

**IEEE Standard for Information technology—
Telecommunications and information exchange between systems
Local and metropolitan area networks—
Specific requirements**

Part 17: Resilient packet ring (RPR) access method and physical layer specifications

IEEE Computer Society

Sponsored by the
LAN/MAN Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

20 September 2011

IEEE Std 802.17™-2011
(Revision of IEEE Std 802.17™-2004,
as amended by IEEE Std 802.17b™-2007
and IEEE Std 802.17c™-2010)

IEEE Std 802.17™-2011
(Revision of IEEE Std 802.17™-2004,
as amended by IEEE Std 802.17b™-2007
and IEEE Std 802.17c™-2010)

**IEEE Standard for Information technology—
Telecommunications and information exchange between systems
Local and metropolitan area networks—
Specific requirements**

Part 17: Resilient packet ring (RPR) access method and physical layer specifications

Sponsor
**LAN/MAN Standards Committee
of the
IEEE Computer Society**

Approved 16 May 2011
IEEE-SA Standards Board

Abstract: This standard defines the medium access control characteristics, physical layer interface methods and layer management parameters for the resilient packet ring (RPR) access method for ring topologies. A set of protocols for detecting and initializing the shared ring configuration, recovering from failures, and regulating fair access to the shared medium are also described. Specifications are provided for interface to a number of physical layers, supporting data rates up to 10 Gb/s. System considerations and management information base (MIB) specifications are also provided in this standard.

Keywords: fairness, IEEE 802.17, layer management, metropolitan area network (MAN), medium access control, physical layer, protection switching, resilient packet ring (RPR), ring topology, topology detection

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA
Copyright © 2011 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 20 September 2011. Printed in the United States of America.

IEEE and 802 are registered trademarks in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-0-7381-6710-7 STD97140
Print: ISBN 978-0-7381-6711-4 STDPD97140

*IEEE prohibits discrimination, harassment and bullying. For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>.
No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied **“AS IS.”**

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation, or every ten years for stabilization. When a document is more than five years old and has not been reaffirmed, or more than ten years old and has not been stabilized, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon his or her independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal interpretation of the IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Recommendations to change the status of a stabilized standard should include a rationale as to why a revision or withdrawal is required. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854-4141
USA

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

This introduction is not part of IEEE Std 802.17-2011, IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 17: Resilient packet ring (RPR) access method and physical layer specifications.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association website at <http://ieeexplore.ieee.org/xpl/standards.jsp>, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA website at <http://standards.ieee.org>.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/updates/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/interp/index.html>.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. A patent holder or patent applicant has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses. Other Essential Patent Claims may exist for which a statement of assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions are reasonable or non-discriminatory. Further information may be obtained from the IEEE Standards Association.

Participants

The following individuals participated in the 802.17 Working Group during the development of this standard:

John Lemon, Chair
Michael Kelsen, Vice-Chair

Samir Abzakh
Thomas Alexander
Michael Allen
Khaled Amer
Vinay Bannai
Charles Barry
Tim Barry
Constantinos Bassias
Prasenjit Biswas
Rhett Briokovskis
Leon Bruckman
Robert Castellano
Jason Fan
Stein Gjessing
Nitin Gogate
Martin Green
John F. Hawkins
Asif Hazarika
Marc Holness
Russ Homer

Chang Huang
Wai-Chau Hui
David V. James
Peter G. Jones
Jim Kao
Philip Kruzinski
Paritosh Kulkarni
Kshitij Kumar
Robert D. Love
Derek Mayweather
Li Mo
Gal Mor
Vahid Naraghi
Paul Nikolich
Glenn W. Parsons
Chip Paryzek
Harry Peng
Tim Plunkett
Paul Quesenberry
Refael Ram
K. K. Ramakrishnan

Gady Rosenfeld
Raj Sharma
Atul Shinde
Donald Sorenson
Bob Sultan
Robert Sultan
George Suwala
Michael Takefman
Gary Turner
Necdet Uzun
Link Verstegen
Steven Wood
Donghui Xie
Li Xue
Wang Yan
Mete Yilmaz
Pinar Yilmaz
George Young
David Zelig
Daniel Zhu

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Thomas Alexander	Junghoon Jee	Glenn W. Parsons
Butch Anton	Peter G. Jones	Maximilian Riegel
Danilo Antonelli	Shinkyo Kaku	Robert Robinson
Vinay Bannai	Junghong Kao	Benjamin Rolfe
John Barr	Piotr Karocki	Randall Safier
Nancy Bravin	Michael Kelsen	John Sauer
William Byrd	Stuart J. Kerry	Bartien Sayogo
Juan Carreon	Yongbum Kim	Benson Schliesser
Keith Chow	Bruce Kraemer	Gil Shultz
Charles Cook	John Lemon	Kapil Sood
Russell Dietz	Greg Luri	Manikantan Srinivasan
Thomas Dineen	Elvis Maculuba	Thomas Starai
Sourav Dutta	Wayne W. Manges	Walter Struppler
Devon Gayle	Gary Michel	Robert Sultan
Gregory Gillooly	Michael S. Newman	Michael Takefman
Randall Groves	Charles Ngethe	William Taylor
C. Guy	Nick S. A. Nikjoo	Mark-Rene Uchida
John F. Hawkins	Paul Nikolich	Scott Valcourt
Atsushi Ito	John Notor	Prabodh Varshney
Raj Jain	Chris Osterloh	Oren Yuen

When the IEEE-SA Standards Board approved this standard on 16 May 2011, it had the following membership:

Richard H. Hulett, Chair
John Kulick, Vice Chair
Robert M. Grow, Past President
Judith Gorman, Secretary

Masayuki Ariyoshi	Jim Hughes	Gary Robinson
William Bartley	Joseph L. Koepfinger*	Jon Walter Rosdahl
Ted Burse	David J. Law	Sam Sciacca
Clint Chaplin	Thomas Lee	Mike Seavey
Weal Diab	Hung Ling	Curtis Siller
Jean-Philippe Faure	Oleg Logvinov	Phil Winston
Alexander Gelman	Ted Olsen	Howard Wolfman
Paul Houzé		Don Wright

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Michael Janezic, *NIST Representative*

Michelle Turner
IEEE Standards Program Manager, Document Development

Kathryn Bennett
IEEE Standards Program Manager, Technical Program Development

Contents

1. Overview	1
1.1 Scope	1
1.2 Purpose	1
1.3 RPR features	2
1.4 Document structure	2
2. Normative references	4
3. Terms, definitions, and notation.....	6
3.1 Conformance levels.....	6
3.2 Terms and definitions.....	6
3.3 Service definition method and notation	13
3.3.1 Classification of service primitives	13
3.4 State machines.....	14
3.4.1 State table notation	14
3.5 Arithmetic and logical operators	16
3.6 Numerical representation	16
3.7 Field notations.....	16
3.7.1 Use of italics	16
3.7.2 Field conventions.....	17
3.7.3 Field value conventions	17
3.8 Bit numbering and ordering	18
3.9 Byte-sequential formats	18
3.10 Left-to-right ordering	19
3.11 Representation of MAC addresses	20
3.12 Mapping of numeric data values to fields.....	20
3.13 Informative notes	21
3.14 Conventions for C code used in state machines.....	21
3.15 Ringlet orientation conventions	21
4. Abbreviations and acronyms	22
5. Architecture overview	24
5.1 Terminology	24
5.2 Layer model	25
5.3 Ring structure	26
5.4 Station structure	26
5.5 MAC architecture	27
5.5.1 Datapath connectivity	27
5.5.2 Ringlet selection	28
5.5.3 MAC datapath flows	29
5.5.4 Receive rules	30
5.6 MAC service	30
5.6.1 MAC data primitives	30
5.6.2 Service classes	31
5.6.3 MAC flow control primitives	32
5.7 Frame transmissions	32
5.7.1 Unicast transmissions	32
5.7.2 Flooded transmissions	33

5.7.3 Multicast transmissions	34
5.8 Frame formats	34
5.9 Frame transmissions	34
5.9.1 Local-source/local-destination transmissions	34
5.9.2 Local-source multicast transmissions.....	35
5.9.3 Local-source unknown-unicast transmissions.....	36
5.9.4 Remote-source unicast transmissions.....	36
5.9.5 Remote-source multicast transmissions	37
5.10 Spatial reuse.....	38
5.11 Bandwidth allocation	38
5.11.1 Allocation enforcement.....	39
5.11.2 Allocation consistency	39
5.12 Fairness	41
5.12.1 Equal-weighted fairness	41
5.12.2 Fairness frame distribution.....	42
5.12.3 Multi-choke fairness frame distribution.....	42
5.13 Transit-queuing options	42
5.14 Fault response methods.....	43
5.14.1 Fault response mechanisms.....	43
5.14.2 Protection hierarchy	44
5.14.3 Wrap then steer.....	45
5.15 Topology discovery	45
5.16 Frame ordering.....	46
5.16.1 Strict and relaxed transmissions.....	46
5.17 Operations, administration, and maintenance (OAM).....	46
5.17.1 Echo operations	46
5.17.2 Flushing previously sourced traffic.....	47
5.17.3 Management information base (MIB).....	47
5.18 Spatially aware sublayer	48
5.19 Protected inter-ring connection.....	48
6. Medium access control (MAC) service and reference model	49
6.1 Overview.....	49
6.2 Terminology and variables	49
6.3 Overview of MAC services	50
6.3.1 Service types	50
6.3.2 Service classes.....	51
6.4 MAC services to the client layer.....	52
6.4.1 MA_DATA.request.....	52
6.4.2 MA_DATA.indication	56
6.4.3 MA_CONTROL.request	58
6.4.4 MA_CONTROL.indication.....	59
6.5 MAC compliance test points.....	61
6.5.1 1 Gb/s PacketPHY	62
6.5.2 10 Gb/s PacketPHY	62
6.5.3 SONET/SDH	62
6.6 MAC reference model	62
6.6.1 MAC control sublayer.....	63
6.6.2 MAC datapath sublayer.....	64
6.6.3 Flow of data within the MAC	65
6.6.4 Reconciliation sublayer	67
6.6.5 Medium access control.....	67
6.6.6 Operations, administration, and maintenance (OAM)	68

