect-PDU(InLoopState = Through, OutLoopState = Through))	Collect
10	Collect	CollectTimer times out. / InPortState == OK && OutPortState != OK => InPort.req(Collect-PDU(InLoopState = Loopback, OutLoopState = OutPortState))	Collect
11	Collect	CollectTimer times out. / InPortState != OK && OutPortState == OK => OutPort.req(Collect-PDU(InLoopState = InPortState, OutLoopState = Loopback))	Collect
12	Collect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	Collect

#	Etat actuel	Événement/condition => action	Etat suivant
13	Collect	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	Collect
14	Collect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	Collect
15	Collect	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	Collect
16	Collect	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
17	Collect	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	Collect
18	Collect	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
19	Collect	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	Collect
20	Collect	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
21	Collect	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	Collect	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
23	Collect	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
24	Collect	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan

#	Etat actuel	Événement/condition => action	Etat suivant
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

**Tableau 112 – Diagramme d'états de contrôle de connexion
(Connection control) – CollectWait**

#	Etat actuel	Événement/condition => action	Etat suivant
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

#	Etat actuel	Événement/condition => action	Etat suivant
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
24	CollectWait	InPort.ind(source_address, Collect-PDU) / source_address == my address && InPortState == OK => Restart DetectCollectTimer.	CollectWait

#	Etat actuel	Événement/condition => action	Etat suivant
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

**Tableau 113 – Diagramme d'états de contrôle de connexion
(Connection control) – Select**

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

#	Etat actuel	Événement/condition => action	Etat suivant
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

#	Etat actuel	Événement/condition => action	Etat suivant
25	Select	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	Select
26	Select	InPort.ind(source_address, Select-PDU) / source_address == my address && InPortState == OK => Start LaunchTimer.	TokenStartWait
27	Select	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	LaunchWait
28	Select	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
29	Select	InPort.ind(source_address, Select-PDU) / source_address > my address =>	Select
30	Select	OutPort.ind(source_address, Select-PDU) / source_address == my address && InPortState != OK && OutPortState == OK => Start LaunchTimer.	TokenStartWait
31	Select	OutPort.ind(source_address, Select-PDU) / source_address < my address && InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
32	Select	OutPort.ind(source_address, Select-PDU) / source_address > my address && InPortState != OK && OutPortState == OK =>	Select
33	Select	OutPort.ind(Select-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Select-PDU)	Select
34	Select	InPort.ind(PDU) / (PDU == Token-PDU PDU == MyStatus-PDU PDU == Transient1-PDU PDU == Dummy-PDU PDU == Transient2-PDU PDU == CyclicDataW-PDU PDU == CyclicDataB-PDU PDU == CyclicDataOut1-PDU PDU == CyclicDataOut2-PDU PDU == CyclicDataIn1-PDU PDU == CyclicDataIn2-PDU) && InPortState == OK =>	Select

#	Etat actuel	Événement/condition => action	Etat suivant
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

**Tableau 114 – Diagramme d'états de contrôle de connexion
(Connection control) – TokenStartWait**

#	Etat actuel	Événement/condition => action	Etat suivant
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

#	Etat actuel	Événement/condition => action	Etat suivant
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

#	Etat actuel	Événement/condition => action	Etat suivant
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

**Tableau 115 – Diagramme d'états de contrôle de connexion
(Connection control) – LaunchWait**

#	Etat actuel	Événement/condition => action	Etat suivant
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

#	Etat actuel	Événement/condition => action	Etat suivant
12	LaunchWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	LaunchWait
13	LaunchWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
14	LaunchWait	InPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	LaunchWait
15	LaunchWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
16	LaunchWait	OutPort.ind(ConnectAck-PDU(PortCheckResult) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	LaunchWait
17	LaunchWait	OutPort.ind(Select-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Select-PDU)	LaunchWait
18	LaunchWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
19	LaunchWait	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
20	LaunchWait	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
21	LaunchWait	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
22	LaunchWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == Checking => OutPortState = OK; InPort.req(Scan-PDU)	Scan
23	LaunchWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	LaunchWait
24	LaunchWait	InPort.ind(source_address, Select-PDU) / source_address == my address && =>	LaunchWait
25	LaunchWait	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	LaunchWait

#	Etat actuel	Événement/condition => action	Etat suivant
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

**Tableau 116 – Diagramme d'états de contrôle de connexion
(Connection control) – TokenReleaseWait**

#	Etat actuel	Événement/condition => action	Etat suivant
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

#	Etat actuel	Événement/condition => action	Etat suivant
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

#	Etat actuel	Événement/condition => action	Etat suivant
25	TokenReleaseWait	OutPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Scan-PDU)	TokenReleaseWait
26	TokenReleaseWait	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState != OK => InPort.req(Select-PDU)	LaunchWait
27	TokenReleaseWait	InPort.ind(source_address, Select-PDU) / source_address < my address && InPortState == OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
28	TokenReleaseWait	InPort.ind(source_address, Select-PDU) / source_address > my address =>	Select
29	TokenReleaseWait	OutPort.ind(source_address, Select-PDU) / source_address < my address && InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
30	TokenReleaseWait	OutPort.ind(source_address, Select-PDU) / source_address > my address && InPortState != OK && OutPortState == OK =>	Select
31	TokenReleaseWait	OutPort.ind(Select-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Select-PDU)	TokenReleaseWait
32	TokenReleaseWait	InPort.ind(Launch-PDU) =>	TokenReleaseWait
33	TokenReleaseWait	OutPort.ind(Launch-PDU) =>	TokenReleaseWait
34	TokenReleaseWait	InPort.ind(PDU) / (PDU == MyStatus-PDU PDU == Transient1-PDU PDU == Dummy-PDU PDU == Transient2-PDU PDU == CyclicDataW-PDU PDU == CyclicDataB-PDU PDU == CyclicDataOut1-PDU PDU == CyclicDataOut2-PDU PDU == CyclicDataIn1-PDU PDU == CyclicDataIn2-PDU) && InPortState == OK =>	TokenReleaseWait
35	TokenReleaseWait	OutPort.ind(PDU) / (PDU == MyStatus-PDU PDU == Transient1-PDU PDU == Dummy-PDU PDU == Transient2-PDU PDU == CyclicDataW-PDU PDU == CyclicDataB-PDU PDU == CyclicDataOut1-PDU PDU == CyclicDataOut2-PDU PDU == CyclicDataIn1-PDU PDU == CyclicDataIn2-PDU) && OutPortState == OK =>	TokenReleaseWait
36	TokenReleaseWait	Control Cyclic.req(Control_Type) / Control_Type == Restart => CyclicControl = Running; Control Cyclic.cnf(State = Running)	TokenReleaseWait

#	Etat actuel	Événement/condition => action	Etat suivant
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

**Tableau 117 – Diagramme d'états de contrôle de connexion
(Connection control) – TokenReleased**

#	Etat actuel	Événement/condition => action	Etat suivant
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
9	TokenReleased	NetworkWatchTime times out.=>	Select

#	Etat actuel	Événement/condition => action	Etat suivant
10	TokenReleased	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	TokenReleased
11	TokenReleased	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	TokenReleased
12	TokenReleased	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	TokenReleased
13	TokenReleased	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	TokenReleased
14	TokenReleased	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
15	TokenReleased	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	TokenReleased
16	TokenReleased	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan
17	TokenReleased	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == OK && OutPortState == Checking => OutPortState = NG	TokenReleased
18	TokenReleased	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Scan-PDU)	Scan
19	TokenReleased	InPort.ind(Scan-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan
20	TokenReleased	InPort.ind(Scan-PDU) / InPortState == Checking && OutPortState == OK => InPortState = OK; OutPort.req(Scan-PDU)	Scan
21	TokenReleased	OutPort.ind(Scan-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Scan-PDU)	Scan

#	Etat actuel	Événement/condition => action	Etat suivant
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
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

#	Etat actuel	Événement/condition => action	Etat suivant
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
41	TokenReleased	InPort.ind(source_address, PDU) / source_address != my address && (PDU == CyclicDataW-PDU PDU == CyclicDataB-PDU PDU == CyclicDataOut1-PDU PDU == CyclicDataOut2-PDU PDU == CyclicDataIn1-PDU PDU == CyclicDataIn2-PDU) && InPortState == OK && OutPortState == OK => CReceived(PDU) OutPort.req(PDU): Restart NetworkWatchTimer.	TokenWait

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

#	Etat actuel	Événement/condition => action	Etat suivant
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

**Tableau 118 – Diagramme d'états de contrôle de connexion
(Connection control) – TokenWait**

#	Etat actuel	Événement/condition => action	Etat suivant
1	TokenWait	InPort.ind(Port_State) / Port_State == LinkUp && InPortState != OK && OutPortState == OK => Start ConnectTimer; InPortState = Checking	TokenWait
2	TokenWait	OutPort.ind(Port_State) / Port_State == LinkUp && InPortState == OK && OutPortState != OK => Start ConnectTimer; OutPortState = Checking	TokenWait
3	TokenWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != OK => InPortState = LinkDown	Initial
4	TokenWait	InPort.ind(Port_State) / Port_State == LinkDown && InPortState != LinkDown && OutPortState == OK => InPortState = LinkDown	Scan

#	Etat actuel	Événement/condition => action	Etat suivant
5	TokenWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState != OK && OutPortState == OK => OutPortState = LinkDown	Initial
6	TokenWait	OutPort.ind(Port_State) / Port_State == LinkDown && InPortState == OK && OutPortState != LinkDown => OutPortState = LinkDown	Scan
7	TokenWait	ConnectTimer times out. / InPortState == Checking && OutPortState == OK => InPort.req(Connect-PDU(PortChoice=In));	TokenWait
8	TokenWait	ConnectTimer times out. / InPortState == OK && OutPortState == Checking => OutPort.req(Connect-PDU(PortChoice=Out))	TokenWait
9	TokenWait	NetworkWatchTime times out. =>	Select
10	TokenWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=OK)); InPortState = OK	TokenWait
11	TokenWait	InPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK OutPortState == OK) && InPortState != LinkDown => InPort.req(ConnectAck-PDU(PortCheckResult=NG)); InPortState = NG	TokenWait
12	TokenWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == In && (InPortState == OK OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=OK)); OutPortState = OK	TokenWait
13	TokenWait	OutPort.ind(Connect-PDU(PortChoice)) / PortChoice == Out && (InPortState == OK OutPortState == OK) && OutPortState != LinkDown => OutPort.req(ConnectAck-PDU(PortCheckResult=NG)); OutPortState = NG	TokenWait
14	TokenWait	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK && InPortState == Checking && OutPortState == OK => InPortState = OK	Scan
15	TokenWait	InPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == NG && InPortState == Checking && OutPortState == OK => InPortState = NG	TokenWait
16	TokenWait	OutPort.ind(ConnectAck-PDU(PortCheckResult)) / PortCheckResult == OK && InPortState == OK && OutPortState == Checking => OutPortState = OK	Scan

#	Etat actuel	Événement/condition => action	Etat suivant
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
26	TokenWait	InPort.ind(source_address, Select-PDU) / source_address > my address && InPortState == OK =>	Select
27	TokenWait	OutPort.ind(source_address, Select-PDU) / source_address < my address && InPortState != OK && OutPortState == OK => OutPort.req(Select-PDU)	LaunchWait
28	TokenWait	OutPort.ind(source_address, Select-PDU) / source_address > my address && InPortState != OK && OutPortState == OK =>	Select
29	TokenWait	OutPort.ind(source_address, Select-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Select-PDU)	TokenWait
30	TokenWait	InPort.ind(Launch-PDU) / InPortState == OK && OutPortState != OK => InPort.req(Launch -PDU)	TokenWait

#	Etat actuel	Événement/condition => action	Etat suivant
31	TokenWait	InPort.ind(Launch-PDU) / InPortState == OK && OutPortState == OK => OutPort.req(Launch -PDU)	TokenWait
32	TokenWait	OutPort.ind(Launch-PDU) / InPortState != OK && OutPortState == OK => OutPort.req(Launch -PDU)	TokenWait
33	TokenWait	OutPort.ind(Launch-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Launch -PDU)	TokenWait
34	TokenWait	InPort.ind(Token-PDU) / InPortState == OK =>	TokenReleaseWait
35	TokenWait	OutPort.ind(Token-PDU) / InPortState != OK && OutPortState == OK =>	TokenReleaseWait
36	TokenWait	OutPort.ind(Token-PDU) / InPortState == OK && OutPortState == OK => InPort.req(Token-PDU)	TokenWait
37	TokenWait	InPort.ind(source_address, PDU) / (PDU == MyStatus-PDU PDU == Dummy-PDU) && InPortState == OK && OutPortState != OK => InPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
38	TokenWait	InPort.ind(source_address, PDU) / (PDU ==Transient1-PDU PDU ==Transient2-PDU) && InPortState == OK && OutPortState != OK => ACReceived(PDU); InPort.req(PDU); Restart NetworkWatchTimer.	TokenWait
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

#	Etat actuel	Événement/condition => action	Etat suivant
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
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

#	Etat actuel	Événement/condition => action	Etat suivant
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

**Tableau 119 – Diagramme d'états de contrôle de connexion
(Connection control) – NTNTestMaster**

#	Etat actuel	Événement/condition => action	Etat suivant
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

**Tableau 120 – Diagramme d'états de contrôle de connexion
(Connection control) – NTNTestSlave**

#	Etat actuel	Événement/condition => action	Etat suivant
1	NTNTestSlave	OutPort.ind(Connect-PDU) => OutPortReq(Connect-PDU)	NTNTestSlave
2	NTNTestSlave	OutPort.ind(Port_State) / Port_State == LinkDown =>	Initial

8.1.4.2 Fonctions

Les fonctions utilisées pour le contrôle de la connexion sont indiquées dans le Tableau 121.

Tableau 121 – Liste des fonctions de Connection control (contrôle de connexion)

Nom	Description
Latest(Scan-PDU)	Décide si les données de Scan-PDU sont les dernières. Si tel est le cas, le retour est TRUE; dans le cas contraire, le retour est FALSE.
CPDReceived(Collect-PDU)	Retour de Collect-PDU au diagramme d'états de distribution de paramètres communs.
CReceived(PDU)	Retour de PDU au diagramme d'états de transmission cyclique.
ACReceived(PDU)	Retour de PDU au diagramme d'états de transmission acyclique.

8.1.5 Common parameter dist

8.1.5.1 Diagramme d'états de distribution de paramètres communs

La description détaillée du diagramme d'états de distribution de paramètres communs (Common parameter dist) est indiquée dans le Tableau 122.

**Tableau 122 – Table d'états de Common parameter dist
(distribution de paramètres communs)**

#	Etat actuel	Événement/condition => action	Etat suivant
1	SetUp	CPD Set.req(NodeId, Param) => CPUpdate(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

#	Etat actuel	Événement/condition => action	Etat suivant
5	Initial	CPD Set.req(NodeID, Param) => CPUpdate(NodeId, Param); CPID = CreateCPID(NodeId); CPIDUpdate(CPID); CPD Set.ind	Initial
6	CPNodeTypeSelect	/ MasterNode == Master && ReceivingNodes > 0 => CreateDistResultList()	MasterInit
7	CPNodeTypeSelect	/ MasterNode == Master && ReceivingNodes == 0	MasterEnd
8	CPNodeTypeSelect	/ MasterNode == Slave && HasValidCP() == TRUE	SlaveEnd
9	CPNodeTypeSelect	/ MasterNode == Slave && HasValidCP() != TRUE	SlaveInit
10	MasterInit	CollectEnd == TRUE => MasterNode = CPDMaster(); ReceivingNodes = DestNodeNum()	CPNodeTypeSelect
11	MasterInit	TokenPassingStart==TRUE => CPW.req	CPWSend
12	MasterInit	CPD Set.req(NodeID, Param) => CPUpdate(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) => CPUpdate(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

#	Etat actuel	Événement/condition => action	Etat suivant
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) => CPUpdate(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(RecvNodeList) == TRUE && CPComp(CP) == 0 => CPIDUpdate(CPID)	SlaveEnd
26	SlaveInit	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) != 0 => CPIDUpdate(CPID); CPCheck(CP); CPCheckResult = CPChecking	ReceivedCPChecking
27	SlaveInit	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) == 0 => CPWCR.req(CheckResult = CPNotReceived)	SlaveInit
28	SlaveInit	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) != 0 => CPWCR.req(CheckResult = CPCheckResult)	SlaveEnd
29	SlaveInit	CPD Set.req(NodeID, Param) => CPUpdate(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(RecvNodeList) == TRUE && CPComp(CP) == 0 => CPIDUpdate(CPID)	SlaveEnd
32	ReceivedCPChecking	CPW.ind(RecvNodeList, CPID, CP) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) != 0 => CPIDUpdate(CPID); CPChanged = TRUE	ReceivedCPChecking

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

#	Etat actuel	Événement/condition => action	Etat suivant
46	SlaveEnd	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) == 0 => CPWCR.req(CheckResult = CPNotReceived)	SlaveInit
47	SlaveEnd	CPWC.ind(RecvNodeList, CPID) / IncludeThisNode(RecvNodeList) == TRUE && CPComp(CP) != 0 => CPWCR.req(CheckResult = CPCheckResult)	SlaveEnd

8.1.5.2 Fonctions

Les fonctions activées pour la distribution des paramètres de connexion sont indiquées dans le Tableau 123.