6.6.7 MAC layer management entity (MLME).....	68
6.7 Protocol Implementation Conformance Statement (PICS) proforma for Clause 6	69
6.7.1 Introduction	69
6.7.2 Identification.....	69
6.7.3 PICS tables for Clause 6.....	70
7. Medium access control datapath	71
7.1 Datapath overview	71
7.2 Terminology and variables.....	72
7.2.1 Common state machine definitions	72
7.2.2 Common state machine variables.....	74
7.2.3 Common state machine routines	77
7.2.4 Variables and literals defined in other clauses	79
7.2.5 Flow count variables.....	81
7.3 Service classes.....	83
7.3.1 Service class classA.....	84
7.3.2 Service class classB	85
7.3.3 Service class classC	86
7.3.4 Reclamation.....	86
7.4 Datapaths.....	86
7.4.1 Add paths.....	86
7.4.2 Transit paths	87
7.4.3 Passthrough mode.....	88
7.4.4 Protection datapaths.....	88
7.5 Rate control	93
7.5.1 MAC shaper overview.....	93
7.5.2 Add queue flow control	94
7.5.3 IdleShaper state machine	95
7.5.4 MacControlShaper state machine.....	97
7.5.5 ClassAShaper state machine.....	98
7.5.6 ClassBShaper state machine.....	101
7.5.7 Fairness eligible shaper state machines.....	104
7.5.8 DownstreamShaper state machine.....	110
7.6 Receive operation.....	112
7.6.1 Receive operation for strict data frames.....	112
7.6.2 Reception in wrapping systems	114
7.6.3 Receive operation state machines.....	115
7.6.4 WrongRinglet state machine	145
7.7 Transmit operation	146
7.7.1 Ringlet selection	147
7.7.2 Determination of cleave point	159
7.7.3 Setting of <i>ttl</i> and <i>ttlBase</i>	160
7.7.4 StageQueueSelection state machine	163
7.7.5 DataAddCount state machine	168
7.7.6 ControlAddCount state machine	170
7.7.7 Single queue MAC design.....	172
7.7.8 Dual queue MAC design	175
7.7.9 TransmitCount state machine	179
7.7.10 TransmitRoute state machine	182
7.8 Protocol Implementation Conformance Statement (PICS) proforma for Clause 7	184
7.8.1 Introduction	184
7.8.2 Identification.....	184
7.8.3 PICS tables for Clause 7	185

8. MAC physical interface	189
8.1 Overview.....	189
8.1.1 Objectives.....	189
8.1.2 Relationship to other standards	190
8.2 MAC physical layer service interface.....	190
8.2.1 PHY_DATA.request	190
8.2.2 PHY_DATA.indication.....	191
8.2.3 PHY_LINK_STATUS.indication	192
8.2.4 Mapping of PHY_READY.indication	193
8.3 PacketPHY physical layer interfaces and PHYs.....	193
8.3.1 PacketPHY reconciliation sublayers	193
8.3.2 PacketPHYS.....	194
8.4 SONET/SDH physical layer interfaces and PHYs	195
8.4.1 SONET/SDH reconciliation sublayers.....	196
8.4.2 SONET/SDH adaptation sublayers	196
8.4.3 SONET/SDH physical layer entities (PHYS).....	198
8.5 Protocol Implementation Conformance Statement (PICS) proforma for Clause 8	200
8.5.1 Introduction	200
8.5.2 Identification	200
8.5.3 Major capabilities/options.....	201
8.5.4 PICS tables for Clause 8.....	201
9. Frame formats	203
9.1 Overview.....	203
9.2 Data frame format.....	203
9.2.1 Data frame sizes	203
9.2.2 Data frame fields	204
9.3 Control frame format	205
9.3.1 Control frame sizes.....	205
9.3.2 Control frame fields	206
9.4 Fairness frame format	208
9.4.1 Fairness frame sizes	208
9.4.2 Fairness frame fields	208
9.5 Idle frame format	209
9.5.1 Idle frame sizes.....	209
9.5.2 Idle frame fields	209
9.6 <i>baseControl</i> subfields	210
9.7 <i>extendedControl</i> subfields	212
9.8 Protocol Implementation Conformance Statement (PICS) proforma for Clause 9	214
9.8.1 Introduction	214
9.8.2 Identification	214
9.8.3 PICS tables for Clause 9	215
10. Fairness	216
10.1 Overview.....	216
10.1.1 Fairness instances.....	216
10.1.2 Services and features.....	219
10.1.3 Fairness algorithm overview	219
10.2 Terms, definitions, variables, and routines	230
10.2.1 Common state machine definitions	230
10.2.2 Common state machine variables.....	231

10.2.3	Common state machine routines	235
10.2.4	Variables and routines defined in other clauses	236
10.3	Frame formats	237
10.3.1	Fairness frame format.....	237
10.3.2	Fairness differential delay (FDD) frame format.....	238
10.4	Fairness state machines	238
10.4.1	PerByte state machine	239
10.4.2	PerAgingInterval state machine	242
10.4.3	AggressiveRateAdjust state machine	246
10.4.4	ConservativeRateAdjust state machine	248
10.4.5	PerAdvertisingInterval state machine.....	251
10.4.6	PerReportingInterval state machine	255
10.4.7	ActiveWeightsComputation state machine	256
10.4.8	FairnessFrameReceive state machine.....	258
10.4.9	FddFrameTransmit state machine	261
10.4.10	FrttComputation state machine.....	262
10.5	Explanation of aging and rates (informative)	268
10.6	Protocol Implementation Conformance Statement (PICS) proforma for Clause 10	271
10.6.1	Introduction	271
10.6.2	Identification.....	271
10.6.3	PICS tables for Clause 10	272
11.	Topology discovery and protection.....	274
11.1	Overview	274
11.1.1	Protocol overview.....	275
11.1.2	Topology database maintenance.....	276
11.1.3	Context containment.....	277
11.1.4	Secondary MAC addresses.....	278
11.1.5	LRTT measurement protocol	278
11.1.6	Fault response mechanisms	279
11.2	Terminology and variables.....	280
11.2.1	Terminology	280
11.2.2	Common state machine definitions	281
11.2.3	Common state machine variables.....	282
11.2.4	<i>ringInfo</i> fields	284
11.2.5	<i>myTopoInfo</i> fields.....	285
11.2.6	<i>topoEntry[rid][hops]</i> fields.....	287
11.2.7	Common state machine routines	288
11.2.8	Variables and routines defined in other clauses	290
11.2.9	Defect indications	290
11.3	Frame formats	292
11.3.1	Topology and protection (TP) frame format	292
11.3.2	Topology checksum (TC) frame format.....	293
11.3.3	Loop round-trip time request frame format.....	295
11.3.4	Loop round-trip time response frame format	296
11.3.5	ATD frame format	296
11.4	Defined ATT encodings	298
11.4.1	Weight ATT.....	298
11.4.2	Station bandwidth ATT	299
11.4.3	Station settings ATT	299
11.4.4	Station name ATT	300
11.4.5	Management address ATT	300
11.4.6	Station interface index ATT	301

11.4.7 Secondary MAC ATT	301
11.4.8 Organization-specific ATT	302
11.5 Topology database	303
11.5.1 Topology database structure.....	303
11.5.2 Attribute updates	307
11.5.3 Lower level representation of topology database.....	307
11.5.4 Topology change sequence	308
11.6 State machines	310
11.6.1 State machine functions	310
11.6.2 ReceiveMonitor state machine	311
11.6.3 TopologyControl state machine	313
11.6.4 ParseTpFrame state machine.....	317
11.6.5 ProtectionUpdate state machine	321
11.6.6 TopologyValidation state machine.....	330
11.6.7 TransmitTpFrame state machine.....	336
11.6.8 ReceiveTpFrame state machines.....	338
11.6.9 TransmitTcFrame state machine	340
11.6.10 ReceiveTcFrame state machine.....	342
11.6.11 Transmit rules for ATD frames	343
11.6.12 Receive rules for ATD frames	344
11.6.13 SecondaryUpdate state machine.....	345
11.6.14 TimingLrttFrame state machine	350
11.7 Protocol Implementation Conformance Statement (PICS) proforma for Clause 11	355
11.7.1 Introduction	355
11.7.2 Identification	355
11.7.3 PICS tables for Clause 11	356
12. Operations, administration, and maintenance (OAM)	360
12.1 Overview.....	360
12.1.1 Protocol overview	360
12.1.2 OAM functions supported by RPR	361
12.1.3 Fault management	361
12.1.4 Echo operations	362
12.1.5 Flush operations	362
12.1.6 Organization-specific operations.....	363
12.1.7 SAS notify operations	363
12.1.8 PIRC management operations.....	363
12.2 Terminology and variables	363
12.2.1 Common state machine definitions	363
12.2.2 Common variables.....	363
12.2.3 Common routines	364
12.2.4 Literals and routines defined in other clauses	364
12.3 OAM frame formats.....	365
12.3.1 Echo request/response payload	365
12.3.2 Flush frame.....	366
12.3.3 Organization-specific frame	367
12.3.4 SAS notify frame.....	368
12.3.5 PIRC frame.....	369
12.4 OAM service primitives.....	371
12.4.1 Echo MA_CONTROL.request.....	371
12.4.2 Echo MA_CONTROL.indication	372
12.4.3 Flush MA_CONTROL.request	372
12.4.4 Flush MA_CONTROL.indication	373

12.4.5 Organization-specific MA_CONTROL.request.....	373
12.4.6 Organization-specific MA_CONTROL.indication	374
12.4.7 SAS notify MA_CONTROL.request	375
12.5 OAM state machines.....	376
12.5.1 OamFrameTransmit state machine.....	376
12.5.2 OamFrameReceive state machine	377
12.6 Performance monitoring	380
12.6.1 Performance monitoring counters	380
12.6.2 Available and unavailable seconds.....	381
12.7 Protocol Implementation Conformance Statement (PICS) proforma for Clause 12	382
12.7.1 Introduction	382
12.7.2 Identification.....	382
12.7.3 PICS tables for Clause 12.....	383
13. Layer management entity interface.....	386
13.1 Overview of the management model	386
13.2 MLME service interface	386
13.2.1 MLME_GET.request.....	387
13.2.2 MLME_SET.request	387
13.2.3 MLME_EVENT.indication	388
13.2.4 MLME_RESET.request	389
13.3 MLME services.....	390
13.3.1 RPR interface configuration	390
13.3.2 Topology discovery monitoring	391
13.3.3 Performance and accounting measurements	391
13.4 Protocol Implementation Conformance Statement (PICS) proforma for Clause 13	394
13.4.1 Introduction	394
13.4.2 Identification.....	394
13.4.3 PICS tables for Clause 13.....	395
14. Spatially aware sublayer.....	396
14.1 Spatially aware sublayer overview	396
14.2 Terminology and variables.....	397
14.2.1 Common definitions	397
14.2.2 Common state machine variables.....	397
14.2.3 Common state machine routines	398
14.2.4 Variables and literals defined in other clauses	398
14.2.5 SAS database fields	399
14.2.6 SAS statistics counters	401
14.3 SAS association database.....	401
14.3.1 Logical representation of SAS tables	401
14.3.2 Static unicast entries	403
14.3.3 Static multicast entries.....	403
14.3.4 Dynamic entries.....	403
14.3.5 Mapping of VIDs to FIDs.....	404
14.4 SDB operations	404
14.4.1 SAS learning process.....	404
14.4.2 Querying the SDB	405
14.4.3 SDB entry removal	405
14.5 SAS operations.....	406
14.5.1 Principles of unicast operation	406
14.5.2 Principles of multicast operation	407

14.5.3	Transmit operation	409
14.5.4	SAS receive operation.....	412
14.6	Protocol Implementation Conformance Statement (PICS) proforma for Clause 14	415
14.6.1	Introduction	415
14.6.2	Identification	415
14.6.3	PICS tables for Clause 14.....	416
15.	Protected inter-ring connection	418
15.1	Protected inter-ring connection overview.....	418
15.1.1	PIRC stations.....	418
15.1.2	PIRC load balancing.....	419
15.1.3	PIRC misconfiguration detection.....	419
15.1.4	PIRC failure conditions	419
15.1.5	PIRC maintenance commands	419
15.1.6	PIRC reversion mode	420
15.1.7	PIRC protection hierarchy.....	420
15.1.8	PIRC ring-interconnection forwarding states.....	420
15.2	Terminology and variables	420
15.2.1	Common state machine definitions	420
15.2.2	Common state machine variables.....	421
15.2.3	Common state machine routines	421
15.2.4	Variables and literals defined in other clauses	422
15.3	PIRC database.....	423
15.3.1	pircProv provisioning database	423
15.3.2	pircInfo dynamic information database.....	424
15.3.3	pircMateMsg mate information database.....	424
15.4	State machines	424
15.4.1	PircMateMsgReceive state machine	424
15.4.2	PircControl state machine	425
15.4.3	PircProtectionUpdate state machine.....	427
15.4.4	PircMateMsgTransmit state machine.....	431
15.5	PIRC operations.....	432
15.5.1	PIRC receive operation	432
15.5.2	PIRC transmit operation.....	434
15.6	Protocol Implementation Conformance Statement (PICS) proforma for Clause 15	436
15.6.1	Introduction	436
15.6.2	Identification	436
15.6.3	PICS tables for Clause 15	437
	Annex A (informative) Bibliography	438
	Annex B (normative) PacketPHY reconciliation sublayers.....	440
B.1	Overview.....	440
B.2	1 Gb/s PacketPHY reconciliation sublayer (PRS-1).....	440
B.2.1	General requirements	441
B.2.2	GMII data stream	445
B.2.3	GMII functional specifications.....	445
B.2.4	Electrical characteristics.....	446
B.3	10 Gb/s PacketPHY Reconciliation Sublayer (PRS-10).....	446
B.3.1	General requirements	446
B.3.2	XGMII data stream.....	451
B.3.3	Functional specifications.....	451

B.3.4 Electrical characteristics	452
B.3.5 XGXS and XAUI	452
B.4 Protocol Implementation Conformance Statement (PICS) proforma for Annex B	453
B.4.1 Introduction	453
B.4.2 Identification.....	453
B.4.3 Major capabilities/options	454
B.4.4 PICS tables for Annex B	454
 Annex C (normative) SONET/SDH reconciliation sublayers.....	456
C.1 Overview	456
C.1.1 Relationship to other sublayers	456
C.1.2 SRS and GRS interfaces.....	456
C.1.3 Link status signals	458
C.1.4 Electrical specifications.....	459
C.2 Physical frame format for SRS and GRS.....	459
C.2.1 SRS physical frame format.....	459
C.2.2 GRS physical frame format.....	460
C.3 SRS and GRS using the 8-bit SPI-3 interface.....	462
C.3.1 General requirements.....	462
C.3.2 Mapping of SPI-3 signals to service interface primitives	465
C.3.3 SRS and GRS 8-bit SPI datastream.....	467
C.3.4 Functional specifications	467
C.3.5 Electrical specifications.....	467
C.4 SRS and GRS using the 32-bit SPI-3 interface.....	467
C.4.1 General requirements.....	467
C.4.2 Mapping of SPI-3 signals to service interface primitives	471
C.4.3 SRS and GRS 32-bit SPI datastream.....	475
C.4.4 Functional specifications	475
C.4.5 Electrical timing specifications	475
C.5 SRS and GRS using the SPI-4 Phase 1 interface	475
C.5.1 General requirements.....	475
C.5.2 Mapping of SPI-4 signals to service interface primitives	479
C.5.3 SRS and GRS 64-bit SPI datastream.....	481
C.5.4 Functional specifications	481
C.5.5 Electrical specifications.....	481
C.6 SRS and GRS using SPI-4.2 interface	481
C.6.1 General requirements.....	482
C.6.2 Mapping of SPI-4 signals to service interface primitives	485
C.6.3 SRS and GRS SPI-4.2 datastream.....	486
C.6.4 Functional specifications	486
C.6.5 Electrical specifications.....	486
C.7 Protocol Implementation Conformance Statement (PICS) proforma for Annex C	487
C.7.1 Introduction	487
C.7.2 Identification.....	487
C.7.3 Major capabilities/options	488
C.7.4 PICS tables for Annex C	488
 Annex D (normative) SNMP MIB definitions	491
D.1 Introduction	491
D.2 The SNMP management framework.....	491
D.3 Security considerations	491
D.4 MIB Structure	493

D.4.1	Structure of the MIB	493
D.5	Relationship to other MIBs.....	494
D.5.1	Relationship to the Interfaces MIB	494
D.5.2	Relationship to PHY MIBs	497
D.6	Definitions for the RPR MIB	497
	Annex E (normative) CRC and parity calculations	643
E.1	Cyclic redundancy check 16-bit (CRC16) algorithmic definition.....	643
E.1.1	Serial CRC16 calculation.....	643
E.1.2	CRC16 calculations.....	644
E.1.3	Protected header-field changes.....	644
E.1.4	Illustration of CRC16 checks	645
E.2	Cyclic redundancy check 32-bit (CRC32) algorithmic definition.....	645
E.2.1	Serial CRC32 calculation	646
E.2.2	Exchanged ExorSum calculations.....	647
E.2.3	Payload CRC stomping	647
E.2.4	Illustration of CRC32 checks	648
E.3	Parity algorithmic definition	649
E.3.1	Parity calculation.....	649
E.3.2	Illustration of fairness-frame checks	649
E.4	Protocol Implementation Conformance Statement (PICS) proforma for Annex E	650
E.4.1	Introduction	650
E.4.2	Identification	650
E.4.3	PICS tables for Annex E	651
	Annex F (informative) 802.1D and 802.1Q bridging conformance	652
F.1	Bridging overview	652
F.1.1	802 bridging reference model	652
F.1.2	RPR support of the MAC service.....	653
F.1.3	Transmission between local and remote end stations	653
F.1.4	Maintaining filtering integrity of the 802 bridged network	655
F.1.5	RPR support for basic bridging model with spatial reuse for local hosts	659
F.1.6	Duplication/misordering prevention	659
F.1.7	MAC client invocation of optional RPR service parameters	660
F.1.8	RPR requirements for 802.1D and 802.1Q bridging conformance.....	660
F.2	Architectural model of an 802.1D compliant RPR bridge.....	661
F.2.1	Bridge relay entity	661
F.2.2	Ports	661
F.2.3	Higher layer entities	661
F.3	RPR MAC Internal Sublayer Service	662
F.3.1	RPR MAC support of Internal Sublayer Service	662
F.3.2	RPR MAC support of Enhanced Internal Sublayer Service	662
F.4	Bridge protocol entity interactions	666
F.5	MAC client transmission requirements	666
F.6	MAC client reception requirements	667
	Annex G (informative) Implementation guidelines.....	669
G.1	Sizing of secondary transit queue and <i>addRateAI</i>	669
G.1.1	Calculation of minimum size for secondary transit queue	669
G.1.2	Calculation of maximum <i>addRateAI</i>	670
G.2	ClassA shaping effects on jitter	670

G.2.1 ClassA shaper characteristics	670
G.2.2 ClassA shaper behaviors.....	671
G.2.3 ClassA clamped-credit shaper behaviors.....	671
Annex H (informative) C-code illustrations of CRC computations	673
Annex I (informative) Datapath scenarios	688
I.1 Duplicate frame scenarios	688
I.1.1 Unidirectional source bypass.....	688
I.1.2 Unidirectional wrapped source bypass.....	689
I.1.3 Bidirectional destination bypass.....	689
I.1.4 Bidirectional destination removals	690
I.1.5 Source and destination removals	690
I.2 Reordered frame scenarios.....	691
I.2.1 Protection switch during bidirectional flood	691
I.2.2 Cascading failures during bidirectional flood	692
I.2.3 Protection switch during unicast transmission on steering system	693
I.2.4 Cascading protection switch during unidirectional flood, wrapping.....	694
Annex J (informative) Spatial indications and shaping.....	695
J.1 Overview	695
J.2 Spatial bandwidth allocation	696
J.2.1 Single queue spatial allocation	696
J.2.2 Dual queue spatial allocation.....	697
J.2.3 Cumulative ringlet allocation	698
J.3 Spatial client queuing	699
Annex K (informative) Client-based OAM operations using echo and flush	701
K.1 Connectivity monitoring using echo request/response	701
K.1.1 Background.....	701
K.1.2 Scope of operations	701
K.1.3 Connectivity monitor.....	701
K.1.4 Failure declaration and clearing	702
K.2 Resteering using flush	703
K.2.1 Background.....	703
K.2.2 Approach	703
K.2.3 Using echo frames to flush steer protected rings.....	704
Index	705

List of tables

Table 3.1—State table notation example	14
Table 3.2—Called state table notation example	15
Table 3.3—Special symbols and operators.....	16
Table 3.4—Names of fields and subfields	17
Table 3.5— <i>wrap</i> field values.....	18
Table 5.1—Service classes and their quality-of-service relationships	31
Table 6.1—service_class values	54
Table 6.2—ringlet_id values.....	54
Table 6.3—flooding_form values	55
Table 6.4—Source address value combinations	56
Table 6.5—reception_status values	57
Table 6.6—Control request opcodes.....	59
Table 6.7—Control indication opcodes	60
Table 6.8—Compliance test point types.....	61
Table 7.1—Service classes	84
Table 7.2—Service mapping	88
Table 7.3—Center wrap adjustments.....	91
Table 7.5— <i>myEdgeState</i> setting	92
Table 7.4—Edge wrap adjustments	92
Table 7.6—Send indication summary.....	94
Table 7.7—IdleShaper state table	97
Table 7.8—MacControlShaper state table	98
Table 7.9—ClassAShaper state table.....	100
Table 7.10—ClassBShaper state table	103
Table 7.11—PreCongestionShaper state table.....	106
Table 7.12—PostCongestionShaper state table	108
Table 7.13—SourceShaper state table	109
Table 7.14—FairnessEligibleIndication state table	111
Table 7.15—DownstreamShaper state table	112
Table 7.16—Strip rules	116
Table 7.17—Client filter selection summary	117
Table 7.18—MAC control filter selection summary	118
Table 7.19—ReceiveFromEdge state table.....	120
Table 7.20—ReceiveCheck state table	124
Table 7.21—ReceiveCount state table.....	129
Table 7.22—ReceiveStrip state table.....	131
Table 7.23—ReceiveAdjust state table	135
Table 7.24—ReceiveFilter state table	137
Table 7.25—ReceiveFilterDataCount state table	140
Table 7.26—ReceiveFilterControlCount state table	143
Table 7.27—WrongRinglet state table	146
Table 7.28—Flooding rules	151
Table 7.29—RingletSelection state table	154
Table 7.30—Initial <i>ttl</i> field values	162
Table 7.31—StageQueueSelection state table	166
Table 7.32—DataAddCount state table	169
Table 7.33—ControlAddCount state table	171
Table 7.34—SingleQueueTransmit state table	174
Table 7.35—DualQueueTransmit state table	178
Table 7.36—TransmitCount state table	180
Table 7.37—TransmitRoute state table	183
Table 8.1—C2 signal labels for GFP and HDLC-like framing	195