Tableau 123 – Liste des fonctions de Connection control (contrôle de connexion)

Nom	Description
CPUdate(NodeId, Param)	Mise à jour des paramètres communs.
CreateCPID(NodeId)	Génération de l'ID des paramètres communs avec la valeur NodeId et l'heure.
CPIDUpdate(CPID)	Mise à jour de l'ID des paramètres communs (CPID).
CPDMaster()	Détermine si le nœud procède à une remise de paramètres. Si tel est le cas, le retour est TRUE; dans le cas contraire, le retour est FALSE. Un nœud procède à la remise des paramètres en fonction des critères suivants: 1) S'il s'agit d'un nœud de gestion, le nœud procède à la remise des paramètres. 2) S'il n'existe aucun nœud de gestion, si la transmission cyclique est exécutée et si le numéro de nœud est le plus petit parmi les nœuds concernés par la transmission cyclique, le nœud exécute la remise des paramètres. 3) En l'absence de nœud de gestion, si aucun nœud ne procède à la transmission cyclique, si le nœud conserve les paramètres communs et si le numéro du nœud est le plus petit des nœuds qui détiennent les paramètres communs, le nœud exécute la remise des paramètres.
IsValidCP()	Comparaison de l'ID des paramètres communs remis au nœud par rapport à l'ID des paramètres communs détenus. Si les ID sont identiques, le retour est TRUE; dans le cas contraire, le retour est FALSE.
DestNodeNum()	Retour du numéro des nœuds auxquels il est nécessaire d'envoyer des paramètres. La nécessité d'envoyer des paramètres est déterminée comme suit. L'ID des paramètres communs d'un nœud est 0. L'ID des paramètres communs remis d'un nœud est différent de l'ID des paramètres communs qui doit être envoyé.
CreateDestNodeList()	Génération de la liste des nœuds auxquels il est nécessaire d'envoyer des paramètres. La nécessité d'envoyer des paramètres est déterminée de la même façon que pour DestNodeNum().
CreateResultCheckList()	Génération de la liste des nœuds au niveau desquels il est nécessaire de vérifier le résultat de l'envoi des paramètres. La nécessité d'envoyer des paramètres est déterminée de la même façon que pour DestNodeNum().
AddDestNode(NodeId)	Ajout des nœuds au niveau desquels il est nécessaire d'envoyer des paramètres à la liste des nœuds générée dans CreateDestNodeList().
RemoveDestNode(NodeId)	Supprime un nœud spécifique de la liste de nœuds générée dans CreateDestNodeList().
IncludeThisNode(RecvNodeList)	Détermine si le nœud est inclus dans RecvNodeList. Si tel est le cas, le retour est TRUE; dans le cas contraire, le retour est FALSE.
CPComp(CP)	Comparaison du paramètre commun indiqué comme CP par rapport au paramètre commun conservé. S'ils sont identiques, le retour est 0; dans le cas contraire, le retour est un numéro autre que 0.
CPCheck(CP)	Vérification de la bonne réception du paramètre commun indiqué comme CP. Après la vérification, le résultat (CPCheckOK ou CPCheckNG) est saisi dans CPCheckResult et l'événement CPCheckFinished a lieu.

8.1.5.3 Mapping du service interne

Le service interne que la distribution de paramètres communs émet ou reçoit est mappé dans le service pour la transmission acyclique. Le service interne et le mapping des services pour la transmission acyclique sont indiqués dans le Tableau 124.

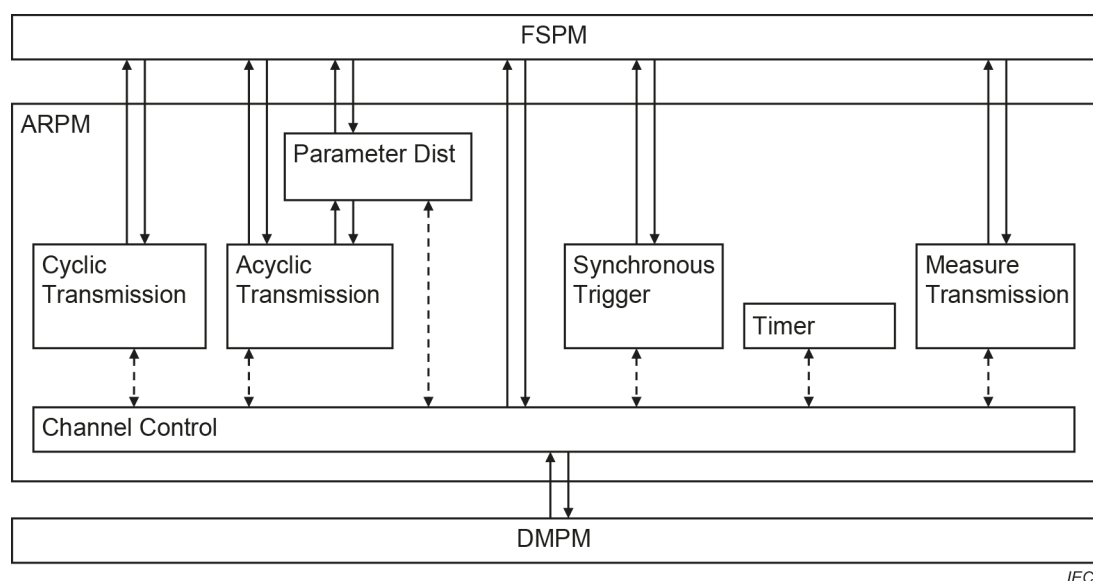
Tableau 124 – Mapping du service interne (internal service) et du service de transmission acyclique (acyclic transmission service)

Service interne	Service de transmission acyclique	Paramètre
CPW.req	AC Param Send.req	Request Type (Type de demande): Push Request (Demande push) Data Type (Type de données): Parameter Distribution (Distribution de paramètres) Données: Voir Tableau 17.
CPW.ind	AC Param Send.ind	Request Type (Type de demande): Push Request (Demande push) Data Type (Type de données): Parameter Distribution (Distribution de paramètres) Données: Voir Tableau 17.
CPWC.req	AC Param Send.req	Request Type (Type de demande): Push Request (Demande push) Data Type (Type de données): Parameter Distribution (Distribution de paramètres) Data (Données): voir Tableau 18.
CPWC.ind	AC Param Send.ind	Request Type (Type de demande): Push Request (Demande push) Data Type (Type de données): Parameter Distribution (Distribution de paramètres) Données: Voir Tableau 18.
CPWCR.req	AC Param Send.req	Request Type (Type de demande): Push Request (Demande push) Data Type (Type de données): Parameter Distribution (Distribution de paramètres) Data (Données): voir Tableau 19.
CPWCR.ind	AC Param Send.ind	Request Type (Type de demande): Push Request (Demande push) Data Type (Type de données): Parameter Distribution (Distribution de paramètres) Data (Données): voir Tableau 19.

8.2 ARPM – Type F

8.2.1 Vue d'ensemble

La structure de l'ARPM est indiquée à la Figure 21. La ligne continue représente un problème de service et la ligne en pointillés représente une liaison entre les machines de protocole qui utilisent notamment des paramètres.



Anglais	Français
Parameter Dist	Parameter Dist
Cyclic Transmission	Cyclic transmission (Transmission cyclique)
Acyclic Transmission	Acyclic transmission (Transmission acyclique)
Synchronous Trigger	Synchronous trigger (Déclenchement synchrone)
Timer	Temporisateur
Measure Transmission	Transmission de mesure
Channel Control	Channel control (Contrôle de canal)

Figure 22 – Structure du type F de ARPM

8.2.2 Acyclic transmission (Transmission acyclique)

8.2.2.1 Définition de primitive

Le FSPM émet un service AC Send.req ou AC Send ND.req pour la transmission acyclique. La distribution des paramètres émet un service AC Send.req ou AC Send ND.req pour la transmission acyclique. La transmission acyclique émet un service AC Send.ind pour le FSPM. La transmission acyclique émet un service AC Param Send.ind pour la distribution des paramètres.

8.2.2.2 Diagramme d'états de la transmission acyclique (Acyclic transmission)

Les états du diagramme d'états de la transmission acyclique (Acyclic transmission) sont indiqués dans le Tableau 125 et les détails sont spécifiés dans le Tableau 126.

Tableau 125 – États de la transmission acyclique (Acyclic transmission)

Nom	Description
IDLE	Attente du début de la transmission acyclique
ACSENDER	Transmission acyclique en cours
ACSENDER_ND	Transmission acyclique d'après la méthode de limitation de balayage de nœud en cours.

Tableau 126 – Table d'états de la transmission acyclique

#	Etat actuel	Événement/condition => action	Etat suivant
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
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

#	Etat actuel	Événement/condition => action	Etat suivant
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 Fonctions

Les fonctions activées lors de la transmission acyclique sont indiquées dans le Tableau 127.

Tableau 127 – Fonctions de transmission acyclique

Nom	Description
CreateTransient-PDU (netno, nodeno, actype, data)	Génère transient1-PDU si actype==Transient1, transientAck-PDU si actype==TransientAck, transient2-PDU si actype==Transient2, parameter-PDU si actype==Parameter, paramCheck-PDU si actype==ParamChecket et timer-PDU si actype==Timer. Génère plusieurs transientAck-PDU en fonction de la taille des données indiquées comme argument si actype==TransientAck.
Eeque(pdus)	Place en file d'attente les pdu indiquées comme arguments. Place les PDU en file d'attente et les classe selon la valeur seqNumber lorsque plusieurs PDU sont données.
EequeND(pdus)	Place en file d'attente les PDU données comme arguments (pour limiter le nombre de transmissions de nœuds). Si plusieurs PDU sont spécifiées, la mise en file d'attente est exécutée dans un ordre défini par la valeur seqNumber des PDU.
Deque()	Récupération du début de la file d'attente.
DequeND()	Récupération du début de la file d'attente (pour limiter le nombre de transmissions de nœuds).
QueLen()	Retour de la taille de la file d'attente.
QueLenND()	Retour de la taille de la file d'attente (pour limiter le nombre de transmissions de nœuds).

8.2.2.4 Variables

Les fonctions activées lors de la transmission acyclique sont indiquées dans le Tableau 128.

Tableau 128 – Variables de transmission acyclique

Nom	Description
ACTicket	Indicateur signalant que la transmission acyclique peut avoir lieu
ACTicketND	Indicateur signalant que la transmission acyclique visant à limiter le nombre de transmissions de nœuds peut avoir lieu
TokenHopCounter	Compteur de passages de jeton
TraAllows	Nombre de fois où la transmission transitoire peut avoir lieu
TraAllowsND	Nombre de fois où la transmission transitoire peut avoir lieu (restriction du nombre de transmissions de nœuds)
TraAvailHopCounter	Valeur minimale du compteur de passage de jetons
TraLastHopCounter	Compteur de passage de jeton de dernière transmission

8.2.3 Transmission cyclique

8.2.3.1 Définition de primitive

Le FSPM émet un service CT Update.req pour la transmission cyclique. La transmission cyclique émet un service CT Update.ind pour le FSPM.

8.2.3.2 Diagramme d'états de la transmission cyclique (Cyclic transmission)

Les états du diagramme d'états de la transmission cyclique (Cyclic transmission) sont indiqués dans le Tableau 129 et les détails sont spécifiés dans le Tableau 130.

Tableau 129 – États de la transmission cyclique (Cyclic transmission)

Nom	Description
IDLE	Attente du début de la transmission cyclique
RWrRWwSENDER	Transmission RWr ou RWw en cours
RXRYSENDER	Transmission RX ou RY en cours

Tableau 130 – Table d'états de la transmission cyclique

#	Etat actuel	Événement/condition => action	Etat suivant
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
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

#	Etat actuel	Événement/condition => action	Etat suivant
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 (pas de changement d'état)
14	Any (any state)	CReceived(cyclicDataRY-PDU) => Si l'ordre du numéro de séquence est respecté: ReceivedRXRY = RetrieveRXRY(cyclicDataRY-PDU); CT Update.ind(RXRY, Offset Address, Size, ReceivedRXRY) Si l'ordre du numéro de séquence n'est pas respecté: Discarded.	Any (pas de changement d'état)
15	Any (any state)	CReceived(CyclicDataRWr-PDU) => Si l'ordre du numéro de séquence est respecté: ReceivedRWrRWw = RetrieveRWrRWw(CyclicDataRWr- PDU); CT Update.ind(RWrRWw, Offset Address, Size, ReceivedRWrRWw) Si l'ordre du numéro de séquence n'est pas respecté: Discarded.	Any (pas de changement d'état)
16	Any (any state)	CReceived(CyclicDataRWw-PDU) => Si l'ordre du numéro de séquence est respecté: ReceivedRWrRWw = RetrieveRWrRWw(CyclicDataRWw- PDU); CT Update.ind(RWrRWw, Offset Address, Size, ReceivedRWrRWw) Si l'ordre du numéro de séquence n'est pas respecté: rejet	Any (pas de changement d'état)
17	Any (any state)	CStopSending() => CTicket = FALSE	IDLE

8.2.3.3 Fonctions

Les fonctions activées lors de la transmission cyclique sont indiquées dans le Tableau 131.

Tableau 131 – Fonctions de transmission cyclique (Cyclic transmission)

Nom	Description
Length	Recherche de la taille des données indiquée par l'argument.
Mise à jour	Mise à jour des données des éléments RXRY et RWrRWw détenus par la transmission cyclique pour transmission.
CreateCyclicDataRXRY-PDU (seqno, type, RXRY)	Génère cyclicDataRX-PDU si type==RX et cyclicDataRY-PDU si type==RY. Définit la valeur seqno des bits 6..1 de seqNumber de cyclicDataRX-PDU ou cyclicDataRY-PDU. Utilise 1 468 octets dès le début si RXRY donné comme argument dépasse 1 468 octets. Les données restantes sont définies comme RemainingRXRY. Définit 1 dans le bit 7 de seqNumber du cyclicDataRX-PDU ou cyclicDataRY-PDU si la longueur (RemainingRXRY) == 0, ce qui indique que la PDU est la dernière PDU.
CreateCyclicDataRWrRWw-PDU (seqno, type, RWrRWw)	Génère cyclicDataRWr-PDU si le type==RWr et cyclicDataRWw-PDU si le type==RWw. Définit la valeur seqno des bits 6..1 de seqNumber du cyclicDataRWr-PDU ou cyclicDataRWw-PDU. Utilise 1 468 octets dès le début si RXRY donné comme argument dépasse 1 468 octets. Les données restantes sont définies comme RemainingRWrRWw. Définit 1 dans le bit 7 de seqNumber du cyclicDataRWr-PDU ou cyclicDataRWw-PDU si la longueur (RemainingRWrRWw) == 0, ce qui indique que la PDU est la dernière PDU.
RetrieveRXRY(PDU)	Récupération de l'adresse de décalage, de la taille et des données RXRY de la PDU.
RetrieveRWrRWw(PDU)	Récupération de l'adresse de décalage, de la taille et des données RWrRWw de la PDU.

8.2.3.4 Variables

Les fonctions activées lors de la transmission cyclique sont indiquées dans le Tableau 132.

Tableau 132 – Variables de la transmission cyclique (Cyclic transmission)

Nom	Description
CTicket	Indicateur signalant que la transmission cyclique peut avoir lieu.

8.2.4 Channel control (Contrôle de canal)

8.2.4.1 Diagramme d'états de contrôle de canal

Les états du diagramme d'états de contrôle de canal (Channel control) de la station principale sont décrits dans le Tableau 132, les détails étant spécifiés du Tableau 135 au Tableau 149.

Les états du diagramme d'états de contrôle de canal (Channel control) de la station subordonnée sont décrits dans le Tableau 134, les détails étant spécifiés du Tableau 150 au Tableau 155.