Table 8.3—Flag and control escape sequences for HDLC-like framing	197
Table 8.2—GFP EXI and UPI values for RPR	197
Table 8.4—LAPS frame field values for RPR	198
Table 8.5—SONET/SDH supported path layers and capacities	199
Table 9.1—Data frame size constraints	203
Table 9.2—Control frame size constraints	205
Table 9.4— <i>controlType</i> values	207
Table 9.3—Control frame constraints for <i>extendedControl</i> subfield values	207
Table 9.5—Fairness frame constraints for <i>baseControl</i> subfield values	208
Table 9.7— <i>ri</i> values	210
Table 9.6—Idle frame constraints for <i>baseControl</i> subfield values	210
Table 9.9— <i>sc</i> values	211
Table 9.10—Service class and fairness eligible encodings	211
Table 9.8— <i>ft</i> values	211
Table 9.11—Service class and fairness eligible interpretations	212
Table 9.12— <i>fi</i> values	213
Table 10.1—Fairness operations	230
Table 10.2—The <i>agingInterval</i> as a function of <i>lineRate</i>	232
Table 10.3—Fairness frame type (<i>ffType</i>) values	237
Table 10.4—Fairness differential delay frame <i>baseRingControl</i> subfield values	238
Table 10.5—PerByte	241
Table 10.6—PerAgingInterval	246
Table 10.8—AggressiveRateAdjust	247
Table 10.9—ConservativeRateAdjust	249
Table 10.10—PerAdvertisingInterval	253
Table 10.11—PerReportingInterval	256
Table 10.12—ActiveWeightsComputation	258
Table 10.13—FairnessFrameReceive state table	259
Table 10.14—FddFrameTransmit	262
Table 10.15—FrttComputation	266
Table 10.16—PHY types and associated rates	270
Table 11.1—TP frame header field-value restrictions	292
Table 11.2— <i>psw</i> and <i>pse</i> values	293
Table 11.3—TC frame header field-value restrictions	294
Table 11.4—LRTT request frame header field-value restrictions	295
Table 11.5—LRTT response frame header field-value restrictions	296
Table 11.6—ATD frame header field-value restrictions	297
Table 11.7— <i>type</i> encoding for ATD frame	297
Table 11.8—Mapping of address type values to supported address types	301
Table 11.9—Ring-level topology database: <i>ringInfo</i>	304
Table 11.10—Topology database for local station: <i>myTopoInfo</i>	305
Table 11.11—Topology database for one ringlet: <i>topoEntry[ringlet][hops]</i>	306
Table 11.12—ReceiveMonitor state table	312
Table 11.13—TopologyControl state table	315
Table 11.14—ParseTpFrame state table	319
Table 11.15—ProtectionUpdate state table	326
Table 11.16—TopologyValidation state table	334
Table 11.17—TransmitTpFrame state table	337
Table 11.18—ReceiveTpFrame state table	339
Table 11.19—TransmitTcFrame state table	341
Table 11.20—ReceiveTcFrame state table	343
Table 11.21—SecondaryUpdate state table	349
Table 11.22—TimingLrttFrame state table	353
Table 12.1— <i>sesThreshold</i> example values	364

Table 12.2— <i>responseRinglet</i> values	366
Table 12.3—SAS notify frame header field-value restrictions	368
Table 12.4—type values	369
Table 12.5—PIRC frame header field-value restrictions	370
Table 12.6—OamFrameTransmit state table	377
Table 12.7—OamFrameReceive state table	379
Table 13.1—Event indications.....	389
Table 13.2—Statistics definitions	392
Table 13.3—MAC statistics.....	392
Table 14.2—Example SDB for multicast lookup	402
Table 14.3—Example static unicast management table	402
Table 14.1—Example SDB for unicast lookup	402
Table 14.4—Example static multicast management table	403
Table 14.5—Aging time parameter value.....	404
Table 14.6—purgeScope to purgeValue relationship	405
Table 14.7—Purge invocations.....	406
Table 14.8—SasTransmit state table	411
Table 14.9—SasReceive state table	413
Table 15.1—PIRC pircProv database	423
Table 15.2—PIRC pircInfo database	424
Table 15.3—PIRC pircMateMsg database	424
Table 15.4—PircMateMsgReceive state table.....	425
Table 15.5—PircControl state table.....	427
Table 15.6—PircProtectionUpdate state table	429
Table 15.7—PircMateMsgTransmit state table	432
Table 15.8—PircReceive state table	434
Table 15.9—PircTransmit state table	435
Table B.1—Transmit and receive lane associations	449
Table C.1— <i>signalFail</i> and <i>signalDegrade</i> mapping.....	458
Table C.2—GFP payload header field values.....	462
Table C.3— <i>TMOD[1:0]</i> and corresponding valid data	470
Table C.4— <i>RMOD[1:0]</i> and corresponding valid data	471
Table C.6— <i>TxPrty</i> and <i>TxDATA</i> parity	477
Table C.5— <i>TxSize</i> and corresponding valid data	477
Table C.7— <i>RxPrty</i> and <i>RxDATA</i> parity	478
Table C.8— <i>RxSize</i> and corresponding valid data	478
Table C.9— <i>TSTAT</i> and corresponding FIFO status	483
Table C.10— <i>RSTAT</i> and corresponding FIFO status	484
Table D.1—RPR MIB SAS read-write and read-create attributes	492
Table D.2—Structure of the MIB	493
Table D.3—Specific interface MIB objects.....	495
Table F.1—Transmit rule requirements.....	667
Table F.2—Combined host/bridge receive rule requirements	668
Table J.1—MAC client interface	699

List of figures

Figure 3.1—Service definitions	13
Figure 3.2—Bit numbering and ordering.....	18
Figure 3.3—Byte-sequential field format illustrations	19
Figure 3.4—Multi-byte field illustrations	19
Figure 3.5—Organization-specific <i>controlDataUnit</i> field format.....	20
Figure 3.6—48-bit MAC address format.....	20
Figure 3.8—Ringlet orientation conventions	21
Figure 3.7—Illustration of fairness-frame structure	21
Figure 5.1—RPR service and reference model relationship to the ISO/IEC OSI reference model.....	25
Figure 5.2—Dual-ring structure.....	26
Figure 5.4—Single station view of MAC architecture	27
Figure 5.3—Station structure	27
Figure 5.5—End-to-end view of MAC architecture	28
Figure 5.7—MAC data flows.....	29
Figure 5.6—Secondary address transmissions.....	29
Figure 5.8—MAC service interface.....	30
Figure 5.9—MAC data primitives	31
Figure 5.10—Unicast frame transmissions	32
Figure 5.11—Flooded transmissions	33
Figure 5.12—Cleave point migration	33
Figure 5.13—Basic data frame format.....	34
Figure 5.15—Local-source multicast transmissions	35
Figure 5.14—Local-source/local-destination transmissions	35
Figure 5.16—Local-source unknown-unicast transmission.....	36
Figure 5.18—Remote-source multicast transmissions.....	37
Figure 5.17—Remote-source unicast transmissions	37
Figure 5.19—Bandwidth recycling mechanisms	38
Figure 5.20—ClassA allocation enforcement	39
Figure 5.21—Single queue station S1 allocations	40
Figure 5.22—Unregulated vs. equal-weighted fair access on ringlet0	41
Figure 5.23—Fairness frames	42
Figure 5.24—Transit-queuing options	42
Figure 5.25—Fault response mechanisms	44
Figure 5.27—Closed-ring and open-ring topologies	45
Figure 5.26—Protection topologies	45
Figure 5.29—Flushing previously sourced traffic	47
Figure 5.28—Echo operation, ringlet selection possibilities	47
Figure 6.1—RPR service and reference model relationship to the ISO/IEC OSI reference model.....	49
Figure 6.2—MAC service model	53
Figure 6.3—MAC test points	61
Figure 6.4—Single station view of MAC architecture	62
Figure 6.5—MAC control architecture	63
Figure 6.6—MAC datapath architecture	64
Figure 6.7—MAC reference model, showing internal MAC functions and data paths.....	65
Figure 6.8—MAC reference model continued, showing center wrap data paths	66
Figure 6.9—MAC reference model continued, showing edge wrap data paths	66
Figure 7.1—MAC datapath sublayer relationship to the ISO/IEC OSI reference model	71
Figure 7.2—MAC datapath relationships	72
Figure 7.3—MAC datapaths	87
Figure 7.4—Wrap methods	89
Figure 7.5—Credit adjustments over time	93
Figure 7.6—Protection event example.....	113