Tableau 133 – États du contrôle de canal de la station principale

Nom	Description
MasterDown	Non démarré
Listen	Après mise sous tension
MasterArbitration	Sélection du gestionnaire de contrôle de transmission
PrimaryMasterScatterTD	Détection du nœud connecté comme gestionnaire de contrôle de transmission
PrimaryMasterSettingUp	Réglage du chemin de passage du jeton et réglage du chemin dans les stations subordonnées comme gestionnaire de contrôle de transmission.
PrimaryMasterHoldToken	Maintien du jeton comme gestionnaire de contrôle de transmission
PrimaryMasterSolicitToken	Attente du jeton comme gestionnaire de contrôle de transmission

Nom	Description
PrimaryMasterInviting	Détection de nœud au retour au système
MasterMeasurement	Mesure des retards associés au chemin de transmission.
MasterWaitTD	Attente de retour au système d'un nœud qui n'est pas le gestionnaire de contrôle de transmission
MasterWaitSetup	Attente de réglage de chemin de passage de jeton pour un nœud qui n'est pas le gestionnaire de contrôle de transmission
MasterSolicitToken	Attente de jeton d'un nœud qui n'est pas le gestionnaire de contrôle de transmission
MasterHoldToken	Maintien de jeton d'un nœud qui n'est pas le gestionnaire de contrôle de transmission

Tableau 134 – États du contrôle de canal de la station subordonnée

Nom	Description
SlaveDown	Non démarré
SlaveWaitTD	Déconnecté
SlaveWaitSetup	Attente des informations du chemin de transmission
SlaveSolicitToken	En attente de jeton

Tableau 135 – Table d'états de station principale – MasterDown

#	Etat actuel	Événement/condition => action	Etat suivant
1	MasterDown	PowerON => ListenTimer startup; ChannelGroup = NULL	Listen

Tableau 136 – Table d'états de station principale – Listen

#	Etat actuel	Événement/condition => action	Etat suivant
1	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 => pdu = CreatePersuasion(); Send.req (sport, DA, pdu) à tous les ports autres que 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) à tous les ports autres que rport; TR9 = TRUE	MasterWaitSetup
3	Listen	Receive.ind (rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 => ListenTimer restart	Listen

#	Etat actuel	Événement/condition => action	Etat suivant
4	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que 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) à tous les ports autres que 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) à tous les ports autres que rport	Listen
9	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que 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) à tous les ports autres que rport; ListenTimer restart	Listen
11	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ListenTimer restart	Listen
12	Listen	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	Listen
13	Listen	ListenTimer time-out => ListenTimer stop; TR3 = TRUE	PrimaryMasterScatterTD

Tableau 137 – Table d'états de station principale – MasterArbitration

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

#	Etat actuel	Événement/condition => action	Etat suivant
11	MasterArbitration	SendArbTimer time-out => pdu = CreatePersuasion(); Send.req (port, BROADCAST, pdu) à tous les ports; SendArbTimer restart	MasterArbitration
12	MasterArbitration	RvLastArbTimer timeout => SendArbTimer stop; RvLastArbTimer stop; TR3 = TRUE	PrimaryMasterScatterTD

Tableau 138 – Table d'états de station principale – PrimaryMasterScatterTD

#	Etat actuel	Événement/condition => action	Etat suivant
1	PrimaryMasterScatterTD	/ TR3 == TRUE && RingCheck == FALSE => TR3 =FALSE; Nodes=0; ANodes=0; pdu = CreateTestData(); Send.req (port, BROADCAST, pdu) à tous les ports; ChannelGroup = MCAST(MyAddr); TDAckTimer startup SynchronizationMaster=TRUE;	PrimaryMasterScatterTD
2	PrimaryMasterScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 &&CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; TDAckTimer stop; TR2 = TRUE	MasterArbitration
3	PrimaryMasterScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr &&ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; TDAckTimer stop; TR8 = TRUE	MasterWaitTD
4	PrimaryMasterScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => TDAckTimer stop; TR2 = TRUE	MasterArbitration
5	PrimaryMasterScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) == Low => Nodes++	PrimaryMasterScatterTD
6	PrimaryMasterScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) != Low => TDAckTimer stop; TR2 = TRUE	MasterArbitration

#	Etat actuel	Événement/condition => action	Etat suivant
7	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster ScatterTD
8	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster ScatterTD
9	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster ScatterTD
10	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que 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) à tous les ports autres que 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) à tous les ports autres que rport MyStatusReceived(pdu)	PrimaryMaster ScatterTD
13	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu)	PrimaryMaster ScatterTD
14	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	PrimaryMaster ScatterTD
15	PrimaryMaster ScatterTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster ScatterTD
16	PrimaryMaster ScatterTD	TDackTimer time-out / Nodes > 0 => TDackTimer stop; TR4 = TRUE	PrimaryMaster SettingUp
17	PrimaryMaster ScatterTD	TDackTimer time-out / Nodes == 0 => TDackTimer stop; TR3 = TRUE	PrimaryMaster ScatterTD

#	Etat actuel	Événement/condition => action	Etat suivant
18	PrimaryMaster ScatterTD	/ TR3 == TRUE && RingCheck == TRUE => TR3 =FALSE; Nodes=0;ANodes=0; Invalidate port 2; pdu = CreateTestData(); Send.req (port, BROADCAST, pdu) au port 1; ChannelGroup = MCAST(MyAddr); TDackTimer startup	PrimaryMaster ScatterTD
19	PrimaryMaster ScatterTD	TDackTimer time-out / Nodes > 0 && RingCheck == TRUE && IsValidPort(port 2) != TRUE => Invalidate port 1; Validate port 2; pdu = CreateTestData(); Send.req (port, BROADCAST, pdu) au port 2; ChannelGroup = MCAST(MyAddr); TDackTimer startup	PrimaryMaster ScatterTD
20	PrimaryMaster ScatterTD	TDackTimer time-out / Nodes > 0 && RingCheck == TRUE && IsValidPort(port 1) != TRUE => TDackTimer stop; TR4 = TRUE	PrimaryMaster SettingUp

Tableau 139 – Table d'états de station principale – PrimaryMasterSettingUp

#	Etat actuel	Événement/condition => action	Etat suivant
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) à tous les 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) à tous les ports autres que 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) à tous les ports autres que 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

#	Etat actuel	Événement/condition => action	Etat suivant
5	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster SettingUp
6	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que 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) à tous les 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) à tous les ports autres que rport	PrimaryMaster SettingUp
10	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que 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) à tous les ports autres que 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) à tous les ports autres que 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) à tous les ports autres que 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) à tous les ports autres que rport	PrimaryMaster SettingUp

#	Etat actuel	Événement/condition => action	Etat suivant
16	PrimaryMaster SettingUp	SackTimer time-out => SackTimer stop; pdu = CreatePersuasion(); Send.req (port, BROADCAST, pdu) à tous les 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() == TRUE then Invalidate port 2; CreateTokenRoute(); pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) à tous les ports; SackTimer startup	PrimaryMaster SettingUp
18	PrimaryMaster SettingUp	/ TR4 == TRUE && RingCheck == TRUE && (IsValidPort(port 1) == FALSE IsValidPort(port 2) == FALSE) && IsSlavePortInvalidated() == FALSE => TR4 = FALSE;SARcvd=0; Nodes+=ANodes;CTRProgress = NotComplete; CreateTokenRoute(); pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) à tous les ports; SackTimer startup	PrimaryMaster SettingUp
19	PrimaryMaster SettingUp	/ RingCheck == TRUE && IsValidPort(port 1) == TRUE && IsValidPort(port 2) == TRUE && IsSlavePortInvalidated() == TRUE => TR4 = FALSE;SARcvd=0; Nodes+=ANodes; CTRProgress = NotComplete; CreateTokenRoute(); pdu = CreateSetup(); Send.req (port, destination MAC address, pdu) à tous les ports; SackTimer startup	PrimaryMaster SettingUp
20	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / RingChecked == TRUE && DA == MyAddr && SA != MyAddr && ftype == 0x14 && SARcvd+1 == SetupNodes => SackTimer stop; Invalidation de port connecté à un port invalide de la station subordonnée; pdu = CreateSetup(); Send.req (port, MAC address of slave with invalid port, pdu); SlavePortValidating=TRUE; SackTimer startup	PrimaryMaster SettingUp

#	Etat actuel	Événement/condition => action	Etat suivant
21	PrimaryMaster SettingUp	Receive.ind(rport, DA, SA, ftype, pdu) / RingChecked == TRUE && DA == MyAddr && SA != MyAddr && ftype == 0x14 && SlavePortValidating == TRUE => SlavePortValidating = FALSE; SAckTimer stop; TR5 = TRUE	PrimaryMaster HoldToken

Tableau 140 – Table d'états de station principale – PrimaryMasterHoldToken

#	Etat actuel	Événement/condition => action	Etat suivant
1	PrimaryMaster HoldToken	/ TR5 == TRUE => TR5 = FALSE; SetInvitationFlag(); SetMeasureFlag(); CTicket = FALSE; sends = 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) à tous les 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

#	Etat actuel	Événement/condition => action	Etat suivant
7	PrimaryMaster HoldToken	/ PH == 12 => ACTicket = TRUE; SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMaster HoldToken
8	PrimaryMaster HoldToken	/ PH == 20 && ((ACTicketND == FALSE) && (ACTicket == FALSE)) => SetNextPhase(PrimaryMasterHoldToken,PH);	PrimaryMaster Inviting
9	PrimaryMaster HoldToken	/ PH == 3 && sents < Multiple Transmit => sents = sents + 1; PduNum = QueLen(); while (PduNum > 0) pdu = Deque(); Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu); PduNum = PduNum -1	PrimaryMaster HoldToken
10	PrimaryMaster HoldToken	/ PH == 3 && sents == Multiple Transmit => QueDelete(); pdu = CreateToken(); Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu); sents = 1; PH = 4	PrimaryMaster HoldToken
11	PrimaryMaster HoldToken	/ PH == 4 && sents < Multiple Token => pdu = Deque(); Send.req (port, ChannelGroup, pdu) à tous les 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) à tous les ports autres que 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) à tous les ports autres que rport; TR8 = TRUE	MasterWaitTD
15	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => TR2 = TRUE	MasterArbitrati on

#	Etat actuel	Événement/condition => action	Etat suivant
16	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster HoldToken
17	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster HoldToken
18	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster HoldToken
19	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster HoldToken
20	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) à tous les ports autres que rport; CReceived(pdu)	PrimaryMaster HoldToken
21	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) à tous les ports autres que rport; MyStatusReceived(pdu)	PrimaryMaster HoldToken
22	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu)	PrimaryMaster HoldToken
23	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	PrimaryMaster HoldToken
24	PrimaryMaster HoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster HoldToken
25	PrimaryMaster HoldToken	SendCyclic(pdu) => Send.req (sport, ChannelGroup, pdu) à tous les ports; Enque(pdu)	PrimaryMaster HoldToken
26	PrimaryMaster HoldToken	/ MeasureFlag == ON => MeasureFlag = NULL TR12 = TRUE	MasterMeasur ement

Tableau 141 – Table d'états de station principale – PrimaryMasterSolicitToken

#	Etat actuel	Événement/condition => action	Etat suivant
1	PrimaryMasterSolicitToken	/ TR6 == TRUE => TR6 = FALSE; NetWatchTimer startup; SetNewTokenFlag()	PrimaryMasterSolicitToken
2	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; NetWatchTimer stop; TR2 = TRUE	MasterArbitration
3	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; NetWatchTimer stop; TR8 = TRUE	MasterWaitTD
4	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => NetWatchTimer stop; TR2 = TRUE	MasterArbitration
5	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMasterSolicitToken
6	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMasterSolicitToken
7	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMasterSolicitToken
8	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) à tous les ports autres que rport; NetWatchTimer stop; AddReceivedToken(pdu.tseq_no); TR5 = TRUE	PrimaryMasterHoldToken
9	PrimaryMasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== TRUE => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMasterSolicitToken

#	Etat actuel	Événement/condition => action	Etat suivant
10	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport; NetWatchTimer restart	PrimaryMaster SolicitToken
11	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) à tous les ports autres que rport; CReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
12	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) à tous les ports autres que rport; MyStatusReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
13	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
14	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu); NetWatchTimer restart	PrimaryMaster SolicitToken
15	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMaster SolicitToken
16	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() != MyAddr && NewTokenFlag == ON && RingCheck == FALSE => NetWatchTimer stop; pdu = CreateToken(); Send.req (sport, LostTokenDst(), pdu) à tous les ports; NetWatchTimer restart	PrimaryMaster SolicitToken
17	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() != MyAddr && NewTokenFlag == OFF && RingCheck == FALSE => NetWatchTimer stop; pdu = CreatePersuasion(); Send.req (sport, BROADCAST, pdu) à tous les ports; TR3 = TRUE	PrimaryMaster ScatterTD

#	Etat actuel	Événement/condition => action	Etat suivant
18	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() == MyAddr => NetWatchTimer stop; pdu = CreateToken(); Send.req (sport, MyAddr, pdu) à tous les ports; TR5 = TRUE	PrimaryMaster HoldToken
19	PrimaryMaster SolicitToken	NetWatchTimer time-out / LostTokenDst() != MyAddr && RingCheck == TRUE => NetWatchTimer stop; Failure location assessment; pdu = CreateSetup (Invalidate failure location port); Send.req (port, failure upstream node, pdu) to valid port; UpperSlavePortInvalidating = TRUE; NetWatchTimer restart	PrimaryMaster SolicitToken
20	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x14 && RingCheck == TRUE && UpperSlavePortInvalidating == TRUE => NetWatchTimer stop; UpperSlavePortInvalidating == FALSE Validate invalid port; pdu = CreateSetup (Invalidate failure location port); Send.req (port, failure location downstream node, pdu) to validated port; LowerSlavePortInvalidating == TRUE NetWatchTimer restart	PrimaryMaster SolicitToken
21	PrimaryMaster SolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x14 && RingCheck == TRUE && LowerSlavePortInvalidating == TRUE => NetWatchTimer stop; LowerSlavePortInvalidating = FALSE pdu = CreatePersuasion(); Send.req (sport, BROADCAST, pdu) à tous les ports; TR3 = TRUE	PrimaryMaster ScatterTD

Tableau 142 – Table d'états de station principale – PrimaryMasterInviting

#	Etat actuel	Événement/condition => action	Etat suivant
1	PrimaryMasterInviting	/ TR7 == TRUE && RingCheck == FALSE => TR7 = FALSE; TDackTimer startup; ANodes = 0; pdu = CreateTestData(); Send.req (sport, BROADCAST, pdu) à tous les ports	PrimaryMasterInviting
2	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; TDackTimer stop; TR2 = TRUE	MasterArbitration
3	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; TDackTimer stop; TR8 = TRUE	MasterWaitTD
4	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x11 => TDackTimer stop; TR2 = TRUE	MasterArbitration
5	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) == Low => ANodes++	PrimaryMasterInviting
6	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x12 && CmpPriority(pdu.mst_pri, SA) != Low => TDackTimer stop; TR2 = TRUE	MasterArbitration
7	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMasterInviting
8	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMasterInviting
9	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMasterInviting
10	PrimaryMasterInviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMasterInviting

#	Etat actuel	Événement/condition => action	Etat suivant
11	PrimaryMaster Inviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) à tous les ports autres que rport; CReceived(pdu);	PrimaryMasterInvit ing
12	PrimaryMaster Inviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) à tous les ports autres que rport; MyStatusReceived(pdu);	PrimaryMasterInvit ing
13	PrimaryMaster Inviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu);	PrimaryMasterInvit ing
14	PrimaryMaster Inviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu);	PrimaryMasterInvit ing
15	PrimaryMaster Inviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	PrimaryMasterInvit ing
16	PrimaryMaster Inviting	TDAckTimer time-out / ANodes > 0 => TDAckTimer stop; TR4 = TRUE	PrimaryMasterSett ingUp
17	PrimaryMaster Inviting	TDAckTimer time-out / ANodes == 0 => TDAckTimer stop; TR5 = TRUE	PrimaryMasterHol dToken
18	PrimaryMaster Inviting	/ TR7 == TRUE && RingCheck == TRUE && SlavePortStateChanged() == TRUE => TR7 = FALSE; pdu = CreateSetup (Validate port with port status change); Send.req (sport, port with port status change, pdu) à tous les ports; SlavePortValidating = TRUE	PrimaryMasterInvit ing
19	PrimaryMaster Inviting	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x14 && RingCheck == TRUE && SlavePortValidating = TRUE => TDAckTimer startup; ANodes = 0; pdu = CreateTestData(); Send.req (sport, BROADCAST, pdu) à tous les ports	PrimaryMasterInvit ing

Tableau 143 – Table d'états de station mprincipaleaître – MasterWaitTD

#	Etat actuel	Événement/condition => action	Etat suivant
1	MasterWaitTD	/ TR8 == TRUE => TR8 = FALSE; TDWaitTimer startup; ChannelGroup = NULL	MasterWaitTD
2	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) à tous les ports autres que rport; TDWaitTimer restart	MasterWaitTD
3	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => TDAckTimer stop; pdu = CreateTestDataAck();Send.req(rport, SA, pdu); pdu = CreateTestData(); Send.req (rport, DA, pdu) à tous les ports autres que rport; TR9 = TRUE	MasterWaitSetup
4	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterWaitTD
5	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterWaitTD
6	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterWaitTD
7	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que rport; TDWaitTimer restart	MasterWaitTD
8	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE &&& SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) ftype == 0x20 ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterWaitTD
9	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu);	MasterWaitTD
10	MasterWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterWaitTD

#	Etat actuel	Événement/condition => action	Etat suivant
11	MasterWaitTD	TDWaitTimer time-out => TDWaitTimer stop; TR3 = TRUE	PrimaryMaster ScatterTD

Tableau 144 – Table d'états de station principale – MasterWaitSetup

#	Etat actuel	Événement/condition => action	Etat suivant
1	MasterWaitSetup	/ TR9 == TRUE => TR9 = FALSE; SWaitTimer startup	MasterWaitSetup
2	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; SWaitTimer stop; TR2 = TRUE	MasterArbitration
3	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == BROADCAST && SA != MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; SWaitTimer stop; TR8 = TRUE	MasterWaitTD
4	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestDataAck(); Send.req(rport, SA, pdu); pdu = CreateTestData(); Send.req (rport, DA, pdu) à tous les ports autres que rport; SWaitTimer restart	MasterWaitSetup
5	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterWaitSetup
6	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => SWaitTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, pdu); TR10 = TRUE	MasterSolicitToken
7	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterWaitSetup
8	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterWaitSetup

#	Etat actuel	Événement/condition => action	Etat suivant
9	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE &&& SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que rport; SWaitTimer restart	MasterWaitSetup
10	MasterWaitSetup	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) à tous les ports autres que rport	MasterWaitSetup
11	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu);	MasterWaitSetup
12	MasterWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterWaitSetup
13	MasterWaitSetup	SWaitTimer time-out => SWaitTimer stop; TR3 = TRUE	PrimaryMaster ScatterTD

**Tableau 145 – Table d'états de station principale – MasterSolicitToken
(sans mesure de retard du chemin de transmission)**

#	Etat actuel	Événement/condition => action	Etat suivant
1	MasterSolicitToken	/ TR10 == TRUE => TR10 = FALSE; PrevTSeqNo = NULL; LeaveTimer startup	MasterSolicitToken
2	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop	MasterArbitration
3	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop	MasterWaitTD
4	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (rport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken

#	Etat actuel	Événement/condition => action	Etat suivant
5	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
6	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, pdu); LeaveTimer startup	MasterSolicitToken
7	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
8	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
9	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no) == FALSE => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no); TR11 = TRUE	MasterHoldToken
10	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no) == TRUE => Send.req (sport, DA, pdu) à tous les ports autres que rport;	MasterSolicitToken
11	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
12	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) à tous les ports autres que rport; CReceived(pdu)	MasterSolicitToken
13	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) à tous les ports autres que rport; MyStatusReceived(pdu)	MasterSolicitToken

#	Etat actuel	Événement/condition => action	Etat suivant
14	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu)	MasterSolicitToken
15	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	MasterSolicitToken
16	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
17	MasterSolicitToken	LeaveTimer time-out => LeaveTimer stop; TR8 = TRUE	MasterWaitTD

**Tableau 146 – Table d'états de station principale – MasterSolicitToken
(avec mesure de retard du chemin de transmission)**

#	Etat actuel	Événement/condition => action	Etat suivant
1	MasterSolicitToken	/ TR10 == TRUE => TR10 = FALSE; PrevTSeqNo = NULL; LeaveTimer startup	MasterSolicitToken
2	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop	MasterArbitration
3	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop	MasterWaitTD
4	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (rport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
5	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken

#	Etat actuel	Événement/condition => action	Etat suivant
6	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, pdu); LeaveTimer startup	MasterSolicitToken
7	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
8	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
9	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no) == FALSE => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no); TR11 = TRUE	MasterHoldToken
10	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no) == TRUE => Send.req (sport, DA, pdu) à tous les ports autres que rport;	MasterSolicitToken
11	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
12	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) à tous les ports autres que rport; CReceived(pdu)	MasterSolicitToken
13	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x20 => Send.req (sport, DA, pdu) à tous les ports autres que rport; MyStatusReceived(pdu)	MasterSolicitToken
14	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu)	MasterSolicitToken

#	Etat actuel	Événement/condition => action	Etat suivant
15	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	MasterSolicitToken
16	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterSolicitToken
17	MasterSolicitToken	LeaveTimer time-out => LeaveTimer stop; TR8 = TRUE	MasterWaitTD
18	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x40 => MeasureRecieved(rport, DA, SA, pdu)	MasterSolicitToken
19	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x41 => MeasureAckRecieved(rport, DA, SA, pdu)	MasterSolicitToken
20	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x42 => OffsetRecieved(rport, DA, SA, pdu)	MasterSolicitToken
21	MasterSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x43 => UpdateRecieved(rport, DA, SA, pdu)	MasterSolicitToken

Tableau 147 – Table d'états de station principale – MasterHoldToken

#	Etat actuel	Événement/condition => action	Etat suivant
1	MasterHoldToken	/ TR11 == TRUE => TR11 = FALSE; SetNextPhase(MasterHoldToken,0);	MasterHoldToken
2	MasterHoldToken	/ PH == 1 => MyStatusSendTimingFlag = OFF; pdu = CreateMyStatus(); Send.req(port,ChannelGroup,pdu) à tous les ports; MyStatusSend(pdu); CTicket = TRUE; TokenHopCounter = token-PDU.tokenHopCounter; TraAvailHopCounter = token-PDU.traAvailHopCounter; TraLastHopCounter = token-PDU.traLastHopCounter; TraAllows = token-PDU.traAllows; TraAllowsND = GetTraAllowsND(); SetNextPhase(MasterHoldToken,0);	MasterHoldToken
3	MasterHoldToken	/ PH == 10 && CTicket == FALSE => SetNextPhase(MasterHoldToken,PH);	MasterHoldToken

#	Etat actuel	Événement/condition => action	Etat suivant
4	MasterHoldToken	/ PH == 11 => ACTicketND = TRUE; SetNextPhase(MasterHoldToken,PH);	MasterHoldToken
5	MasterHoldToken	/ PH == 12 => ACTicket = TRUE; SetNextPhase(MasterHoldToken,PH);	MasterHoldToken
6	MasterHoldToken	/ PH == 20 && ((ACTicketND == FALSE) && (ACTicket == FALSE)) => SetNextPhase(MasterHoldToken,PH);	MasterHoldToken
7	MasterHoldToken	/ PH == 3 && sents < Multiple Transmit => sents = sents + 1; PduNum = QueLen(); while (PduNum > 0) pdu = Deque(); Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu); PduNum = PduNum -1	MasterHoldToken
8	MasterHoldToken	/ PH == 3 && sents == Multiple Transmit => QueDelete(); pdu = CreateToken(); Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu); sents = 1; PH = 3	MasterHoldToken
9	MasterHoldToken	/ PH == 4 && sents < Multiple Token => pdu = Deque(); Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu); sents = sents + 1	MasterHoldToken
10	MasterHoldToken	/ PH == 4 && sents == Multiple Token => QueDelete(); PH = 0; TR10 = TRUE	MasterSolicitToken
11	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) == Low => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) à tous les ports autres que rport; TR2=TRUE	MasterArbitration
12	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 && CmpPriority(pdu.mst_pri, SA) != Low => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) à tous les ports autres que rport; TR8=TRUE	MasterWaitTD

#	Etat actuel	Événement/condition => action	Etat suivant
13	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterHoldToken
14	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterHoldToken
15	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => CStopSending(); ACStopSending(); LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU); TR10 = TRUE	MasterSolicitToken
16	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterHoldToken
17	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterHoldToken
18	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterHoldToken
19	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => CStopSending(); ACStopSending() Send.req (sport, DA, pdu) à tous les ports autres que rport; CReceived(pdu); TR10 = TRUE	MasterSolicitToken
20	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && ftype == 0x20 => CStopSending(); ACStopSending() Send.req (sport, DA, pdu) à tous les ports autres que rport; MyStatusReceived(pdu); TR10 = TRUE	MasterSolicitToken
21	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA !=MyAddr && (ftype == 0x28 ftype == 0x1C) => CStopSending(); ACStopSending() Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu); TR10 = TRUE	MasterSolicitToken

#	Etat actuel	Événement/condition => action	Etat suivant
22	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => CStopSending(); ACStopSending(); ACReceived(pdu); TR10 = TRUE	MasterSolicitToken
23	MasterHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	MasterHoldToken
24	MasterHoldToken	SendCyclic(pdu) => Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu)	MasterHoldToken
25	MasterHoldToken	SendACyclic(pdu) => Send.req (sport, ChannelGroup, pdu) à tous les ports; Enque(pdu)	MasterHoldToken

**Tableau 148 – Table d'états de station principale – MasterMeasurement
(sans fonction de mesure de retard du chemin de transmission)**

#	Etat actuel	Événement/condition => action	Etat suivant
1	MasterMeasurement	/ TRn == TRUE => TRn = FALSE	PrimaryMasterHoldToken

**Tableau 149 – Table d'états de station principale – MasterMeasurement
(avec fonction de mesure de retard du chemin de transmission)**

#	Etat actuel	Événement/condition => action	Etat suivant
1	MasterMeasurement	/ TR12 == TRUE => TR12 = FALSE MeasureTransmissionFlag = TRUE	MasterMeasurement
2	MasterMeasurement	SendMeasure(port, DA, pdu) => Send.req(port, DA, pdu)	MasterMeasurement
3	MasterMeasurement	SendUpdate(port, DA, pdu) => Send.req(port, DA, pdu)	PrimaryMasterHoldToken
4	MasterMeasurement	SendOffset(port, DA, pdu) => Send.req(port, DA, pdu)	MasterMeasurement
5	MasterMeasurement	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x41 => MeasureAckReceived()	MasterMeasurement

Tableau 150 – Table d'états de station subordonnée – SlaveDown

#	Etat actuel	Événement/condition => action	Etat suivant
1	SlaveDown	PowerON => ChannelGroup = NULL	SlaveWaitTD

Tableau 151 – Table d'états de station subordonnée – SlaveWaitTD

#	Etat actuel	Événement/condition => action	Etat suivant
1	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitTD
2	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestDataAck(); Send.req(rport, SA, pdu); pdu = CreateTestData(); Send.req (sport, DA, pdu) à tous les ports autres que rport;	SlaveWaitSetup
3	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitTD
4	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitTD
5	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitTD
6	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE &&&& SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitTD
7	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE &&&& SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) ftype == 0x20 ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitTD
8	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	SlaveWaitTD
9	SlaveWaitTD	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitTD

Tableau 152 – Table d'états de station subordonnée – SlaveWaitSetup

#	Etat actuel	Événement/condition => action	Etat suivant
1	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) à tous les ports autres que rport; ChannelGroup = NULL	SlaveWaitTD
2	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestDataAck(); Send.req(rport, SA, pdu); pdu = CreateTestData(); Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitSetup
3	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitSetup
4	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU); TR3 = TRUE	SlaveSolicitToken
5	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitSetup
6	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitSetup
7	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / SA != MyAddr && ftype == 0x15 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitSetup
8	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / IsMulticast(DA) == TRUE &&& SA !=MyAddr && ((ftype ≥ 0x82 && ftype ≤ 0x85) ftype == 0x20 ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport; CReieved(pdu)	SlaveWaitSetup
9	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	SlaveWaitSetup
10	SlaveWaitSetup	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveWaitSetup

**Tableau 153 – Table d'états de station subordonnée – SlaveSolicitToken
(sans mesure de retard du chemin de transmission)**

#	Etat actuel	Événement/condition => action	Etat suivant
1	SlaveSolicitToken	/ TR3 == TRUE => TR3 = FALSE; LeaveTimer startup	SlaveSolicitToken
2	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop; ChannelGroup = NULL	SlaveWaitTD
3	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
4	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
5	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU)	SlaveSolicitToken
6	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
7	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
8	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no); TR4 = TRUE	SlaveHoldToken

#	Etat actuel	Événement/condition => action	Etat suivant
9	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no) == TRUE => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
10	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
11	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) à tous les ports autres que rport; CReceived(pdu)	SlaveSolicitToken
12	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype ≥ 0x20 => Send.req (sport, DA, pdu) à tous les ports autres que rport; MyStatusReceived(pdu)	SlaveSolicitToken
13	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu)	SlaveSolicitToken
14	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	SlaveSolicitToken
15	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
16	SlaveSolicitToken	LeaveTimer time-out => LeaveTimer stop; ChannelGroup = NULL	SlaveWaitTD

**Tableau 154 – Table d'états de station subordonnée – SlaveSolicitToken
(avec mesure de retard du chemin de transmission)**

#	Etat actuel	Événement/condition => action	Etat suivant
1	SlaveSolicitToken	/ TR3 == TRUE => TR3 = FALSE; LeaveTimer startup	SlaveSolicitToken
2	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop; ChannelGroup = NULL	SlaveWaitTD
3	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
4	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
5	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => LeaveTimer stop; ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU)	SlaveSolicitToken
6	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
7	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
8	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no)== FALSE => Send.req (sport, DA, pdu) à tous les ports autres que rport; LeaveTimer stop; AddReceivedToken(pdu.tseq_no); TR4 = TRUE	SlaveHoldToken

#	Etat actuel	Événement/condition => action	Etat suivant
9	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr && IsReceivedToken(pdu.tseq_no) == TRUE => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
10	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
11	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => Send.req (sport, DA, pdu) à tous les ports autres que rport; CReceived(pdu)	SlaveSolicitToken
12	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype ≥ 0x20 => Send.req (sport, DA, pdu) à tous les ports autres que rport; MyStatusReceived(pdu)	SlaveSolicitToken
13	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && (ftype == 0x28 ftype == 0x1C) => Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu)	SlaveSolicitToken
14	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => ACReceived(pdu)	SlaveSolicitToken
15	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveSolicitToken
16	SlaveSolicitToken	LeaveTimer time-out => LeaveTimer stop; ChannelGroup = NULL	SlaveWaitTD
17	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x40 => MeasureRecieved(rport, DA, SA, pdu)	SlaveSolicitToken
18	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x41 => MeasureAckRecieved(rport, DA, SA, pdu)	SlaveSolicitToken

#	Etat actuel	Événement/condition => action	Etat suivant
19	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x42 => OffsetRecieved(rport, DA, SA, pdu)	SlaveSolicitToken
20	SlaveSolicitToken	Receive.ind(rport, DA, SA, ftype, pdu) / ftype == 0x43 => UpdateRecieved(rport, DA, SA, pdu)	SlaveSolicitToken

Tableau 155 – Table d'états de station subordonnée – SlaveHoldToken

#	Etat actuel	Événement/condition => action	Etat suivant
1	SlaveHoldToken	/ TR4 == TRUE => TR4 = FALSE; SetNextPhase(SlaveHoldToken,0);	SlaveHoldToken
2	SlaveHoldToken	/ PH == 1 => pdu = CreateMyStatus(); Send.req(port,ChannelGroup,pdu) à tous les ports; CTicket = TRUE; sents = 0; TokenHopCounter = token-PDU.tokenHopCounter; TraAvailHopCounter = token-PDU.traAvailHopCounter; TraLastHopCounter = token-PDU.traLastHopCounter; TraAllows = token-PDU.traAllows; TraAllowsND = GetTraAllowsND(); SetNextPhase(SlaveHoldToken,0);	SlaveHoldToken
3	SlaveHoldToken	/ PH == 10 && CTicket == FALSE => SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
4	SlaveHoldToken	/ PH == 11 => ACTicketND = TRUE; SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
5	SlaveHoldToken	/ PH == 12 => ACTicket = TRUE; SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
6	SlaveHoldToken	/ PH == 20 && ((ACTicketND == FALSE) && (ACTicket == FALSE)) => SetNextPhase(SlaveHoldToken,PH);	SlaveHoldToken
7	SlaveHoldToken	/ PH == 3 && sents < Multiple Transmit => sents = sents + 1; PduNum = QueLen(); while (PduNum > 0) pdu = Deque(); Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu); PduNum = PduNum -1	SlaveHoldToken

#	Etat actuel	Événement/condition => action	Etat suivant
8	SlaveHoldToken	/ PH == 3 && sents == Multiple Transmit => QueDelete(); pdu = CreateToken(); Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu); sents = 1; PH = 3	SlaveHoldToken
9	SlaveHoldToken	/ PH == 4 && sents == Multiple Token => pdu = Deque(); Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu); sents = sents + 1	SlaveHoldToken
10	SlaveHoldToken	/ PH == 4 && sents == Multiple Token => QueDelete(); PH = 0; TR3 = TRUE	SlaveSolicitToken
11	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x10 => Send.req (sport, DA, pdu) à tous les ports autres que rport; CStopSending(); ACStopSending(); ChannelGroup = NULL	SlaveWaitTD
12	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA==BROADCAST && SA !=MyAddr && ftype == 0x11 => pdu = CreateTestData(); Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveHoldToken
13	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x12 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveHoldToken
14	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && ftype == 0x13 => CStopSending(); ACStopSending(); ChannelGroup = MCAST(pdu.mst_addr); SetTrInfo(pdu); pdu = CreateSetupAck(); Send.req(rport, SA, SDU); TR3 = TRUE	SlaveSolicitToken
15	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x13 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveHoldToken
16	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr && ftype == 0x14 => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveHoldToken

#	Etat actuel	Événement/condition => action	Etat suivant
17	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst == MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveHoldToken
18	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == ChannelGroup && SA != MyAddr && ftype == 0x15 && pdu.token_dst != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveHoldToken
19	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype ≥ 0x82 && ftype ≤ 0x85) => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) à tous les ports autres que rport; CReceived(pdu); LeaveTimer startup	SlaveSolicitToken
20	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && ftype == 0x20 => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) à tous les ports autres que rport; MyStatusReceived(pdu); LeaveTimer startup	SlaveSolicitToken
21	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA ==ChannelGroup && SA !=MyAddr && (ftype == 0x28 ftype == 0x1C) => CStopSending(); ACStopSending(); Send.req (sport, DA, pdu) à tous les ports autres que rport; ACReceived(pdu); LeaveTimer startup	SlaveSolicitToken
22	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA == MyAddr && SA != MyAddr && (ftype == 0x23 ftype == 0x25 ftype == 0x29 ftype == 0x2E) => CStopSending(); ACStopSending(); ACReceived(pdu); LeaveTimer startup	SlaveSolicitToken
23	SlaveHoldToken	Receive.ind(rport, DA, SA, ftype, pdu) / DA != MyAddr && SA != MyAddr => Send.req (sport, DA, pdu) à tous les ports autres que rport	SlaveHoldToken
24	SlaveHoldToken	SendCyclic(pdu) => Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu)	SlaveHoldToken
25	SlaveHoldToken	SendACyclic(pdu) => Send.req (port, ChannelGroup, pdu) à tous les ports; Enque(pdu)	SlaveHoldToken

8.2.4.2 Fonctions

Les fonctions de contrôle de canal sont respectivement indiquées dans le Tableau 156 et le Tableau 157 pour les stations principales et subordonnées.

Tableau 156 – Fonctions de contrôle de canal de station principale

Nom	Description
getNextPhase(current_state, current_phase)	Réglage de la phase interne suivante (représentée par la variable PH).
CmpPriority(mst_pri, SA)	Comparaison des niveaux de priorité et retour de "Low" (Bas) ou "High" (Haut). Low: mst_pri < MyPriority (mst_pri == MyPriority && SA < MyAddr) High: mst_pri > MyPriority (mst_pri == MyPriority && SA > MyAddr)
CreateTokenRoute()	Génération du chemin de passage du jeton et définition de SetupNodes.
MCAST(mac_addr)	Réglage du bit I/G de l'adr à 1 et retour de l'adresse convertie en adresse de multidiffusion.
SetInvitationFlag()	Réglage de l'indicateur signalant si le nœud doit être retourné au système. Les valeurs disponibles sont les suivantes: ON: Retour au système OFF: Pas de retour au système NULL: Indéfini
SetMeasureFlag()	Réglage de l'indicateur signalant si le retard du chemin de transmission doit être mesuré. Les valeurs disponibles sont les suivantes: ON: Mesure du retard du chemin de transmission OFF: Pas de mesure du retard du chemin de transmission NULL: Indéfini
SetNewTokenFlag()	Réglage de l'indicateur signalant s'il est nécessaire de régénérer un jeton. Les valeurs disponibles sont les suivantes: ON: Régénération OFF: Pas de régénération
IsValidPort(port)	Evaluation de la validité du port; retour TRUE en cas de validité et FALSE dans le cas contraire.
IsSlavePortInvalidated()	Evaluation de l'existence d'une station subordonnée avec un port invalide; retour TRUE en cas d'existence et FALSE dans le cas contraire.
SlavePortStateChanged()	Retour TRUE si l'état du port de la station subordonnée est passé de "liaison déconnectée" à "liaison établie"; FALSE en l'absence de modification de "liaison déconnectée" à "liaison établie".
LostTokenDst()	Recherche de la destination du jeton perdu.
AddReceivedToken(tseq_no)	Détention du numéro de séquence du jeton reçu.
IsReceivedToken(tseq_no)	Evaluation de l'inclusion de tseq_no dans les numéros de séquence du jeton détenu par AddReceivedToken; retour TRUE en cas d'inclusion et FALSE dans le cas contraire.
IsMulticast(addr)	Retour TRUE si l'adresse est une adresse de multidiffusion; FALSE si l'adresse n'est pas une adresse de multidiffusion.
CreatePersuasion()	Création et retour de persuasion-PDU.
CreateTestData()	Création et retour de testData-PDU.
CreateTestDataAck()	Création et retour de testDataAck-PDU.
CreateSetup()	Création et retour de setup-PDU.
CreateSetupAck()	Création et retour de setupAck-PDU.
CreateToken()	Création et retour de token-PDU.
CReceived(pdu)	Remise de la PDU au diagramme d'états de transmission cyclique.

Nom	Description
ACReceived(pdu)	Remise de la PDU au diagramme d'états de transmission acyclique.
Enque(pdus)	Place en file d'attente les pdu données comme arguments. Place les PDU en file d'attente et les classe selon la valeur seqNumber lorsque plusieurs PDU sont données.
Deque()	Récupération des PDU mises en file d'attente par Enque depuis le début de la file d'attente.
QueDelete()	Suppression de toutes les PDU de la file d'attente.
QueLen()	Retour de la taille de la file d'attente.
SetTrInfo(pdu)	Détention des informations incluses dans le setup-PDU reçu comme PDU.
MyStatusSend(pdu)	Envoi de la notification de transmission de trame MyStatus
GetTraAllowsND()	Retour d'une valeur limite pour les transmissions transitoires de nœud.

Tableau 157 – Fonctions de contrôle de canal de la station subordonnée

Nom	Description
SetTrInfo(pdu)	Détention des informations incluses dans le setup-PDU reçu comme PDU.
AddReceivedToken(tseq_no)	Détention du numéro de séquence du jeton reçu.
IsReceivedToken(tseq_no)	Evaluation de l'inclusion de tseq_no dans les numéros de séquence du jeton détenu par AddReceivedToken; retour TRUE en cas d'inclusion et FALSE dans le cas contraire.
CreatePersuasion()	Création et retour de persuasion-PDU.
CreateTestDataAck()	Création et retour de testDataAck-PDU.
CreateSetupAck()	Création et retour de setupAck-PDU.
CreateToken()	Création et retour de token-PDU.
CReceived(pdu)	Remise de la PDU au diagramme d'états de transmission cyclique.
ACReceived(pdu)	Remise de la PDU au diagramme d'états de transmission acyclique.
GetTraAllowsND()	Retour d'une valeur limite pour les transmissions transitoires de nœud.

8.2.4.3 Variables

Les variables utilisées pour le contrôle de canal (Channel control) sont respectivement indiquées dans le Tableau 158 et le Tableau 159 pour les stations principales et subordonnées.

Tableau 158 – Variables de contrôle de canal de station principale

Nom	Description
ChannelGroup	Adresse multidiffusion indiquant le réseau affilié
Nodes	Indique le nombre de nœuds soumis au contrôle de transmission. (Utilisation comme compteur pour le nombre de réponses de TestDataAck.)
ANodes	Indique le nombre de réponses de TestDataAck lors du retour au système.
SetupNodes	Indique le nombre de destinations de transmission définies.
SARcvd	Indique le nombre de réponses de SetupAck.
PMChangeFlag	Indicateur signalant si le gestionnaire de contrôle de transmission doit être modifié
InvitationFlag	Indicateur signalant si le nœud doit être retourné au système
NewTokenFlag	Indicateur signalant si un nouveau jeton doit être généré après la perte d'un jeton
MyPriority	Niveau de priorité du propre nœud
MyAddr	Adresse MAC du propre nœud
BROADCAST	Adresse MAC de diffusion FF-FF-FF-FF-FF-FF

Nom	Description
CTRProgress	Etat de génération du chemin de passage de jeton
TraAllows	Nombre de fois où la transmission transitoire peut avoir lieu
TraAvailHopCounter	Valeur minimale du nombre de transferts de token-PDU dans lesquels une transmission transitoire peut avoir lieu
TraLastHopCounter	Nombre de transferts de token-PDU de nœud dans lesquels la dernière transmission transitoire a été exécutée
RingCheck	Indicateur signalant si une vérification d'anneau doit être exécutée
SynchronizationMaster	Indicateur signalant le gestionnaire de synchronisation.
Physique	Indique la sous-phase.

Tableau 159 – Variables de contrôle de canal de la station subordonnée

Nom	Description
ChannelGroup	Adresse multidiffusion indiquant le réseau affilié
MyAddr	Adresse MAC du propre nœud
BROADCAST	Adresse MAC de diffusion FF-FF-FF-FF-FF-FF
PrevTSeqNo	Numéro de séquence de jeton reçu
Physique	Indique la sous-phase.

8.2.4.4 Timers

Les temporisateurs utilisés pour le contrôle de canal sont respectivement indiqués dans le Tableau 160 et le Tableau 161 pour les stations principales et subordonnées.

Tableau 160 – Temporisateurs de contrôle de canal de station principale

Nom	Description
ListenTimer	Détermine le moment de surveillance de la présence d'une communication au démarrage.
TDAckTimer	Détermine le retard de réponse TestDataAck.
SAckTimer	Détermine le retard de réponse SetupAck.
RvLastArbTimer	Détermine le retard de réception de Persuasion.
SendArbTimer	Détermine l'intervalle d'envoi de Persuasion.
NetWatchTimer	Détermine le moment auquel surveiller si la communication est normale.
TDWaitTimer	Détermine le retard d'attente de réception de TestData.
SWaitTimer	Détermine le retard d'attente de réception de Setup.
LeaveTimer	Détermine le temps de détection de la déconnexion.

Tableau 161 – Temporisateurs de contrôle de canal de la station subordonnée

Nom	Description
LeaveTimer	Utilisation pour la détection de la déconnexion.
MasterWatchTimer	Utilisation pour la surveillance de la station principale.

8.2.5 Parameter dist

8.2.5.1 Diagramme d'états de distribution des paramètres

Les états du diagramme d'états de distribution des paramètres (Parameter dist) de la station principale sont indiqués dans le Tableau 162 et les détails sont spécifiés dans le Tableau 164.

Les états du diagramme d'états de distribution des paramètres (Parameter dist) de la station subordonnée sont indiqués dans le Tableau 163 et les détails sont spécifiés dans le Tableau 165.

Tableau 162 – États de la distribution des paramètres de la station principale

Nom	Description
IDLE	Paramètres non distribués
InitParamChecking	Vérification initiale de l'état de la distribution des paramètres en cours
ParamSending	Distribution des paramètres en cours
ParamChecking	Vérification de l'état de la distribution des paramètres en cours
ParamSent	Distribution des paramètres terminée

Tableau 163 – États de la distribution des paramètres de la station subordonnée

Nom	Description
ParamNone	Pas de paramètre
ParamCheck	Vérification des paramètres en cours
ParamHold	Détention de paramètres; paramètres non reflétés
ParamOK	Détention de paramètres; paramètres reflétés
ParamErr	Détention de paramètres; erreur de paramètres

Tableau 164 – Table d'états de la distribution des paramètres de la station principale

#	Etat actuel	Événement/condition => action	Etat suivant
1	IDLE	ParameterDistReady() => AC Send.req(ParameterCheck, data)	InitParamChecking
2	InitParamChecking	MyStatusReceived(pdu) / MyStatusFromAllNodes() == FALSE =>	InitParamChecking
3	InitParamChecking	MyStatusReceived(pdu) / MyStatusFromAllNodes() == TRUE && ParamAllNodesReceived() == FALSE => GetNonReceivedNodes(); AC Send.req(netno, nodeno, Parameter, data)	ParamSending
4	InitParamChecking	MyStatusReceived(pdu) / MyStatusFromAllNodes() == TRUE && ParamAllNodesReceived() == TRUE => AC Send.req(ParameterCheck, data)	ParamChecking
5	ParamSending	MyStatusReceived(pdu) / ParamAllNodesReceived() == FALSE =>	ParamSending

#	Etat actuel	Événement/condition => action	Etat suivant
6	ParamSending	MyStatusReceived(pdu) / MyStatusFromAllNodes() == TRUE => AC Send.req(ParameterCheck, data)	ParamChecking
7	ParamChecking	MyStatusReceived(pdu) / ParamAllNodesChecked() == TRUE => AC Send.req(ParameterCheck, data)	ParamSent
8	ParamChecking	MyStatusReceived(pdu) / ParamAllNodesChecked() == FALSE =>	ParamChecking
9	ParamSent	MyStatusReceived(pdu) / ParamAllNodesChecked() == FALSE => GetNonCheckedNodes(); AC Send.req(netno, nodeno, Parameter, data)	ParamSending

Tableau 165 – Table d'états de la distribution des paramètres de la station subordonnée

#	Etat actuel	Événement/condition => action	Etat suivant
1	ParamNone	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck =>	ParamNone
2	ParamNone	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter => DoParameterCheck(pdu)	ParamCheck
3	ParamCheck	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter =>	ParamCheck
4	ParamCheck	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE =>	ParamCheck
5	ParamCheck	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
6	ParamCheck	ParameterCheckComplete(Result) / Result == TRUE =>	ParamHold
7	ParamCheck	ParameterCheckComplete(Result) / Result == FALSE =>	ParamErr
8	ParamErr	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter => DoParameterCheck(pdu)	ParamCheck
9	ParamErr	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE =>	ParamErr

#	Etat actuel	Événement/condition => action	Etat suivant
10	ParamErr	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
11	ParamHold	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter =>	ParamCheck
12	ParamHold	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE => UpdateParameter(pdu)	ParamOK
13	ParamHold	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
14	ParamOK	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == Parameter =>	ParamCheck
15	ParamOK	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == TRUE => UpdateParameter(pdu)	ParamOK
16	ParamOK	AC Send.ind(Dest Netno, Dest nodeno, type, data) / type == ParamCheck && CmpParamID(data) == FALSE =>	ParamNone
17	ParamOK	MasterWatchTimer time-out =>	ParamHold

8.2.5.2 Fonctions

Les fonctions de la distribution des paramètres (Parameter dist) sont respectivement indiquées dans le Tableau 166 et le Tableau 167 pour les stations principales et subordonnées.

Tableau 166 – Fonctions de la distribution des paramètres de la station principale

Nom	Description
ParameterDistReady	Prêt pour la distribution des paramètres
ParamAllNodesReeived()	Evalue si tous les nœuds ont reçu des paramètres. Retour TRUE si tel est le cas; retour FALSE dans le cas contraire.
ParamAllNodesChecked()	Evalue si tous les nœuds ont bénéficié de la vérification des paramètres. Retour TRUE si tel est le cas; retour FALSE dans le cas contraire.
MyStatusFromAllNodes()	Evalue si MyStatus a été reçu de la part de tous les nœuds. Retour TRUE si tel est le cas; retour FALSE dans le cas contraire.
GetNonReceivedNodes()	Obtention des numéros de réseaux et des numéros de nœuds des nœuds qui n'ont pas reçu de paramètres.
GetNonCheckedNodes()	Obtention des numéros de réseaux et des numéros de nœuds des nœuds qui n'ont pas vérifié les paramètres.
GetNetNo(pdu)	Obtention du numéro de réseau depuis la pdu.
GetNodeNo(pdu)	Obtention du numéro de nœud depuis la pdu.

Tableau 167 – Fonctions de la distribution des paramètres de la station subordonnée

Nom	Description
DoParameterCheck(pdu)	Vérification des paramètres donnés par la pdu. Appel de ParameterCheckComplete(result) une fois la vérification terminée. Résultat défini sur TRUE si les paramètres sont normaux et sur FALSE si les paramètres sont anormaux.
CmpParamID(pdu)	Comparaison de l'ID du paramètre donné par la PDU et de l'ID du paramètre donné par la PDU dans DoParameterCheck. Retour TRUE en cas de concordance; retour FALSE dans le cas contraire.
UpdateParameter(pdu)	Définition des paramètres donnés dans la PDU comme les nouveaux paramètres.

8.2.6 Synchronous trigger (Déclenchement synchrone)

8.2.6.1 Définition de primitive

Le déclencheur synchrone émet un service Synchronous trigger internal.ind pour le FSPM.

8.2.6.2 Diagramme d'états du déclencheur synchrone

Les états du diagramme d'états du déclencheur synchrone (Synchronous trigger) de la station principale sont indiqués dans le Tableau 168 et les détails sont spécifiés dans le Tableau 170.

Les états du diagramme d'états du déclencheur synchrone (Synchronous trigger) de la station subordonnée sont indiqués dans le Tableau 169 et les détails sont spécifiés dans le Tableau 171.

Tableau 168 – États du déclencheur synchrone (Synchronous trigger) de la station principale

Nom	Description
ACTIVE	État de fonctionnement

Tableau 169 – États du déclencheur synchrone (Synchronous trigger) de la station subordonnée

Nom	Description
ACTIVE	Etat de fonctionnement

Tableau 170 – Table d'états du déclencheur synchrone (Synchronous trigger) de la station principale

#	Etat actuel	Événement/condition => action	Etat suivant
1	ACTIVE	MyStatusSend (pdu) / (SynchronizationMaster == TRUE) && (getMyStatusSyncFlag (pdu) == TRUE) => Synchronous Trigger Internal.ind	ACTIVE
2	ACTIVE	MyStatusReceived(pdu) / (SynchronizationMaster == FALSE) && getMyStatusSyncFlag () == TRUE => Synchronous Trigger Internal.ind	ACTIVE

**Tableau 171 – Table d'états du déclencheur synchrone (Synchronous trigger)
de la station subordonnée**

#	Etat actuel	Événement/condition => action	Etat suivant
1	ACTIVE	MyStatusReceived(pdu) / getMyStatusSyncFlag () == TRUE => Synchronous Trigger Internal.ind	ACTIVE

8.2.6.3 Fonctions

Les fonctions activées pour le déclencheur synchrone (Synchronous trigger) sont indiquées dans le Tableau 172.

Tableau 172 – Fonctions du déclencheur synchrone (Synchronous trigger)

Nom	Description
getMyStatusSyncFlag	Acquisition de la valeur de SyncFlag de la trame MyStatus.

8.2.6.4 Variables

Le déclencheur synchrone n'utilise aucune variable.

8.2.7 Temporisateur

8.2.7.1 Diagramme d'états de temporisateur

Les états du diagramme d'états de temporisateur sont décrits dans le Tableau 173 et le Tableau 174, les détails sont spécifiés dans le Tableau 175 et dans le Tableau 176.

Tableau 173 – États du temporisateur – Type "au mieux" (Best effort)

Nom	Description
ACTIVE	État de fonctionnement

Tableau 174 – Tableau – États du temporisateur – Type "cycle fixe" (Fixed cycle)

Nom	Description
ACTIVE	État de fonctionnement
Attente	Etat d'attente

Tableau 175 – États du temporisateur – Type "au mieux"

#	Etat actuel	Événement/condition => action	Etat suivant
1	ACTIVE	/ => MyStatusSendTimingFlag = ON	ACTIVE

Tableau 176 – Table d'états du temporisateur – Type "cycle fixe"

#	Etat actuel	Événement/condition => action	Etat suivant
1	Attente	/ MyStatusSendTimingFlag == OFF => ScanTimer startup	ACTIVE
2	ACTIVE	ScanTimer time-out => MyStatusSendTimingFlag = ON	Attente

8.2.7.2 Fonctions

Le temporisateur n'utilise aucune fonction.

8.2.7.3 Variables

Les variables utilisées par le temporisateur sont indiquées dans le Tableau 177.

Tableau 177 – Variables du temporisateur

Nom	Description
MyStatusSendTimingFlag	Définition de l'indicateur de début de communication. ON; Démarrage de la communication OFF: Attente

8.2.7.4 Timers

Les temporisateurs utilisés par le temporisateur à cycle fixe sont indiqués dans le Tableau 178.

Tableau 178 – Temporisateur à cycle fixe

Nom	Description
ScanTimer	Détermination de la durée du cycle de communication.