Figure 7.7—Use of <i>ps</i> bit in wrapping systems	115
Figure 7.8—Receive subentities	115
Figure 7.9—Filter subentities	116
Figure 7.10—Rx SAS subentities	118
Figure 7.11—Transmit entities	147
Figure 7.12—Cleave point example	159
Figure 7.13—Odd number of stations cleave point example	160
Figure 7.14—Even number of stations cleave point example	160
Figure 7.15—Single queue MAC datapath.....	173
Figure 7.16—Dual queue MAC datapath	176
Figure 8.1—RPR RS and PHY relationship to the ISO/IEC OSI reference model.....	190
Figure 8.2—RPR in HDLC-like framing structure.....	198
Figure 8.3—RPR in LAPS framing structure	198
Figure 9.1—Data frame formats	204
Figure 9.2—Control frame format	206
Figure 9.3—Fairness frame format.....	208
Figure 9.4—Idle frame format	209
Figure 9.5— <i>baseControl</i> field format for data frames	210
Figure 9.6— <i>extendedControl</i> field format for data and control frames	212
Figure 10.1—Fairness function relationship to the ISO/IEC OSI reference model	216
Figure 10.3—Fairness instances on an open ring using steering	217
Figure 10.2—Fairness instances on a closed ring.....	217
Figure 10.4—Fairness instances on an open ring using wrapping	218
Figure 10.5—Path of fairness frames on an open ring	218
Figure 10.6—Congestion control objectives	220
Figure 10.7—The <i>fairRate</i> advertised upstream by a single congested station.....	220
Figure 10.8—Path of a rate advertisement	221
Figure 10.9—Congestion control feedback	221
Figure 10.10—Received <i>fairRate</i> more restrictive than local <i>fairRate</i>	222
Figure 10.11—Local <i>fairRate</i> more restrictive than received <i>fairRate</i>	222
Figure 10.12—Advertising the FULL_RATE	223
Figure 10.13—Congestion domains	224
Figure 10.14—Traffic rate measurements	227
Figure 10.15—Fairness frame payload.....	237
Figure 10.16—Fairness differential delay (FDD) payload	238
Figure 10.17—LRTT associated with path that is not wrapped	263
Figure 10.18—Tail downstream of head on same ringlet.....	264
Figure 10.19—Tail upstream of head on opposing ringlet	264
Figure 10.20—Tail downstream of head on opposing ringlet	265
Figure 10.21—Rate counter asymptotically approaching rate	269
Figure 11.1—Topology and protection entities relationship to the ISO/IEC OSI reference model	274
Figure 11.2—Context containment sequence	278
Figure 11.3—Steering protection of unicast traffic	279
Figure 11.4—Wrapping protection of traffic	279
Figure 11.5—Relationship between <i>ri</i> value and east/west.....	280
Figure 11.6—TP payload	292
Figure 11.7— <i>protStatus</i> field format	292
Figure 11.8— <i>prefs</i> field format	293
Figure 11.10—Alignment of the 32-bit fields used to compute topology checksum	294
Figure 11.9—TC payload	294
Figure 11.12—LRTT request payload	295
Figure 11.11— <i>checksumStatus</i> field format	295
Figure 11.13—LRTT response payload	296
Figure 11.15—Type-length-value format	297

Figure 11.14—Attribute discovery payload.....	297
Figure 11.16—Weight <i>attDataUnit</i> format.....	298
Figure 11.17—Station bandwidth <i>attDataUnit</i> format	299
Figure 11.18—Station settings <i>attDataUnit</i> format	299
Figure 11.19—Station name <i>attDataUnit</i> format	300
Figure 11.20—Management address ATT <i>attDataUnit</i> format.....	300
Figure 11.21—Station interface index ATT <i>attDataUnit</i> format	301
Figure 11.23—Organization-specific <i>attDataUnit</i> format.....	302
Figure 11.22—Secondary MAC ATT <i>attDataUnit</i> format.....	302
Figure 11.25—Topology database structure	303
Figure 11.24—Mapping of organization-specific values.....	303
Figure 11.26—Topology database parameters	308
Figure 11.27—Topology change sequences	309
Figure 11.28—Topology and protection relationships	310
Figure 11.29—Protection topologies	322
Figure 11.30—Edge condition updates.....	323
Figure 11.31—Duplicate MAC address scenarios.....	330
Figure 12.1—OAM control entity relationship to the ISO/IEC OSI reference model	360
Figure 12.2—Echo request/response operations.....	362
Figure 12.3—Flushing on an unprotected and wrapped rings	362
Figure 12.4—Echo request/response frame payload format.....	365
Figure 12.5— <i>responseControl</i> field format.....	366
Figure 12.7—Organization-specific frame payload format.....	367
Figure 12.8—Organization-specific <i>controlDataUnit</i> field format	367
Figure 12.6—Flush frame payload format.....	367
Figure 12.9—Mapping of organization-specific values.....	368
Figure 12.10—SAS notify frame payload format.....	368
Figure 12.11—Identifier value format	369
Figure 12.12—Identifier bit map format.....	369
Figure 12.13—PIRC frame payload format.....	370
Figure 12.14—PIRC frame status field.....	370
Figure 12.15—Available second classification.....	381
Figure 13.1—Management entities relationship to the ISO/IEC OSI reference model.....	386
Figure 13.2—Counter-measurement points	391
Figure 14.1—Undirected transmission	396
Figure 14.2—Directed transmission	396
Figure 14.3—SAS directed transmission	408
Figure 14.4—SAS undirected transmission	408
Figure 15.1—PIRC station types	418
Figure B.1—Reconciliation sublayer relationship to the ISO/IEC OSI reference model.....	440
Figure B.2—PRS-1 inputs and outputs	441
Figure B.3—PRS-1 signal mapping.....	442
Figure B.4—PRS-10 inputs and outputs	447
Figure B.5—PRS-10 signal mapping.....	448
Figure C.1—Reconciliation sublayer relationship to the ISO/IEC OSI reference model.....	457
Figure C.2—OSI and OIF reference model layers.....	457
Figure C.4—GRS frame structure.....	460
Figure C.3—SRS frame structure	460
Figure C.5—GFP core header.....	461
Figure C.6—GFP payload header	461
Figure C.7—SRS/GRS inputs and outputs using SPI-3(8).....	463
Figure C.8—8-bit SPI-3 transmit signal mapping	465
Figure C.9—SRS/GRS inputs and outputs using 32-bit SPI-3	469
Figure C.10—32-bit SPI-3 signal mappings	472

Figure C.11—SPI-4.1 single 64-bit interface signals	476
Figure C.12—64-bit SPI-4.1 signal mappings.....	479
Figure C.13—SPI-4.2 interface signals	482
Figure C.14—SPI-4.2 signal mappings	484
Figure E.1—Serial Crc16 reference model.....	643
Figure E.2—CRC16 calculations.....	644
Figure E.3—Protected header-field changes	644
Figure E.4—Illustration of CRC16 checks	645
Figure E.5—Serial CRC32 reference model	646
Figure E.6—Exchanged EXORSum calculations.....	647
Figure E.8—Illustration of CRC32 checks	648
Figure E.7—Data CRC stomping	648
Figure E.9—Parity generation model	649
Figure E.10—Illustration of fairness-frame checks.....	649
Figure F.1—Bridge architecture reference	652
Figure F.2—Loop-free and looped broadcast networks	653
Figure F.3—Local/remote end station transmission scenarios	654
Figure F.4—Asymmetric traffic flow issue	655
Figure F.6—Incorrect bridge filtering behavior	657
Figure F.5—RPR MAC handling of asymmetric traffic flows.....	657
Figure F.7—Correct bridge filtering behavior	659
Figure F.8—Bridge architecture model	661
Figure F.9—Service primitive mapping for frames from a bridge LLC or host	663
Figure F.10—Service primitive mapping for non-LLC frames from a bridge	664
Figure F.11—Enhanced service primitive mapping for frames from a bridge LLC or host	665
Figure F.12—Enhanced service primitive mapping for non-LLC frames from a bridge	666
Figure G.1—Provisioned classA bandwidth scenario	670
Figure G.2—ClassA credits fluctuations	671
Figure G.3—ClassA clamped-credit shaper behaviors.....	672
Figure I.1—Duplicate scenario 1: Unidirectional source bypass	688
Figure I.2—Duplicate scenario 2: Unidirectional wrapped source bypass.....	689
Figure I.3—Duplicate scenario 3: Bidirectional destination bypass	689
Figure I.4—Duplicate scenario 4: Bidirectional destination removals.....	690
Figure I.5—Duplicate scenario 5: Source and destination removals.....	690
Figure I.6—Reorder scenario 1: Protection switch during bidirectional flood	691
Figure I.7—Reorder scenario 2: Cascading failures during bidirectional flood.....	692
Figure I.8—Reorder scenario 3: Protection switching during unicast transmission	693
Figure I.9—Reorder scenario 4: Cascading failures during unidirectional flood.....	694
Figure J.1—Single queue spatial allocation.....	696
Figure J.2—Dual queue spatial allocation	697
Figure J.3—Cumulative station allocations	698
Figure K.1—Flushing a steered ring.....	704

**IEEE Standard for Information technology—
Telecommunications and information exchange between systems
Local and metropolitan area networks—
Specific requirements**

Part 17: Resilient packet ring (RPR) access method and physical layer specifications

IMPORTANT NOTICE: This standard is not intended to ensure safety, security, health, or environmental protection. Implementers of the standard are responsible for determining appropriate safety, security, environmental, and health practices or regulatory requirements.

This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Documents.” They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/IPR/disclaimers.html>.

1. Overview

Resilient packet ring (RPR) is a metropolitan area network (MAN) technology supporting data transfer among stations interconnected in a dual-ring configuration.

1.1 Scope

This standard defines a resilient packet ring access protocol for use in local, metropolitan, and wide area networks, along with appropriate physical layer specifications for transfer of data packets at rates scalable to multiple gigabits per second.

1.2 Purpose

The standard defines a very high-speed network protocol that is optimized for packet transmission in resilient ring topologies needing service quality differentiation, efficient use of bandwidth, fair use of bandwidth, ease of use, and robustness.

1.3 RPR features

Key features of RPR that can distinguish it from other network interconnects, include the following:

- a) Addressing. Unicast, multicast, and simple broadcast data transfers are supported.
- b) Services. Multiple service qualities are supported, as listed below.
 - Per-service-quality flow-control protocols regulate traffic introduced by clients.
 - 1) ClassA. The allocated/guaranteed bandwidth has low circumference-independent jitter.
 - 2) ClassB. The allocated/guaranteed bandwidth has bounded circumference-dependent jitter.
 - Allows for transmissions of excess information rate (EIR) bandwidths (with classC properties).
 - 3) ClassC. Provides best-effort services.
 - c) Efficiency. Design strategies increase effective bandwidths beyond those of a broadcast ring.
 - 1) Concurrent transmission. Clockwise and counterclockwise transmissions can be concurrent.
 - 2) Bandwidth reallocation. Bandwidths can be reallocated on nonoverlapping segments.
 - 3) Bandwidth reclamation. Unused bandwidths can be reclaimed by opportunistic services.
 - 4) Spatial bandwidth reuse. Opportunistic bandwidths are reused on nonoverlapping segments.
 - 5) Temporal bandwidth reuse. Unused opportunistic bandwidth can be consumed by others.
 - d) Fairness. Fairness ensures proper partitioning of opportunistic traffic.
 - 1) Weighted. Weighted fair access to available ring capacity.
 - 2) Simple. Point-of-congestion flow-control facilitates per-destination queuing in client.
 - 3) Detailed. The (optional) multi-choke fairness allows the client to selectively throttle its transmissions based on multiple congestion point indications.
 - e) Plug-and-play. Automatic topology discovery and advertisement of station capabilities allow systems to become operational without manual intervention.
 - f) Robustness. Multiple features support robust frame transmissions.
 - 1) Responsive. Service restoration time is less than 50 ms after a station or link failure.
 - 2) Lossless. Queue and shaper specifications avoid frame loss in normal operation.
 - 3) Tolerant. Fully distributed control architecture eliminates single points of failure.
 - 4) OAM. Operations, administration, and maintenance support service provider environments.