8.2.8 Transmission de mesure

8.2.8.1 Définition de primitive

Le FSPM émet un service StartMeasureInternal.req pour la transmission de mesure. MeasureTransmission émet un service StartMeasureInternal.cnf pour le FSPM.

8.2.8.2 Diagramme d'états de transmission de mesure

Les états du diagramme d'états de la transmission de mesure (Measure transmission) de la station principale sont indiqués dans le Tableau 179 et les détails sont spécifiés dans le Tableau 181.

Les états du diagramme d'états de la transmission de mesure de la station subordonnée sont indiqués dans le Tableau 180 et les détails sont spécifiés dans le Tableau 182.

Tableau 179 – États de la transmission de mesure (Measure transmission) de la station principale

Nom	Description
M_IDLE	Etat initial
SM_MEASURE	Mesure comme gestionnaire de synchronisation
SM_WAIT_MEASUREACK	Attente de MeasureAck comme gestionnaire de synchronisation
M_WAIT_MEASUREACK	Mesure
M_WAIT_UPDATE	Attente de mise à jour de résultat de mesure

Tableau 180 – États de la transmission de mesure (Measure transmission) de la station subordonnée

Nom	Description
S_IDLE	attente
S_WAIT_MEASUREACK	Mesure
S_WAIT_UPDATE	Attente de mise à jour de résultat de mesure

Tableau 181 – Table d'états de la transmission de mesure (Measure transmission) de la station principale

#	Etat actuel	Événement/condition => action	Etat suivant
1	M_IDLE	Start Measure Internal.req / (SynchronizationMaster == TRUE) && (MeasureTransmission Flag == TRUE) => MeasureTransmission Flag == FALSE slaveCounterMeasureDone = 0 slaveCounterMeasureAll = getSlaveCounterMeasureAll()	SM_MEASURE
2	M_IDLE	MeasureRecieved(rport, DA, SA, pdu) / (SynchronizationMaster == FALSE) && DA == MyAddr => startRoundtripTimer(); DA = SA; pdu = CreateMeasureAck(); SendMeasureAck(rport, DA, pdu); roundtrip = stopRoundtripTimer();	M_WAIT_OFFSET
3	M_IDLE	MeasureRecieved(rport, DA, SA, pdu) / (SynchronizationMaster == FALSE) && (DA != MyAddr) => SendMeasureAck(sport, DA, pdu) à tous les ports autres que rport	M_WAIT_MEASUREACK
4	M_IDLE	Get Offset Internal.req / SynchronizationMaster == FALSE => Get Offset Internal.cnf(offset)	M_IDLE
5	SM_MEASURE	/ slaveCounterMeasureDone < slaveCounterMeasureAll => startRoundTripTimer(); DA = getTerminatedSlaveAddress(slaveCounterMeasureDone); port = getTerminatedSlaveDirection(slaveCounterMeasureDone); pdu = CreateMeasure(); SendMeasure(port, DA, pdu)	SM_WAIT_MEASURE_ACK

#	Etat actuel	Événement/condition => action	Etat suivant
6	SM_MEASURE	/ slaveCounterMeasureDone == slaveCounterMeasureAll => DA = ChannelGroup; pdu = CreateUpdate(); SendUpdate(sport, DA, pdu) à tous les ports Start Measure Internal.cnf(TRUE)	M_IDLE
7	SM_WAIT_MEASUREACK	MeasureAckRecieved() => roundtrip = stopRoundTripTimer(); offset = getOffset(roundtrip); DA = getTerminatedSlaveAddress(SlaveCounterMeasureDone); sport = getTerminatedSlaveDirection(SlaveCounterMeasureDone); pdu = CreateOffset(offset); SendOffset(sport, DA, pdu); slaveCounterMeasureDone++	SM_MEASURE
8	SM_WAIT_MEASUREACK	IsRoundtripTimerOver() => Start Measure Internal.cnf(FALSE)	M_IDLE
9	M_WAIT_MEASUREACK	MeasureAckRecieved(rport, DA, SA, pdu) => roundtrip = stopRoundtripTimer(); SendMeasureAck(sport, DA, pdu) à tous les ports autres que rport startArrivalOffsetTimer()	M_WAIT_OFFSET
10	M_WAIT_MEASUREACK	IsRoundtripTimerOver()	M_IDLE
11	M_WAIT_OFFSET	OffsetRecieved(rport, DA, SA, pdu) => roundtrip = stopRoundtripTimer(); SendOffset(sport, DA, pdu) à tous les ports autres que rport startArrivalOffsetTimer()	M_WAIT_UPDATE
12	M_WAIT_OFFSET	IsArrivalOffsetTimerOver()	M_IDLE
13	M_WAIT_UPDATE	UpdateRecieved(rport, DA, SA, pdu) => SendUpdate(sport, DA, pdu) à tous les ports autres que rport offset = generateOffset(roundtrip, distributeOffset)	M_IDLE
14	M_WAIT_UPDATE	IsArrivalUpdateTimerOver()	M_IDLE

Tableau 182 – Table d'états de la transmission de mesure (Measure transmission) de la station subordonnée

#	Etat actuel	Événement /condition => action	Etat suivant
1	S_IDLE	MeasureReceived(rport, DA, SA, pdu) / DA == MyAddr => startRoundtripTimer(); DA = SA; pdu = CreateMeasureAck(); SendMeasureAck(rport, DA, pdu); roundtrip = stopRoundtripTimer();	S_WAIT_OFFSET
2	S_IDLE	MeasureRecieved(rport, DA, SA, pdu) / DA != MyAddr => SendMeasure(sport, DA, pdu) à tous les ports autres que rport startRoundtripTimer();	S_WAIT_MEASUREACK

#	Etat actuel	Événement /condition => action	Etat suivant
3	S_IDLE	Get Offset Internal.req => Get Offset Internal.cnf(offset)	S_IDLE
4	S_WAIT_MEASUREACK	MeasureAckRecieved(rport, DA, SA, pdu) => roudtrip = stopRoundtripTimer(); SendMeasureAck(sport, DA, pdu) à tous les ports autres que rport startArrivalOffsetTimer()	S_WAIT_OFFSET
5	S_WAIT_MEASUREACK	IsRoundtripTimerOver()	S_IDLE
6	S_WAIT_OFFSET	OffsetRecieved(rport, DA, SA, pdu) => roudtrip = stopRoundtripTimer(); SendOffset(sport, DA, pdu) à tous les ports autres que rport startArrivalOffsetTimer()	S_WAIT_UPDATE
7	S_WAIT_OFFSET	IsArrivalOffsetTimerOver()	S_IDLE
8	S_WAIT_UPDATE	UpdateRecieved(rport, DA, SA, pdu) => SendUpdate(sport, DA, pdu) à tous les ports autres que rport offset = generateOffset(roundtrip, distributeOffset)	S_IDLE
9	S_WAIT_UPDATE	IsArrivalUpdateTimerOver() =>	S_IDLE

8.2.8.3 Fonctions

Les fonctions utilisées par la transmission de mesure sont respectivement indiquées dans le Tableau 183 et le Tableau 184 pour les stations principales et subordonnées.

Tableau 183 – Fonctions de la transmission de mesure (Measure transmission) de la station principale

Nom	Description
getSlaveCounterMeasureAll()	Acquisition du nombre de stations subordonnées qui procèdent à la mesure du retard.
getTerminatedSlaveAddress(s)	Acquisition de l'adresse MAC de la station subordonnée de fin.
getTerminatedSlaveDirection(s)	Acquisition du port connecté à la station subordonnée de fin.
getOffset(roundtrip)	Acquisition de la valeur de décalage de l'aller-retour.
CreateMeasure()	Création et retour de mesure-PDU.
CreateMeasureAck()	Création et retour de mesureAck-PDU.
CreateOffset(offset)	Création et retour de offset-PDU qui reflète la valeur de décalage.
CreateUpdate()	Création et retour de update-PDU.
SendMeasure(port, DA, pdu)	Envoi du port, de la DA et de la PDU au diagramme d'états de contrôle de canal.
SendMeasureAck(port, DA, pdu)	Envoi du port, de la DA et de la PDU au diagramme d'états de contrôle de canal.
SendOffset(port, DA, pdu)	Envoi du port, de la DA, de la SA et de la PDU au diagramme d'états de contrôle de canal.
SendUpdate(port, DA, pdu)	Envoi du port, de la DA et de la PDU au diagramme d'états de contrôle de canal.
startRoundtripTimer()	Démarrage de RoundTripTimer.
stopRoudtripTimer()	Arrêt de RoundTripTimer et retour de la valeur RoundTripTimer.

Nom	Description
IsRoundtripTimerOver()	Retour TRUE en cas de temporisation RoundTripTimer et FALSE dans tous les autres cas.
startArrivalOffsetTimer()	Démarrage de ArrivalOffsetTimer.
stopArrivalOffsetTimer()	Arrêt de ArrivalOffsetTimer.
IsArrivalOffsetTimerOver()	Retour TRUE en cas de temporisation theArrivalOffsetTimer et FALSE dans tous les autres cas.
startArrivalUpdateTimer()	Démarrage de ArrivalUpdateTimer.
stopArrivalUpdateTimer()	Arrêt de ArrivalUpdateTimer.
IsArrivalUpdateTimerOver()	Retour TRUE en cas de temporisation ArrivalUpdateTimer et FALSE dans tous les autres cas.

Tableau 184 – Fonctions de la transmission de mesure (Measure transmission) de la station subordonnée

Nom	Description
CreateMeasure()	Création et retour de mesure-PDU.
CreateMeasureAck()	Création et retour de measureAck-PDU.
CreateUpdate()	Création et retour de update-PDU.
SendMeasure(port, DA, pdu)	Envoi du port, de la DA et de la PDU au diagramme d'états de contrôle de canal.
SendMeasureAck(port, DA, pdu)	Envoi du port, de la DA et de la PDU au diagramme d'états de contrôle de canal.
SendUpdate(port, DA, pdu)	Envoi du port, de la DA et de la PDU au diagramme d'états de contrôle de canal.
startRoundtripTimer()	Démarrage de RoundTripTimer.
stopRoudtripTimer()	Arrêt de RoundTripTimer et retour de la valeur RoundTripTimer.
IsRoundtripTimerOver()	Retour TRUE en cas de temporisation RoundTripTimer et FALSE dans tous les autres cas.
startArrivalOffsetTimer()	Démarrage de ArrivalOffsetTimer.
stopArrivalOffsetTimer()	Arrêt de ArrivalOffsetTimer.
IsArrivalOffsetTimerOver()	Retour TRUE en cas de temporisation ArrivalOffsetTimer et FALSE dans tous les autres cas.
startArrivalUpdateTimer()	Démarrage de ArrivalUpdateTimer.
stopArrivalUpdateTimer()	Arrêt de ArrivalUpdateTimer.
IsArrivalUpdateTimerOver()	Retour TRUE en cas de temporisation ArrivalUpdateTimer et FALSE dans tous les autres cas.

8.2.8.4 Variables

Les variables de la transmission de mesure sont indiquées dans le Tableau 185.

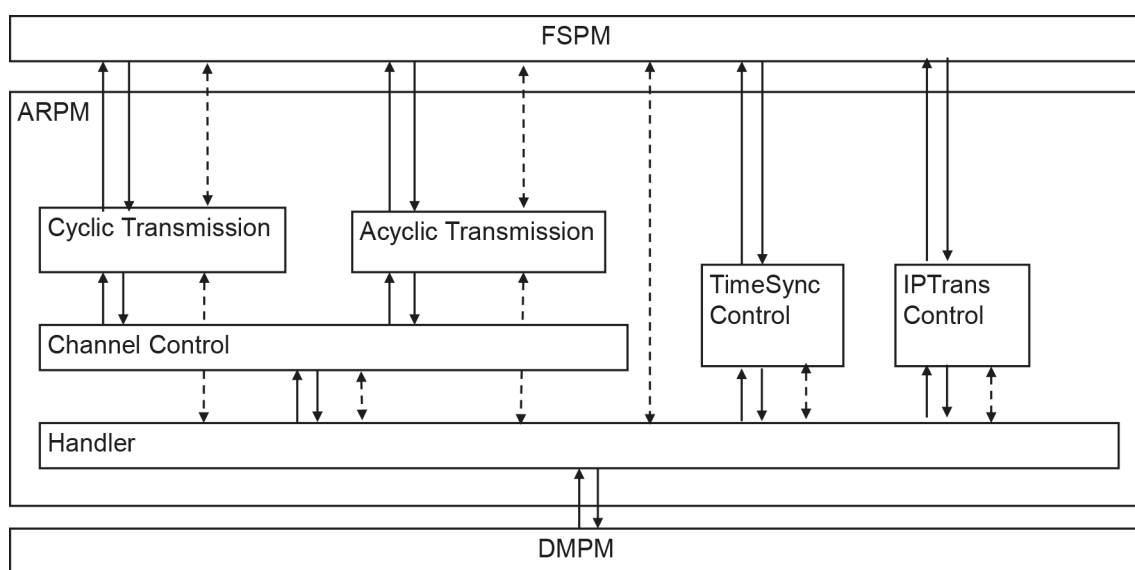
Tableau 185 – Variables de la transmission de mesure (Measure transmission) de la station principale

Nom	Description
slaveCounterMeasureDone	Indique le nombre de stations subordonnées pour lesquelles la mesure du retard est terminée.
SlaveCounterMeasureAll	Indique le nombre total de stations subordonnées pour lesquelles la mesure du retard est exécutée.
roundtrip	Indique la durée de l'aller-retour.
offset	Indique la valeur de décalage.
roundTrip	Indique la durée de l'aller-retour.
offset	Indique la valeur de décalage.

8.3 ARPM – Type T

8.3.1 Vue d'ensemble

L'ARPM offre une interface avec FSPM ou DMPM. Comme représenté à la Figure 23, l'ARPM est constituée de 6 machines de sous-protocole. Les traits pleins indiquent l'émission d'un service, et les traits discontinus indiquent les liaisons entre les machines de protocole utilisant des variables.



IEC

Anglais	Français
Cyclic Transmission	Cyclic transmission (Transmission cyclique)
Acyclic Transmission	Acyclic transmission (Transmission acyclique)
Channel Control	Channel control (Contrôle de canal)
Handler	Handler
TimeSync Control	TimeSync Control
IPTrans Control	IPTrans Control

Figure 23 – Structure du type T d'ARPM

8.3.2 Cyclic transmission (Transmission cyclique)

8.3.2.1 Définition de primitive

Les services fournis par la transmission cyclique sont représentés dans le Tableau 186.

Tableau 186 – Primitives fournies par la transmission cyclique

Nom de primitive	Editeur
C_Update.req	FSPM
C_Update.ind	ARPM

8.3.2.2 Diagramme d'états de la transmission cyclique (Cyclic transmission)

La description détaillée du diagramme d'états de la transmission cyclique est représentée dans le Tableau 187.

Tableau 187 – Etat de la transmission cyclique (Cyclic transmission)

Nom	Description
INACTIVE	Attente d'activation
ACTIVE	Transmission cyclique en cours

La description détaillée du diagramme d'états de la transmission cyclique est indiquée dans le Tableau 188.

Tableau 188 – Table d'états de la transmission cyclique

#	Etat actuel	Evénement/Condition => Action	Etat suivant
1	INACTIVE	/ CTicket = TRUE => seqNo = 0	ACTIVE
2	INACTIVE	C_Update.req(fType, Address, Size, Data) => Update(Address, Size, Data)	INACTIVE
3	ACTIVE	/ Length(RemainingData) > 0 => subPayloads = CreateSubPayloadBlock(DA, TxAsyncInfo, seqNo) pdu = CreateCyclic-PDU(fType, subPayloads) SendCyclic.req(pdu)	ACTIVE
4	ACTIVE	/ Length(RemainingData) == 0 => CTicket = FALSE	INACTIVE
5	ACTIVE	SendCyclic.ind(pdu) => C_Update.ind(fType, Address, Size, Data)	ACTIVE
6	INACTIVE	SendCyclic.ind(pdu) => C_Update.ind(fType, Address, Size, Data)	INACTIVE

8.3.2.3 Fonction

Les fonctions activées pour la transmission cyclique sont représentées dans le Tableau 189.

Tableau 189 – Fonctions utilisées pour la transmission cyclique

Nom	Contenu
Length(data)	Recherche de la taille des données indiquée par l'argument
Update()	Mise à jour des données à envoyer par transmission cyclique
CreatSubPayloadBlock(DA, TxAsyncInfo, seqNo)	La SubPayload ou SubPayloadBlock correspondante est générée au moyen de l'adresse de destination, de la demande de transmission transitoire et du numéro séquentiel, et les données restantes sont définies comme RemainingData
CreateCyclic-PDU(fType, subPayloads)	Génère la trame de transmission cyclique spécifiée par le type de trame

8.3.2.4 Variable

Les variables utilisées pour la transmission cyclique sont représentées dans le Tableau 190.

Tableau 190 – Variables utilisées dans la transmission cyclique

Nom	Contenu
CTicket	Le fanion qui représente une autorisation de transmission de la transmission cyclique
RemainingData	Données restantes à envoyer dans la transmission cyclique

8.3.3 Acyclic transmission (Transmission acyclique)

8.3.3.1 Définition de primitive

Les services fournis par la transmission acyclique sont représentés dans le Tableau 191.

Tableau 191 – Primitives fournies par la transmission acyclique

Nom de primitive	Editeur
AC_Update.req	FSPM
AC_Update.ind	ARPM

8.3.3.2 Diagramme d'états de la transmission acyclique (Acyclic transmission)

Les états du diagramme d'états de la transmission acyclique sont représentés dans le Tableau 192.

Tableau 192 – Etats de la transmission acyclique (Acyclic transmission)

Nom	Description
INACTIVE	Pas de transmission transitoire en cours
ACTIVE	Transmission transitoire en cours

La description détaillée du diagramme d'états de la transmission acyclique est indiquée dans le Tableau 193.

Tableau 193 – Table d'états de la transmission acyclique

#	Etat actuel	Événement/Condition => Action	Etat suivant
1	INACTIVE	/ ACTicket == TRUE =>	ACTIVE
2	ACTIVE	AC_Update.req(fType, DA, Data) / fType != 0xC2 => pdu = CreateAcyclicPDU (fType, DA, Data) SendAcyclic.req(pdu)	ACTIVE
3	ACTIVE	AC_Update.req(fType, DA, Data) / (Status == C_Detection Status == S_Detection) && (fType == 0xC2) => pdu = CreateAcyclicPDU (fType, DA, Data) SendAcyclic.req(pdu) ACTicket = FALSE	INACTIVE
4	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Priority) && (fType == 0xC0) && (CmpPriority() == High) => AC_Update.ind(fType, pdu)	ACTIVE
5	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Priority) && (fType == 0xC0) && (CmpPriority() == Low) => AC_Update.ind(fType, pdu) TimerStop(MMTimer) NodeType = C SetStatus(P_ON) ACTicket = FALSE	INACTIVE
6	ACTIVE	TimerExpired(MMTimer) => ACTicket = FALSE	INACTIVE
7	ACTIVE	SendAcyclic.ind(pdu) / (Status == C_Detection Status == S_Detection) && (fType == 0xC1) && (IsPrevChanged() == TRUE) => AC_Update.ind(fType, pdu)	ACTIVE
8	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Detection) && (fType == 0xC2) => ANodes++ AC_Update.ind(fType, pdu)	ACTIVE
9	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Detection) && (fType == 0xC1) => AC_Update.ind(fType, pdu) TimerStop(CCDTimer) SetStatus(M_Priority) TimerStart(MMTimer)	ACTIVE
10	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Detection) && (fType == 0xC0) && (CmpPriority() == High) => AC_Update.ind(fType, pdu) TimerStop(CCDTimer) SetStatus(M_Priority) TimerStart(MMTimer)	ACTIVE

#	Etat actuel	Événement/Condition => Action	Etat suivant
11	ACTIVE	SendAcyclic.ind(pdu) / (Status == M_Detection) && (fType == 0xC0) && (CmpPriority() == Low) => AC_Update.ind(fType, pdu) TimerStop(CCDTimer) NodeType = C SetStatus(P_ON) ACTicket = FALSE	INACTIVE
12	ACTIVE	TimerExpired(CCDTimer) / ANodes > 0 => ACTicket = FALSE	INACTIVE
13	ACTIVE	TimerExpired(CCDTimer) / ANodes == 0 => TimerStart(CCDTimer)	ACTIVE
14	INACTIVE	SendAcyclic.ind(pdu) / (NodeType == M NodeType == C) && (Status == TimeSync Status == NetworkConfig) && (fType == 0xC0) && (CmpPriority() == High) => AC_Update.ind(fType, pdu) SetStatus(P_ON)	INACTIVE
15	INACTIVE	SendAcyclic.ind(pdu) / (NodeType == M NodeType == C) && (Status == TimeSync Status == NetworkConfig) && (fType == 0xC0) && (CmpPriority() == Low) => AC_Update.ind(fType, pdu) NodeType = C SetStatus(P_ON)	INACTIVE
16	INACTIVE	SendAcyclic.ind(pdu) / (Status == TimeSync Status == NetworkConfig) && (fType == 0xC1) && (IsPrevChanged() == FALSE) => AC_Update.ind(fType, pdu)	INACTIVE
17	ACTIVE	SendAcyclic.ind(pdu) / (Status == TS_FT) && (fType == 0xC0) && (CmpPriority() == High) => AC_Update.ind(fType, pdu) SetStatus(P_ON)	ACTIVE
18	ACTIVE	SendAcyclic.ind(pdu) / (Status == TS_FT) && (fType == 0xC0) && (CmpPriority() == Low) => AC_Update.ind(fType, pdu) NodeType = C SetStatus(P_ON) ACTicket = FALSE	INACTIVE
19	ACTIVE	TimerExpired(TSChangeTimer) => ACTicket = FALSE	INACTIVE
20	ACTIVE	SendAcyclic.ind(pdu) / (NodeType == M NodeType == C) && (Status == TS_FT) && (fType == 0xC1) => AC_Update.ind(fType, pdu) SetStatus(P_ON) ACTicket = FALSE	INACTIVE

#	Etat actuel	Événement/Condition => Action	Etat suivant
21	ACTIVE	SendAcyclic.ind(pdu) / (NodeType == M) && (Status == TS_FT) && (fType == 0xC2) => AC_Update.ind(fType, pdu) SetStatus(M_Priority) ACTicket = FALSE	INACTIVE
22	ACTIVE	SendAcyclic.ind(pdu) / (NodeType == C) && (Status == TS_FT) && (fType == 0xC2) => AC_Update.ind(fType, pdu) SetStatus(P_ON) ACTicket = FALSE	INACTIVE
23	ACTIVE	SendAcyclic.ind(pdu) / (Status == C_Detection Status == S_Detection) && (fType == 0xC1) && (IsPrevChanged() == FALSE) => AC_Update.ind(fType, pdu) ACTicket = FALSE	INACTIVE
24	INACTIVE	SendAcyclic.ind(pdu) / (Status == TimeSync Status == NetworkConfig) && (fType == 0xC1) && (IsPrevChanged() == TRUE) => AC_Update.ind(fType, pdu) SetStatus(P_ON)	INACTIVE

8.3.3.3 Fonction

Les fonctions activées pour la transmission acyclique sont représentées dans le Tableau 194 .

Tableau 194 – Fonctions utilisées pour la transmission acyclique

Nom	Contenu
CreateAcyclicPDU (fType, DA, Data)	Lorsque les données fournies comme argument pour générer la trame de transmission transitoire spécifiée par fType dépassent la taille maximale possible de la PDU, les données sont divisées et envoyées et reçues au moyen de plusieurs PDU.
TimerStart()	Démarre le temporisateur
TimerStop()	Arrête le temporisateur
TimerExpired()	Détecte la temporisation du temporisateur
CmpPriority()	Détermine la priorité en tant que station principale de gestion et renvoie Low (la priorité de la station propre est faible) ou High (la priorité de la station propre est élevée)
IsPrevChanged()	Comme différence d'informations de configuration avec la détection reçue précédemment, renvoie TRUE si elle est reçue pour la première fois ou est différente, et renvoie FALSE si elle est identique

8.3.3.4 Variable

Les variables utilisées pour la transmission acyclique sont représentées dans le Tableau 195.

Tableau 195 – Variables utilisées dans la transmission acyclique

Nom	Description
ACTicket	Fanion qui représente le début de la transmission transitoire
ANodes	Nombre de stations connectées détectées

8.3.4 Channel control (Contrôle de canal)

8.3.4.1 Définition de primitive

Les services fournis par le contrôle de canal sont représentés dans le Tableau 196.

Tableau 196 – Primitives fournies par le contrôle de canal

Nom de primitive	Editeur
SendCyclic.req	ARPM
SendCyclic.ind	ARPM
SendACyclic.req	ARPM
SendACyclic.ind	ARPM

8.3.4.2 Diagramme d'états de contrôle de canal

Les états du diagramme d'états du contrôle de canal sont représentés dans le Tableau 197.

Tableau 197 – Etat de contrôle de canal (Channel Control)

Etat	Description
INACTIVE	Attente d'activation
ACTIVE	Actif

Le diagramme d'états du contrôle de canal est représenté dans le Tableau 198.

Tableau 198 – Diagramme d'états du contrôle de canal

#	Etat actuel	Événement/Condition => Action	Etat suivant
1	INACTIVE	/ ChannelCtrl_Flag == TRUE =>	ACTIVE
2	ACTIVE	SendAcyclic.req(pdu) / (Status == M_Priority Status == M_Detection Status == C_Detection Status == S_Detection) && (CTicket == FALSE) => sdports = CheckAvaPorts() Send.req(sdport, DA, pdu)	ACTIVE
4	ACTIVE	SendCyclic.req(pdu) / Status == TS_CT => sdports = CheckAvaPorts() Send.req(sdport, DA, pdu)	ACTIVE
5	ACTIVE	SendAcyclic.req(pdu) / (Status == TS_CT) && (CTicket == FALSE) && (IsRSVAcyclic() == TRUE) => sdports = CheckAvaPorts() Send.req(sdport, DA, pdu)	ACTIVE
7	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (Status == TS_CT) && (DA == MyAdd) && (SA != MyAdd) && (fType == 0xC4 fType == 0xC5 fType == 0xC6 fType == 0xC7) => SendCyclic.ind(pdu)	ACTIVE

#	Etat actuel	Événement/Condition => Action	Etat suivant
8	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (Status == TS_CT) && (DA == BROADCAST DA == MULTICAST) && (SA != MyAdd) && (fType == 0xC4 fType == 0xC5 fType == 0xC6 fType == 0xC7) => SendCyclic.ind(pdu) sdports = CheckAvaPort() Send.req(sdports, DA, pdu)	ACTIVE
9	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == BROADCAST) && (SA != MyAdd) && (fType == 0xC0) => SendAcyclic.ind(pdu) sdports = CheckAvaPort() Send.req(sdports, BROADCAST, pdu)	ACTIVE
10	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == BROADCAST) && (SA != MyAdd) && (fType == 0xC1) => SendAcyclic.ind(pdu) pdu = CreateAcyclicPDU (fType, pdu) sdports = CheckAvaPort() Send.req(sdports, BROADCAST, pdu)	ACTIVE
11	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == MyAdd) && (SA != MyAdd) && (fType == 0xC2 fType == 0xC3) => SendAcyclic.ind(pdu)	ACTIVE
12	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA != MyAdd) && (SA != MyAdd) && (fType == 0xC2 fType == 0xC3) => sdports = CheckAvaPort() Send.req(sdports, DA, pdu)	ACTIVE
13	ACTIVE	TimerExpired(TSChangeTimer) => CTicket = FALSE ACTicket = FALSE ChannelCtrl_Flag = FALSE	INACTIVE
14	ACTIVE	/ (CTicket == FALSE) && (ACTicket == FALSE) => ChannelCtrl_Flag = FALSE	INACTIVE
15	INACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == BROADCAST) && (SA != MyAdd) && (fType == 0xC0) => SendAcyclic.ind(pdu) sdports = CheckAvaPort() Send.req(sdports, BROADCAST, pdu)	INACTIVE
16	INACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (DA == BROADCAST) && (SA != MyAdd) && (fType == 0xC1) => SendAcyclic.ind(pdu) pdu = CreateAcyclicPDU (fType, pdu) sdports = CheckAvaPort() Send.req(sdports, BROADCAST, pdu)	INACTIVE

8.3.4.3 Fonction

Les fonctions utilisées pour le contrôle de canal sont indiquées dans le Tableau 199.

Tableau 199 – Fonctions utilisées pour le contrôle de canal

Nom	Description
CheckAvaPorts()	Vérifie les ports de transmission disponibles
TimerExpired()	Détecte l'expiration du temporisateur

8.3.4.4 Variable

Les variables utilisées pour le contrôle de canal sont représentées dans le Tableau 200.

Tableau 200 – Variables utilisées dans le contrôle de canal

Nom	Description
MyAdd	Adresse de station hôte
BROADCAST	Adresse de diffusion
MULTICAST	Adresse de multidiffusion
ChannelCtrl_Flag	Fanion qui représente l'autorisation d'exécution du contrôle de canal
Statut	Indique l'état de fonctionnement actuel • Pendant la mise sous tension ou un fonctionnement exceptionnel: P_ON • Pendant l'arbitrage de la station principale de gestion: M_Priority • La station hôte est la station principale de gestion, et la configuration de connexion est en train d'être détectée: M_Detection • La station hôte est la station principale de commande, et la détection de configuration de connexion est attendue: C_Detection • La station hôte est une station subordonnée, et la détection de connexion est attendue: S_Detection • Pendant la communication de commande par CC-Link IE TSN: TS_CT • Synchronisation temporelle en cours: TimeSync • Mise en place du réseau en cours: NetworkConfig • Pendant la communication libre par TS0: TS_FT • En fonctionnement continu autre que ce qui précède: Running (En marche)

8.3.5 TimeSync Control

8.3.5.1 Définition de primitive

Les services fournis par TimeSync Control sont représentés dans le Tableau 201.

Tableau 201 – Primitives fournies par TimeSync Control

Nom de primitive	Editeur
TimeSync.req	FSPM
TimeSync.ind	ARPM

8.3.5.2 Diagramme d'états de TimeSync Control

TimeSync Control gère l'état de synchronisation temporelle basé sur l'IEEE Std 802.1AS ou l'IEEE Std 1588. Les états du diagramme d'états sont représentés dans le Tableau 202.

Tableau 202 – Etat de TimeSync Control

Nom	Description
ACTIVE	La synchronisation temporelle est active
INACTIVE	Attente d'activation de la synchronisation temporelle

La description détaillée du diagramme d'états de TimeSync Control est représentée dans le Tableau 203.

Tableau 203 – Table d'états de TimeSync Control

#	Etat actuel	Événement/Condition => Action	Etat suivant
1	INACTIVE	/ TimeSyncCtrl_Flag == TRUE => TimeSyncStatus = NotCompleted	ACTIVE
2	ACTIVE	/ TimeSyncStatus == NotCompleted => Time synchronization compliant with IEEE Std 802.1AS or IEEE Std 1588 TimeSyncStatus = CheckTimeSyncStatus()	ACTIVE
3	ACTIVE	/ TimeSyncStatus == Completed TimeSyncStatus == Error => TimeSyncCtrl_Flag = FALSE TimeSync.ind(TimeSyncStatus)	INACTIVE
4	ACTIVE	TimeSync.req(data) => TimeSync.cnf(TimeSyncManagement(data))	ACTIVE
5	INACTIVE	TimeSync.req(data) => TimeSync.cnf(TimeSyncManagement(data))	INACTIVE

8.3.5.3 Fonction

Les fonctions utilisées pour TimeSync Control sont représentées dans le Tableau 204.

Tableau 204 – Fonctions utilisées pour TimeSync Control

Nom	Description
CheckTimeSyncStatus()	Vérifie l'état d'exécution de la synchronisation temporelle - En cours: Non terminé - Terminaison normale: Terminé - Erreur: Erreur
TimeSyncManagement(data)	Selon les arguments, acquière et définit des données relatives à l'IEEE Std 802.1AS ou à l'IEEE Std 1588, et renvoie le résultat

8.3.5.4 Variable

Les variables utilisées pour TimeSync Control sont représentées dans le Tableau 205.

Tableau 205 – Variables utilisées dans TimeSync Control

Nom	Description
TimeSync_Flag	Fanion indiquant le début de la commande TimeSync
TimeSyncStatus	Etat d'exécution de la synchronisation temporelle

8.3.6 IPTrans Control

8.3.6.1 Définition de primitive

Les services fournis par IPTrans Control sont représentés dans le Tableau 206.

Tableau 206 – Primitives fournies par IPTrans Control

Nom de primitive	Editeur
SLMPSend.req	FSPM
SLMPSend.ind	ARPM

8.3.6.2 Diagramme d'états d'IPTrans Control

Les états du diagramme d'états d'IPTrans Control sont représentés dans le Tableau 207.

Tableau 207 – Etat d'IPTrans Control dans la station principale

Nom	Description
ACTIVE	Actif
INACTIVE	En attente d'être actif

Le diagramme d'états d'IPTrans Control est représenté dans le Tableau 208.

Tableau 208 – Table d'états d'IPTrans Control

#	Etat actuel	Événement/Condition => Action	Etat suivant
1	INACTIVE	/ IPTransCtrl_Flag == TRUE => NetworkConfigStatus = NotCompleted	ACTIVE
2	ACTIVE	SLMPSend.req(SlmpData) / NetworkConfigStatus == NotCompleted => pdu = CreateSlmpPdu(SlmpData) sdports = CheckAvaPorts() Send.req(sdport, pdu)	ACTIVE
3	ACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (NetworkConfigStatus == NotCompleted) && (IsSLMPPdu(pdu) == TRUE) => SLMPSend.ind(pdu) NetworkConfigStatus = CheckNetworkConfigStatus()	ACTIVE
4	ACTIVE	/ NetworkConfigStatus == Completed NetworkConfigStatus == Error => IPTransCtrl_Flag = FALSE	INACTIVE

#	Etat actuel	Événement/Condition => Action	Etat suivant
5	INACTIVE	Received.ind(rcvPort, DA, SA, fType, pdu) / (IsSLMPPdu(pdu) == TRUE) && (Status == TS_FT) => SetStatus(TimeSync)	INACTIVE

8.3.6.3 Fonction

Les fonctions utilisées pour IPTrans Control sont indiquées dans le Tableau 209.

Tableau 209 – Fonctions utilisées pour IPTrans Control

Nom	Contenu
CreatSimpPdu(SimpData)	Génère une trame IP qui inclut SimpData
CheckAvaPorts()	Vérifie les ports de transmission disponibles
CheckNetworkConfigStatus()	Vérifie l'état d'exécution du réglage du réseau • En cours: Non terminé • Terminaison normale: Terminé • Erreur: Erreur

8.3.6.4 Variable

Les variables utilisées pour IPTrans Control sont représentées dans le Tableau 210.