RPR supports up to 255 station attachments and is optimized for rings with a maximum circumference of 2000 km.

NOTE—The 2000 km figure previously mentioned is a design goal and not a specific physical constraint.¹

1.4 Document structure

This document is Part 17 of IEEE Std 802[®],² providing the RPR medium access control (MAC) and physical layer (PHY) specifications. All of the following material is included within the scope of this standard except those items explicitly identified as informative:

- a) Clause 1: Overview
- b) Clause 2: Normative references (provisions of this standard through reference in the text)
- c) Clause 3: Terms, definitions, and notation
- d) Clause 4: Abbreviations and acronyms
- e) Clause 5: Architecture overview

¹Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

²Information on references can be found in Clause 2.

- f) Clause 6: Medium access control (MAC) service and reference model
- g) Clause 7: Medium access control datapath
- h) Clause 8: MAC physical interface
- i) Clause 9: Frame formats
- j) Clause 10: Fairness
- k) Clause 11: Topology discovery and protection
- l) Clause 12: Operations, administration, and maintenance (OAM)
- m) Clause 13: Layer management entity interface
- n) Clause 14: Spatially aware sublayer
- o) Clause 15: Protected inter-ring connection
- p) Annex A: (informative): Bibliography
- q) Annex B: (normative): PacketPHY reconciliation sublayers
- r) Annex C: (normative): SONET/SDH reconciliation sublayers
- s) Annex D: (normative): SNMP MIB definitions
- t) Annex E: (normative): CRC and parity calculations
- u) Annex F: (informative): 802.1D and 802.1Q bridging conformance
- v) Annex G: (informative): Implementation guidelines
- w) Annex H: (informative): C-code illustrations of CRC computations
- x) Annex I: (informative): Datapath scenarios
- y) Annex J: (informative): Spatial indications and shaping
- z) Annex K: (informative): Client-based OAM operations using echo and flush

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used; therefore, each referenced document is cited in text, and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

ANSI T1.105.01-2000, Synchronous Optical Network (SONET)—Automatic Protection Switching.³

EIA/JEDEC JESD8-B, Interface Standard for Nominal 3V/3.3V Supply Digital Integrated Circuits.⁴

IEEE Std 802-2001, IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture.^{5, 6}

IEEE Std 802.1DTM-2004, Information Technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Part 3: Media Access Control (MAC) Bridges.

IEEE Std 802.1QTM-2005, IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks.

IEEE Std 802.3TM-2008, Information Technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.

IETF RFC 1662: PPP in HDLC-Like Framing, W. Simpson, July 1994.⁷

IETF RFC 2578, STD 58: Structure of Management Information Version 2 (SMIV2), K. McCloghrie et al., April 1999.

IETF RFC 2579, STD 58: Textual Conventions for SMIV2, K. McCloghrie et al., April 1999.

IETF RFC 2580, STD 58: Conformance Statements for SMIV2, K. McCloghrie et al., April 1999.

IETF RFC 2615: PPP over SONET/SDH, A. Malis, W. Simpson, June 1999.

IETF RFC 2863: The Interfaces Group MIB, K. McCloghrie, F. Kastenholz, June 2000.

IETF RFC 3291: Textual Conventions for Internet Network Addresses, M. Daniele et al., May 2002.

ISO/IEC 2382-9:1995, Information technology—Vocabulary—Part 9: Data communication.⁸

³ANSI publications are available from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

⁴EIA publications are available from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (<http://global.ihs.com/>).

⁵The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

⁶IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854-4141, USA (<http://standards.ieee.org/>).

⁷IETF publications are available via the World Wide Web at <http://www.ietf.org>.

⁸ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iso.ch/>). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (<http://global.ihs.com/>). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

ISO/IEC 9899:1999, Programming languages—C.

ITU-T Recommendation G.707, Network Node Interface for the Synchronous Digital Hierarchy (SDH).⁹

ITU-T Recommendation G.783, Characteristics of Synchronous Digital Hierarchy (SDH) Equipment Functional Blocks.

ITU-T Recommendation G.7041, Generic Framing Procedure.

ITU-T Recommendation X.85/Y.1321, IP over SDH using LAPS.

OIF System Packet Interface Level 3 (SPI-3): OC48 System Interface for Physical and Link Layer Devices. Implementation agreement: OIF-SPI3-01.0, June 2000.¹⁰

OIF System Packet Interface Level 4 Phase 1 (SPI-4.1): OC192 System Interface for Physical and Link Layer Devices. Implementation agreement: OIF-SPI4-01.0, April 2001.

OIF System Packet Interface Level 4 Phase 2 (SPI-4.2): OC192 System Interface for Physical and Link Layer Devices. Implementation agreement: OIF-SPI4-02.0, January 2001.

Telcordia Technologies GR-253-CORE, Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria (A Module of TSGR, FR-440). Issue 3. September 2000.

⁹ITU-T publications are available from the International Telecommunications Union, Place des Nations, CH-1211, Geneva 20, Switzerland/Suisse (<http://www.itu.int/>).

¹⁰Optical Internetworking Forum (OIF) publications are available via the World Wide Web at <http://www.oiforum.com>, under “Technical Work” and “OIF Implementation Agreements”.

3. Terms, definitions, and notation

3.1 Conformance levels

Several key words are used to differentiate between different levels of requirements and options, as described in this subclause.

3.1.1 may: Indicates a course of action permissible within the limits of the standard with no implied preference (“may” means “is permitted to”).

3.1.2 shall: Indicates mandatory requirements to be strictly followed in order to conform to the standard and from which no deviation is permitted (“shall” means “is required to”).

3.1.3 should: An indication that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (“should” means “is recommended to”).

3.2 Terms and definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary: Glossary of Terms & Definitions* should be consulted for terms not defined in this clause.¹¹

3.2.1 adaptation sublayer: A protocol sublayer that exists for the purpose of converting data from one format to another.

3.2.2 agent: A network management entity (NME) used to configure a station and/or collect data describing the operation of that station.

3.2.3 aggressive rate adjustment: A method of rate adjustment in which the rates can be changed without significant delay and amounts of rate change are not highly damped.

3.2.4 allocated bandwidth: Bandwidth that may be subject to reclamation, but that is to be made available to the allocating station when requested. A class of traffic is said to be allocated when the limits associated with that class are determined by a network operator's engineering/operations department. There are two types of allocated bandwidth: reserved and reclaimable. Allocated bandwidth refers to the amount of bandwidth on a ringlet that provides the necessary capacity for classes of committed rate services.

3.2.5 available capacity: Ring capacity not required to support allocated service and, consequently, available to support opportunistic service.

3.2.6 best-effort: Not associated with an explicit service guarantee.

3.2.7 bidirectional flooding: A frame forwarding transfer involving sending two flooding frames, one on each ringlet (ringlet0 and ringlet1), where each frame is directed to distinct adjacent stations.

3.2.8 bit error ratio (BER): The ratio of the number of bits received in error to the total number of bits transmitted.

¹¹The *IEEE Standards Dictionary: Glossary of Terms & Definitions* is available at <http://shop.ieee.org/>.

3.2.9 bridge: A functional unit interconnecting two or more networks at the data link layer of the OSI reference model.

3.2.10 broadcast: The act of sending a frame addressed to all stations on the network.

3.2.11 broadcast address: A group address that consists of all-ones and identifies all of the stations in a network.

3.2.12 cleave point: The point, between stations, at which a ring is logically divided.

3.2.13 closed ring: An intact ring (i.e., one that has not been cut) that provides a complete path all the way around the ring. A closed ring contains no detected edges.

3.2.14 committed information rate (CIR): The rate, averaged over a minimum interval of time, at which the network agrees to transfer data.

3.2.15 congestion: The state of a ringlet or station determined by calculation of the MAC fairness algorithm that causes restriction or regulation of the addition of fairness eligible traffic to the ring.

3.2.16 congestion domain: The set of contiguous links associated with a ringlet that causes congestion at a common congestion point. This is also the set of links between the congestion head and the congestion tail.

3.2.17 congestion head: The furthest downstream station of a congestion domain. This is the station immediately upstream of the congestion point.

3.2.18 congestion point: The transmit link of a station experiencing congestion and not contributing to downstream congestion. This link is immediately downstream of the congestion head.

3.2.19 congestion tail: The furthest upstream station of a congestion domain.

3.2.20 conservative: An algorithm that tends to exhibit an overdamped transient response.

3.2.21 conservative rate adjustment: A method of rate adjustment in which the rate is generally not changed until sufficient time has passed for the effect of any previous rate adjustment to be observed and whose amount of change exhibits significant dampening and hysteresis.

3.2.22 cyclic redundancy check (CRC): A specific type of frame check sequence computed using a generator polynomial.

3.2.23 destination station: A station to which a frame is addressed.

3.2.24 destination stripping: The removal of frames from the ring by the destination station.

3.2.25 directed transmission: Transmission of a frame to a single station on the medium.

3.2.26 downstream: The direction of data flow on a ringlet.

3.2.27 dual-ring: A ring composed of two ringlets sharing the same set of stations such that the order of station traversal for data frames on one ringlet is exactly opposite that of the other ringlet when no faults exist on the ring.

3.2.28 end-to-end delay: The time required for the transfer of a frame between source and destination stations as measured from the start of frame transmission to the start of frame reception at the respective MAC service interfaces.

3.2.29 excess information rate: The rate at which the network transfers data in excess of the committed information rate when no network congestion exists.

3.2.30 fairness: The property that for any given link on the ring, every source receives an equal proportion of fairness eligible capacity. If all sources have equal weights, then all sources have substantially equal access to the available capacity of all links.

3.2.31 fairness eligible: A quality of a frame that indicates if it is subject to the fairness algorithm.

3.2.32 fairness round-trip time (FRTT): The time that it takes for a fairness value to propagate from the head of a congestion domain to the tail of the same congestion domain and for the first affected data frame to be sent from the tail of the congestion domain to be received by the head of the congestion domain.

3.2.33 fair rate: A rate having the property that one station is prevented from utilizing a disproportionate share of available ringlet capacity with respect to other stations on the ringlet.

3.2.34 flooding: The act of transmitting a frame such that all stations on the ring receive the frame once.

3.2.35 flooding scope: The number of hops that a frame can travel (around the ring) from a given source station to a destination station associated with a given ringlet.

3.2.36 flow control: A method of congestion control in which a station communicates to another station or stations, information intended to regulate the transmission of frames.

3.2.37 frame: The MAC sublayer protocol data unit (PDU).

3.2.38 frame check sequence (FCS): The field immediately preceding the closing delimiter of a frame, allowing the detection of payload errors by the receiving station.

3.2.39 gigabit media independent interface (GMII): The interface between the reconciliation sublayer and the physical coding sublayer (PCS) for 1000 Mb/s operation.

3.2.40 group address: An address that identifies a group of stations on a network.

3.2.41 header error check (HEC): A field appended to the frame header for the purpose of detecting, and optionally correcting, errors in the transmission of the header portion of the frame.

3.2.42 hop-count: The number of hops traversed by data circulating between stations on the ring.

3.2.43 in flight: Transmitted by the source MAC and not yet received by the (final) destination MAC.

3.2.44 in transit: Received by a MAC's PHY and not yet transmitted by the MAC's other PHY.

3.2.45 individual address: An address that identifies a particular station on a network.

3.2.46 jitter: The variation in delay associated with the transfer of frames between two points.

3.2.47 keepalive: An exchange of messages allowing verification that communication between stations is active.

3.2.48 layer management entity (LME): The entity within a protocol layer that performs local management of that layer. This entity provides information about the layer, effects control over it, and indicates the occurrence of certain events within it.

NOTE—The term *LME* can be prefixed with the name of the layer, for example, MAC LME (MLME).

3.2.49 link: A unidirectional channel connecting adjacent stations on a ringlet.

3.2.50 link rate: The data rate with which a MAC communicates with its adjacent MAC entity.

3.2.51 line rate: The rate at which a PHY transfers data on the attached physical medium.

3.2.52 local area network (LAN): A communications network designed for a small geographic area, typically not exceeding a few kilometers in extent, and characterized by moderate to high data transmission rates, low delay, and low bit error rates.

3.2.53 logical link control (LLC) sublayer: That part of the data link layer that supports media independent data link functions, and uses the services of the MAC sublayer to provide services to the network layer.

3.2.54 loop round-trip time (LRTT): The time that it takes for a control frame or a data frame to be sent from one station to another station and back to the original station.

3.2.55 MAC client: The layer entity that invokes the MAC service interface.

3.2.56 MAC layer management entity (MLME): *See: LME.*

3.2.57 major priority inversion: Any reclamation-caused effect that violates the guarantee of any equal or higher class.

3.2.58 management information base (MIB): A repository of information to describe the operation of a specific network device.

3.2.59 maximum transfer unit (MTU): The largest frame (comprising payload and all header and trailer information) that can be transferred across the network.

3.2.60 medium (plural: media): The material on which information signals are carried, e.g., optical fiber, coaxial cable, and twisted-wire pairs.

3.2.61 medium access control (MAC) sublayer: The portion of the data link layer that controls and mediates the access to the network medium.

NOTE—In this standard, the MAC sublayer comprises the MAC datapath sublayer and the MAC control sublayer.

3.2.62 metropolitan area network (MAN): A network interconnecting individual stations and/or LANs, and spanning an area typically occupied by a city.

NOTE—A MAN generally operates at a higher speed than the networks interconnected and crosses network administrative boundaries.

3.2.63 minor priority inversion: A small reclamation-caused effect on any equal or higher service class that does not violate the service guarantees of that class.

3.2.64 multicast: Transmission of a frame to stations specified by a group address.

3.2.65 multicast address: A group address that is not a broadcast address, i.e., is not all-ones, and identifies some subset of stations on the network.

3.2.66 multi-choke: The characteristic of an algorithm, related to the visibility of multiple points of congestion on a ringlet.

3.2.67 network: A set of communicating stations and the media and equipment providing connectivity among the stations.

3.2.68 non-revertive mode: A mode of operation of a station in which, upon clearing of all failures on one of its spans, the station remains in a wait to restore protection state until the wait to restore condition is preempted by a higher priority protection condition elsewhere on the ring.

3.2.69 open ring: A ring that has been cut, thereby preventing it from completing a full loop. It has at least one detected edge.

3.2.70 opposing ringlet: A ringlet whose traffic circulates in the direction opposite that of a given ringlet.

3.2.71 packet: A generic term for a PDU associated with a layer-entity above the MAC sublayer.

3.2.72 passthrough: A condition of a MAC such that frames pass through it as if it were operating as a link between the stations on either side of it.

3.2.73 permissive: Frame transmission that permits (within a conversation) frame reorder and duplication while recovering from failures on the ring, and frame reorder when changing between undirected and directed transmission.

3.2.74 physical layer (PHY): The layer responsible for interfacing with the transmission medium. This includes conditioning signals received from the MAC for transmitting to the medium and processing signals received from the medium for sending to the MAC.

3.2.75 plug-and-play: The requirement that a station perform transit, strip, and ring control activities without operator intervention (except for any intervention needed for connection to the ring). The station can additionally copy and insert frames.

3.2.76 port: A generic term identifying a point of access to a device or service.

3.2.77 primary protection group member: The PIRC protection group member with higher protection priority.

3.2.78 protected inter-ring connection (PIRC) mate station: The other station on the same ring performing the interconnect functionality from that ring to the same destination ring(s).

3.2.79 protected inter-ring connection (PIRC) station: A station that provides connectivity between rings and exchanges PIRC messages with other similarly interconnected devices.

3.2.80 protection group: A collection of two PIRC stations on the same ring performing the interconnect functionality from that ring to the same destination rings.

3.2.81 protection group member: One of the PIRC stations in a protection group.

3.2.82 protection switch event: A received protection control frame that causes the local topology and status database to be updated/changed.

3.2.83 protocol data unit (PDU): Information delivered as a unit between peer layer entities that contains control information and, optionally, data.

3.2.84 protocol implementation conformance statement (PICS): A statement of which capabilities and options have been implemented for a given Open Systems Interconnection (OSI) protocol.

3.2.85 rate: The number of bytes transferred per time.

NOTE—In this standard, rate is measured in bytes per *ageCoef* agingIntervals when nothing else is stated explicitly.

3.2.86 reconciliation sublayer (RS): A sublayer that provides a mapping between the PHY service interface and the medium independent interface of the PHY.

3.2.87 relaxed: Frame transmission that permits frame reorder and duplication within a conversation when recovering from failures on the ring.

3.2.88 remote address: Any unicast address that is not a ring local address.

3.2.89 replicated frame: A frame that is copied for sending bidirectionally on both ringlets.

3.2.90 reserved bandwidth: The amount of bandwidth that is to be kept available (i.e., is not subject to reclamation). This represents the subset of allocated bandwidth that is not dynamically reclaimable by a fairness algorithm.

3.2.91 reclaimable bandwidth: That subset of allocated bandwidth that is dynamically reclaimable by a fairness algorithm.

3.2.92 revertive mode: A mode of operation of a station in which, upon clearing of all failures on one of its spans, the station enters a wait to restore protection state, followed by an entry to the idle state after expiration of the wait to restore time.

3.2.93 ring: The collection of stations and links forming a resilient packet ring.

3.2.94 ring local address: The station address of one station on the ring.

3.2.95 ring round-trip time (RRTT): The time that it takes for a control frame or a data frame to be sent around an entire ring.

3.2.96 ringlet: A closed unidirectional path formed by an ordered set of stations and interconnecting links, such that each station has exactly one link entering the station and one link exiting the station.

3.2.97 scoped transmission: A method of transmitting a multicast frame with a TTL value that limits its reach to the furthest station that requires the frame.

3.2.98 secondary protection group member: The PIRC protection group member with lower protection priority.

3.2.99 service class¹²: The categorization of MAC service by delay bound, relative priority, data rate guarantee, or similar distinguishing characteristics.

3.2.100 service data unit (SDU): Information delivered as a unit between adjacent layer entities, possibly also containing a PDU of the upper layer.

¹²*Service class* and *traffic class* are synonyms for the purposes of this standard. *Service class* is the preferred term.

3.2.101 service guarantee: Delay or jitter bounds or bandwidth guaranteed for an instance of a service class.

3.2.102 shaper: A device that converts an arbitrary traffic flow to a smoothed traffic flow at a specified data rate.

3.2.103 single-choke: The characteristic of an algorithm, related to the visibility of only a single point of congestion on a ringlet. A MAC that deploys this type of fairness algorithm at each station has visibility of no more than one point of congestion on a ringlet.

3.2.104 source station: The station that originates a frame with respect to the ring.

3.2.105 source stripping: The removal of frames by the source station after circulation on the ring.

3.2.106 spatial reuse: The concurrent transfer of independent traffic on nonoverlapping portions of a ringlet.

3.2.107 station: A device attached to a network for the purpose of transmitting and receiving information on that network.

3.2.108 strict: Frame transmission that prevents frame reorder and duplication within a conversation.

3.2.109 strict data frame: A frame having its *so* (strict order) bit set.

3.2.110 topology: The arrangement of links and stations forming a network, together with information on station attributes.

3.2.111 traffic class: *See: service class.*

3.2.112 transfer: The movement of an SDU from one layer to an adjacent layer. Also used generally to refer to any movement of information from one point to another in the network.

3.2.113 transit delay: The delay experienced by a frame transiting a station on the ringlet.

3.2.114 transit queue: A queue maintained for the purpose of storing a transit frame at a station until that frame has permission to continue transit of the ringlet.

3.2.115 transmit (transmission): The action of a station placing a frame on the medium.

3.2.116 transparent bridging: A bridging mechanism that is transparent to the end stations.

3.2.117 unicast: The act of sending a frame addressed to a single station.

3.2.118 undirected transmission: Transmission of a frame on to the medium using flooding.

3.2.119 unidirectional flooding: A frame forwarding transfer involving sending a flooding frame in the downstream direction of one ringlet, with that frame being directed to its sending station.

3.2.120 unknown unicast: A unicast frame for which the location of its destination is unknown.

3.2.121 unallocated bandwidth: Bandwidth that is not allocated to any provisioned service.

3.2.122 unreserved: A designation for traffic capacity that is not reserved. This is also a designation for the traffic occupying that bandwidth. In addition, unreserved bandwidth may or may not be allocated bandwidth.

3.2.123 upper layers: The collection of protocol layers above the data-link layer.

3.2.124 virtual LAN (VLAN): A subset of the active topology of a bridged local area network.

3.2.125 weighted fairness: A class of fairness algorithm that allows the assignment of unequal shares of available ring capacity. For the purposes of this standard, any reference to fairness is to be taken to imply weighted fairness. An alternative definition is the use of a weight associated with each fairness instance, allowing a station to add at a higher or lower rate than other stations without violating fairness objectives. This behavior can be effectively disabled by assigning equal weights to fairness instances.

3.2.126 weighted fair share: The ratio of the weight of a station to the sum of the weights of stations active on the ringlet.

3.3 Service definition method and notation

The service of a layer or sublayer is the set of capabilities that it offers to a user in the next higher (sub)layer. Abstract services are specified in this standard by describing the service primitives and parameters that characterize each service. This definition of service is independent of any particular implementation (see Figure 3.1).