Tableau 210 – Variables utilisées dans IPTrans Control

Nom	Description
IPTransCtrl_Flag	Déclencheur de l'exécution de la transmission de données IP
NetworkConfigStatus	Etat d'exécution du réglage du réseau

8.3.7 Handler

8.3.7.1 Diagramme d'états du Handler

Les états du diagramme d'états du Handler sont représentés dans le Tableau 211.

Tableau 211 – Etat du Handler

Nom	Description
DOWN	Non activé
INIT	Pendant l'initialisation
Run (Exécution)	Pendant la communication en cours d'exécution

Les diagrammes d'états du Handler sont représentés dans le Tableau 212, le Tableau 213 et le Tableau 214.

Tableau 212 – Diagramme d'états DOWN

#	Etat actuel	Événement/Condition => Action	Etat suivant
1	DOWN	POWER ON => SetStatus(P_ON) NodeType = ValidNodeType()	INIT

Tableau 213 – Diagramme d'états INIT

#	Etat actuel	Événement/Condition => Action	Etat suivant
1	INIT	/ (NodeType == M) && (Status == P_ON) => SetStatus(M_Priority) TimerStart(MMTimer) CallTsCtrl(ChannelCtrl)	INIT
2	INIT	/ (NodeType == C) && (Status == P_ON) => SetStatus(C_Detection) CallTsCtrl(ChannelCtrl)	INIT
3	INIT	/ (NodeType == S) && (Status == P_ON) => SetStatus(S_Detection) CallTsCtrl(ChannelCtrl)	INIT
4	INIT	/ (NodeType == M) && (Status == M_Priority) && (ACTicket == FALSE) => SetStatus(M_Detection) ANodes = 0 TimerStart(CCDTimer) CallTsCtrl(ChannelCtrl)	INIT
5	INIT	/ (NodeType == M) && (Status == M_Detection) && (ACTicket == FALSE) => SetStatus(TimeSync) CallTsCtrl(TimeSyncCtrl)	INIT
6	INIT	/ ((NodeType == C && Status == C_Detection) (NodeType == S && Status == S_Detection)) && (ACTicket == FALSE) => SetStatus(TimeSync) CallTsCtrl(TimeSyncCtrl)	INIT
7	INIT	/ (Status == TimeSync) && (TimeSyncStatus == Completetd) => SetStatus(NetworkConfig) CallTsCtrl(IPTransCtrl)	INIT
8	INIT	/ (NodeType == M) && (Status == TimeSync) && (TimeSyncStatus == Error) => SetStatus(M_Detection) ANodes = 0 TimerStart(CCDTimer) CallTsCtrl(ChannelCtrl)	INIT
9	INIT	/ (NodeType == C) && (Status == TimeSync) && (TimeSyncStatus == Error) => SetStatus(C_Detection) CallTsCtrl(ChannelCtrl)	INIT

#	Etat actuel	Événement/Condition => Action	Etat suivant
10	INIT	/ (NodeType == S) && (Status == TimeSync) && (TimeSyncStatus == Error) => SetStatus(S_Detection) CallTsCtrl(ChannelCtrl)	INIT
11	INIT	/ (Status == NetworkConfig) && (NetworkConfigStatus == Completetd) => SetupTS() TsCounter = 0 TimerStart(TSChangeTimer) SetStatus(TsCounter) CallTsCtrl(TsCounter)	Run (Exécution)
12	INIT	/ (NodeType == M) && (Status == NetworkConfig) && (NetworkConfigStatus == Error) => SetStatus(M_Detection) ANodes = 0 TimerStart(CCDTimer) CallTsCtrl(ChannelCtrl)	INIT
13	INIT	/ (NodeType == C) && (Status == NetworkConfig) && (NetworkConfigStatus == Error) => SetStatus(C_Detection) CallTsCtrl(ChannelCtrl)	INIT
14	INIT	/ (NodeType == S) && (Status == NetworkConfig) && (NetworkConfigStatus == Error) => SetStatus(S_Detection) CallTsCtrl(ChannelCtrl)	INIT

Tableau 214 – Diagramme d'états RUN

#	Etat actuel	Événement/Condition => Action	Etat suivant
1	Run (Exécution)	TimerExpired(TSChangeTimer) => SetNextTs(TsCounter) TimerStart(TSChangeTimer) SetStatus(TsCounter) CallTsCtrl(TsCounter)	Run (Exécution)
2	Run (Exécution)	TimerExpired(TSChangeTimer) / (IsInvitation() == TRUE) && (GetNextTs() == 0) => TimerStart(TSChangeTimer) SetStatus(M_Detection) CallTsCtrl(M_Detection)	Run (Exécution)
3	Run (Exécution)	/ (TsCounter == 0) && (NodeType == M) && (Status == P_ON) => SetStatus(M_Priority) TimerStart(MMTimer) CallTsCtrl(ChannelCtrl)	Run (Exécution)
4	Run (Exécution)	/ (TsCounter == 0) && (NodeType == C) && (Status == P_ON) => SetStatus(C_Detection) CallTsCtrl(ChannelCtrl)	Run (Exécution)

#	Etat actuel	Événement/Condition => Action	Etat suivant
5	Run (Exécution)	/ (TSCounter == 0) && (NodeType == M) && (Status == M_Priority) && (ACTicket == FALSE) => SetStatus(M_Detection) ANodes = 0 TimerStart(CCDTimer) CallTsCtrl(ChannelCtrl)	Run (Exécution)
6	Run (Exécution)	/ (TSCounter == 0) && (Status == TimeSync) => SetStatus(NetworkConfig) CallTsCtrl(IPTransCtrl)	Run (Exécution)
7	Run (Exécution)	/ (TSCounter == 0) && (NodeType == C) && (Status == C_Detection) && (ACTicket == FALSE) => SetStatus(NetworkConfig) CallTsCtrl(IPTransCtrl)	Run (Exécution)
8	Run (Exécution)	/ (TSCounter == 0) && (NodeType == C) && (Status == C_Detection) && (ACTicket == FALSE) && (IsPrevChanged() == TRUE) => SetStatus(NetworkConfig) CallTsCtrl(IPTransCtrl)	Run (Exécution)
9	Run (Exécution)	/ (TSCounter == 0) && (NodeType == C) && (Status == C_Detection) && (ACTicket == FALSE) && (IsPrevChanged() == FALSE) => SetStatus(TS_FT) CallTsCtrl(TS_FT)	Run (Exécution)

8.3.7.2 Fonction

Les fonctions utilisées pour le Handler sont représentées dans le Tableau 215.

Tableau 215 – Fonctions utilisées pour le Handler

Nom	Description
SetStatus(Status)	Définit l'état de fonctionnement conformément à l'argument
SetupTS()	Définit l'intervalle de temps
SetNextTS(TsCounter)	Définit l'intervalle de temps suivant de l'argument et définit la valeur de TSChangeTimer assignée à l'intervalle de temps correspondant
CallTsCtrl(Status)	Control Channel Control, Cyclic Transmission, Acyclic Transmission, TimeSync Control et IPTrans Control selon les arguments
GetNextTs(TsCounter)	Obtient l'intervalle de temps suivant de l'argument
ValidNodeType()	Vérifie le type de nœud de la station hôte
TimerStart()	Démarre le temporisateur
TimerExpired()	Détecte la temporisation du temporisateur
IsPrevChanged()	Comme différence d'informations de configuration avec la détection reçue précédemment, renvoie TRUE si elle est reçue pour la première fois ou est différente, et renvoie FALSE si elle est identique

8.3.7.3 Variable

Les variables utilisées pour le Handler sont représentées dans le Tableau 216.

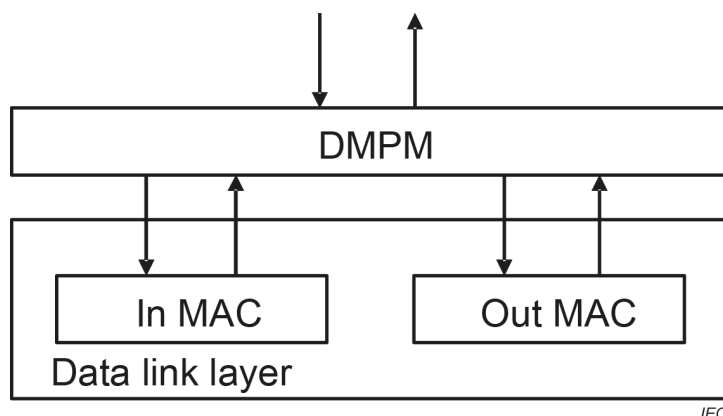
Tableau 216 – Variables utilisées pour le Handler

Nom	Description
Statut	Indique l'état de fonctionnement actuel • Pendant la mise sous tension ou un fonctionnement exceptionnel: P_ON • Pendant l'arbitrage de la station principale de gestion: M_Priority • La station hôte est la station principale de gestion, et la configuration de connexion est en train d'être détectée: M_Detection • La station hôte est la station principale de commande, et la détection de configuration de connexion est attendue: C_Detection • La station hôte est une station subordonnée, et la détection de connexion est attendue: S_Detection • Pendant la communication de commande par CC-Link IE TSN: TS_CT • Synchronisation temporelle en cours: TimeSync • Mise en place du réseau en cours: NetworkConfig • Pendant la communication libre par TS0: TS_FT • En fonctionnement continu autre que ce qui précède: Running (En marche)
NodeType	Indique le type de nœud • Station principale de gestion: O • Station principale de commande: C • Station subordonnée: S
TsCounter	Compteur pour la gestion des intervalles de temps, 0 doit être TS0

9 Machine de protocole de mapping DLL (DMPM)

9.1 DMPM – Type C

La structure du DMPM est indiquée à la Figure 24. Le DMPM mappe le service DMPM au service de couche liaison de données (service DL). Le DMPM utilise les services fournis par la sous-couche MAC de la couche liaison de données comme services DL. Le mapping de DMPM et DLL est indiqué dans le Tableau 217 et la correspondance de l'adresse de destination MAC de la PDU est indiquée dans le Tableau 218.



IEC

Anglais	Français
Data link layer	Couche liaison de données

Figure 24 – Structure du type C de DMPM

Tableau 217 – Mapping du type C de service DMPM et de service DL

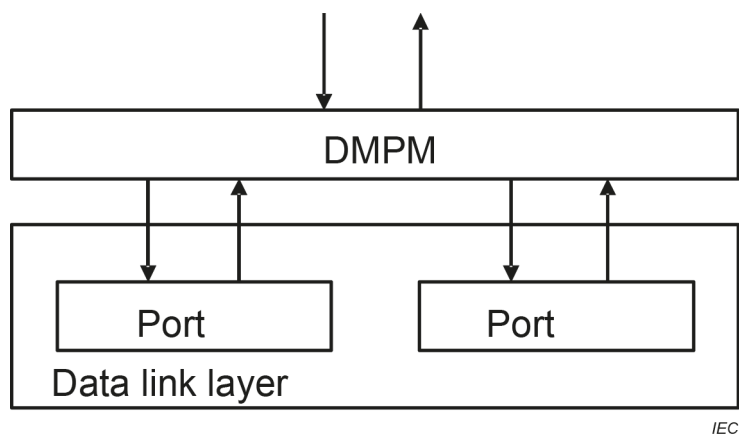
Services DMPM	Services DL	Remarques
InPort.req(PDU)	MA_DATA.req(destination_address, source_address, frame_check_sequence) dans In MAC.	Voir Tableau 218 pour destination_address. source_address est l'adresse MAC d'un nœud.
InPort.ind(PDU)	MA_DATA.ind(destination_address, source_address, PDU, frame_check_sequence) dans In MAC.	Voir Tableau 218 pour destination_address. source_address est l'adresse MAC d'une source.
InPort.ind(Port_State)	Indique l'état de la liaison de In MAC.	
OutPort.req(PDU)	MA_DATA.req(destination_address, source_address, frame_check_sequence) dans Out MAC.	Voir Tableau 218 pour destination_address. source_address est l'adresse MAC d'un nœud.
OutPort.ind(PDU)	MA_DATA.ind(destination_address, source_address, PDU, frame_check_sequence) dans Out MAC.	Voir Tableau 218 pour destination_address. source_address est l'adresse MAC d'une source.
OutPort.ind(Port_State)	Indique l'état de la liaison de Out MAC.	

Tableau 218 – Adresse de destination de chaque PDU de type C

Nom de la PDU	Destination Address (adresse de destination)
Connect-PDU	FF-FF-FF-FF-FF-FF
ConnectAck-PDU	L'adresse MAC d'un nœud auquel une réponse est donnée (source_address du Connect-PDU correspondant)
Scan-PDU	FF-FF-FF-FF-FF-FF
Collect-PDU	FF-FF-FF-FF-FF-FF
Select-PDU	FF-FF-FF-FF-FF-FF
Launch-PDU	FF-FF-FF-FF-FF-FF
PDU Token	FF-FF-FF-FF-FF-FF
MyStatus-PDU	FF-FF-FF-FF-FF-FF
Transient1-PDU	dataType == 0:FF-FF-FF-FF-FF-FF
Transient2-PDU	Une destination se trouve dans le même réseau: l'adresse MAC d'un nœud de destination Un réseau dont l'adresse de destination est différente: L'adresse MAC d'un nœud de relais
NTNTest-PDU	Adresse MAC du propre nœud.
Dummy-PDU	Adresse MAC du propre nœud.
CyclicDataW-PDU	FF-FF-FF-FF-FF-FF
CyclicDataB-PDU	FF-FF-FF-FF-FF-FF
CyclicDataOut1-PDU	FF-FF-FF-FF-FF-FF
CyclicDataOut2-PDU	FF-FF-FF-FF-FF-FF
CyclicDataIn1-PDU	FF-FF-FF-FF-FF-FF
CyclicDataIn2-PDU	FF-FF-FF-FF-FF-FF

9.2 DMPM – Type F

La structure du DMPM est indiquée à la Figure 25. Le DMPM mappe le service DMPM au service de couche liaison de données (service DL). Le DMPM utilise les services fournis par la sous-couche MAC de la couche liaison de données comme services DL. Le mapping du DMPM et du DLL est indiqué dans le Tableau 219.



Anglais	Français
Port	Accès

Figure 25 – Structure du type F de DMPM

Tableau 219 – Mapping du type F de service DMPM et de service DL

Services DMPM	Services DL	Remarques
Send.req(port, DA, pdu)	MA_DATA.req (destination_address, source_address, mac_service_data_unit, frame_check_sequence) de port	destination_address = source_address est l'adresse MAC d'un nœud.
Receive.ind(port, DA, SA, ftype, pdu)	MA_DATA.ind (destination_address, source_address, mac_service_data_unit, frame_check_sequence) de port	DA = destination_address SA = source_address ftype = mac_service_data_unit. ArFType pdu = mac_service_data_unit

9.3 DMPM – Type T

La structure de l'DMPM est indiquée à la Figure 26. La DMPM mappe le service DMPM sur le service de couche liaison de données (service DL). La DMPM utilise le service fourni par la sous-couche MAC de la couche liaison de données comme service DL. Le mapping entre la DMPM et la DLL est indiqué dans le Tableau 220.

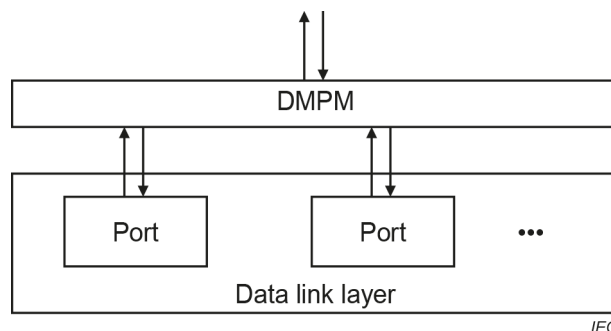


Figure 26 – Structure du type T de DMPM

Tableau 220 – Primitives fournies par DMPM

Nom de primitive	Editeur	Service DL	Remarques
Send.req(ports, DA, eType, fType, pdu)	DMPM	MA_DATA.req at a transmittable port(destination_address, source_address, mac_service_Data_unit, frame_check_sequence)	destination_address = DA source_address = SA(MAC address of node) mac_service_Data_unit = pdu
Receive.ind(port, DA, SA, eType, fType, pdu)	Couche liaison de données	MA_DATA.ind at a receive port(destination_address, source_address, eType, mac_service_Data_unit, frame_check_sequence)	DA = destination_address SA = source_address eType = mac_service_Data_unit.ethernetType PDU = mac_service_Data_unit

L'état de la DMPM est représenté dans le Tableau 221.

Tableau 221 – Etat des données cycliques (Cyclic Data)

Nom	Description
ACTIVE	État de fonctionnement

La description détaillée du diagramme d'états de DMPM est représentée dans le Tableau 222.

Tableau 222 – Table d'états de DMPM

#	Etat actuel	Événement/Condition => Action	Etat suivant
1	ACTIVE	send.req() => MAC_DATA.req()	ACTIVE
2	ACTIVE	MAC_DATA.ind() => Receive.ind	ACTIVE
3	ACTIVE	TimerExpired(MACTimer) => ignore	ACTIVE

Bibliographie

IEC 61158-6 (toutes les parties), *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6: Spécification du protocole de la couche application*

IEC 61784-1 (toutes les parties), *Réseaux industriels – Profils – Partie 1: Profils de bus de terrain – CPF x*

IEC 61784-2 (toutes les parties), *Réseaux industriels – Profils – Partie 2: Profils de bus de terrain supplémentaires pour les réseaux en temps réel fondés sur l'ISO/IEC/IEEE 8802-3 – CPF x*

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

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