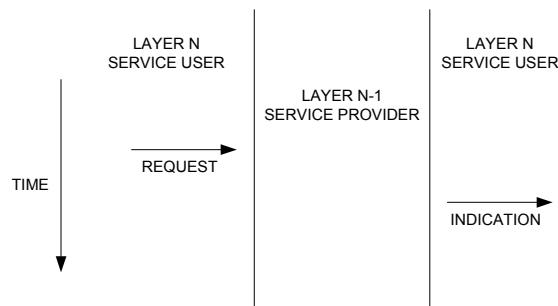


Figure 3.1—Service definitions

Specific implementations can also include provisions for interface interactions that have no direct end-to-end effects. Examples of such local interactions include interface flow control, status requests and indications, error notifications, and layer management. Specific implementation details are omitted from this service specification, because they differ from implementation to implementation and because they do not impact the peer-to-peer protocols.

3.3.1 Classification of service primitives

Primitives are of two generic types:

- REQUEST. The request primitive is passed from layer N to layer N-1 to request that a service be initiated.
- INDICATION. The indication primitive is passed from layer N-1 to layer N to indicate an internal layer N-1 event that is significant to layer N. This event can be logically related to a remote service request, or it can be caused by an event internal to layer N-1.

The service primitives are an abstraction of the functional specification and the user-layer interaction. The abstract definition does not contain local details of the user-provider interaction. For instance, it does not

indicate the local mechanism that allows a user to indicate that it is awaiting an incoming call. Each primitive has a set of zero or more parameters, representing data elements that are passed to qualify the functions invoked by the primitive. Parameters indicate information available in a user-provider interaction. In any particular interface, some parameters can be explicitly stated (even though not explicitly defined in the primitive) or implicitly associated with the service access point. Similarly, in any particular protocol specification, functions corresponding to a service primitive can be explicitly defined or implicitly available.

3.4 State machines

The operation of a protocol can be described by subdividing the protocol into a number of interrelated functions. The operation of the functions can be described by state machines. Each state machine represents the domain of a function and consists of a group of connected, mutually exclusive states. Only one state of a function is active at any given time. A transition from one state to another is assumed to take place in zero time (i.e., no time period is associated with the execution of a state), based on some condition of the inputs to the state machine.

The state machines contain the authoritative specification of the functions they depict. When apparent conflicts between descriptive text and state machines arise, the order of precedence shall be formal state tables first, followed by the descriptive text, over any explanatory figures. This does not override, however, any explicit description in the text that has no parallel in the state tables.

The models presented by state machines are intended as the primary specifications of the functions to be provided. It is important to distinguish, however, between a model and a real implementation. The models are optimized for simplicity and clarity of presentation, whereas any realistic implementation might place heavier emphasis on efficiency and suitability to a particular implementation technology. It is the functional behavior of any unit that has to match the standard, not its internal structure. The internal details of the model are useful only to the extent that they specify the external behavior clearly and precisely.

3.4.1 State table notation

3.4.1.1 Continuously executed state tables

State machines may be represented in tabular form. The table is organized into two columns: a left-hand side representing all of the possible states of the state machine and all of the possible conditions that cause transitions out of each state, and the right-hand side giving all of the permissible next states of the state machine as well as all of the actions to be performed in the various states, as illustrated in Table 3.1. The syntax of the expressions follows standard C notation (see 3.14). No time period is associated with the transition from one state to the next.

Table 3.1—State table notation example

Current state		Row	Next state	
state	condition		action	state
START	sizeOfMacControl > spaceInPTQ	1	—	START
	passM == 0	2		
	—	3	TransmitFromControlQueue();	FINAL
FINAL	SelectedTransferCompletes()	4	—	START
	—	5	—	FINAL

Row 3.1-1: Do nothing if the size of the queued MAC control frame is larger than the PTQ space.

Row 3.1-2: Do nothing in the absence of MAC control transmission credits.

Row 3.1-3: Otherwise, transmit a MAC control frame.

Row 3.1-4: When the transmission completes, start over from the initial state (i.e., START).

Row 3.1-5: Until the transmission completes, remain in this state.

Each combination of current state, next state, and transition condition linking the two is assigned to a different row of the table. Each row of the table, read left to right, provides the name of the current state; a condition causing a transition out of the current state; an action to perform (if the condition is satisfied); and, finally, the next state to which the state machine transitions, but only if the condition is satisfied. The symbol “—” signifies the default condition (i.e., operative when no other condition is active) when placed in the condition column, and it signifies that no action is to be performed when placed in the action column. Conditions are evaluated in order, top to bottom, and the first condition that evaluates to a result of TRUE is used to determine the transition to the next state. If no condition evaluates to a result of TRUE, then the state machine remains in the current state. The starting or initialization state of a state machine is always labeled “START” in the table (although it need not be the first state in the table). Every state table has such a labeled state.

Each row of the table is preferably provided with a brief description of the condition and/or action for that row. The descriptions are placed after the table, and linked back to the rows of the table using numeric tags.

3.4.1.2 Called state tables

A RETURN state is the terminal state of a state machine that is intended to be invoked by another state machine, as illustrated in Table 3.2. Once the RETURN state is reached, the state machine terminates execution, effectively ceasing to exist until the next invocation by the caller, at which point it begins execution again from the START state. State machines that contain a RETURN state are considered to be instantiated only when they are invoked. They do not have any persistent (static) variables.

Table 3.2—Called state table notation example

Current state		Row	Next state	
state	condition		action	state
START	sizeOfMacControl > spaceInPTQ	1	—	FINAL
	passM == 0	2		
	—	3	TransmitFromControlQueue();	RETURN
FINAL	MacTransmitError();	4	errorDefect = TRUE	RETURN
	—	5	—	

Row 3.2-1: The size of the queued MAC control frame is less than the PTQ space.

Row 3.2-2: In the absence of MAC control transmission credits, no action is taken.

Row 3.2-3: MAC control transmissions have precedence over client transmissions.

Row 3.2-4: If the transmission completes with an error, set an error defect indication.

Row 3.2-5: Otherwise, no error defect is indicated.

3.5 Arithmetic and logical operators

In addition to commonly accepted notation for mathematical operators, Table 3.3 summarizes the symbols used to represent arithmetic and logical (boolean) operations. Note that the syntax of operators follows standard C notation (see 3.14).

Table 3.3—Special symbols and operators

Printed character	Meaning
&&	Boolean AND
	Boolean OR
!	Boolean NOT (negation)
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
=	Assignment operator
//	Comment delimiter

3.6 Numerical representation

Decimal, hexadecimal, and binary numbers are used within this document. For clarity, decimal numbers are generally used to represent counts, hexadecimal numbers are used to represent addresses, and binary numbers are used to describe bit patterns within binary fields.

Decimal numbers are represented in their usual 0, 1, 2, ... format. Hexadecimal numbers are represented by a string of one or more hexadecimal (0-9,A-F) digits followed by the subscript 16, except in C-code contexts, where they are written as 0x123EF2, etc. Binary numbers are represented by a string of one or more binary (0,1) digits, followed by the subscript 2. Thus, the decimal number “26” may also be represented as “1A₁₆” or “11010₂”.

MAC addresses and OUI/EUI values are represented as strings of 8-bit hexadecimal numbers separated by hyphens and without a subscript, as for example, “01-80-C2-00-00-15” or “AA-55-11”.

3.7 Field notations

3.7.1 Use of italics

All field names or variable names (such as *level* or *myMacAddress*) and subfields within variables (such as *thisState.level*) are italicized within text, figures, and tables to avoid confusion between such names and similarly spelled words without special meanings. A variable or field name that is used in a subclause

heading or a figure or table caption is also italicized. Variable or field names are not italicized within C code, however, because their special meaning is implied by their context. Names used as nouns (e.g., subclassA0) are also not italicized.

3.7.2 Field conventions

This document describes values that are packetized or MAC-resident, such as those illustrated in Table 3.4.

Table 3.4—Names of fields and subfields

Name	Description
<i>newCRC</i>	Field within a register or frame
<i>thisState.level</i>	Subfield within field <i>thisState</i>
<i>thatState.rateC[n].c</i>	Subfield within array element <i>rateC[n]</i>

Run-together names (e.g., *thisState*) are used for fields because of their compactness when compared with equivalent underscore-separated names (e.g., *this_state*). The use of multiword names with spaces (e.g., “This State”) is avoided to prevent confusion between commonly used capitalized key words and the capitalized word used at the start of each sentence.

A subfield of a field is referenced by suffixing the field name with the subfield name, separated by a period. For example, *thisState.level* refers to the subfield *level* of the field *thisState*. This notation can be continued to represent subfields of subfields (e.g., *thisState.level.next* is interpreted to mean the subfield *next* of the subfield *level* of the field *thisState*).

Two special field names are defined for use throughout this standard. The name *frame* is used to denote the data structure comprising the complete MAC sublayer PDU. Any valid element of the MAC sublayer PDU can be referenced using the notation *frame.xx* (where *xx* denotes the specific element); thus, for instance, *frame.ttl* is used to indicate the *ttl* element of an RPR frame. In addition, the notation *MIB.xx* is used to reference an attribute of the Management Information Base defined within this standard; hence, *MIB.rprIfTable.rprIfEntry.rprIfIndex* references the interface index attribute of the RPR MIB.

Unless specifically specified otherwise, reserved fields are reserved for the purpose of allowing extended features to be defined in future revisions of this standard. For devices conforming to this version of this standard, nonzero reserved fields are not generated; values within reserved fields (whether zero or nonzero) are to be ignored.

3.7.3 Field value conventions

This document describes values of fields. For clarity, names can be associated with each of these defined values, as illustrated in Table 3.5. A symbolic name, consisting of uppercase letters with underscore separators, allows other portions of this document to reference the value by its symbolic name, rather than a numerical value.

Table 3.5—wrap field values

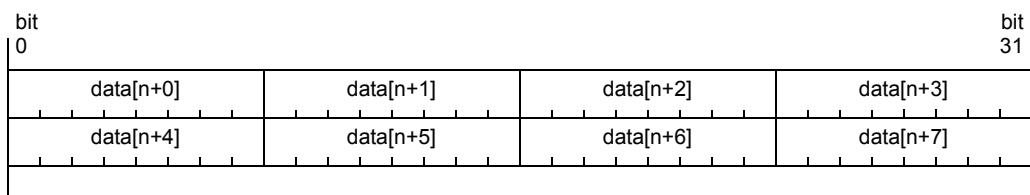
Value	Name	Description
0	WRAP_AVOID	Frame is discarded at the wrap point
1	WRAP_ALLOW	Frame passes through wrap points
2,3	—	Reserved

Unless otherwise specified, reserved values are reserved for the purpose of allowing extended features to be defined in future revisions of this standard. Devices conforming to this version of this standard do not generate reserved values for fields, and process fields containing reserved values as though the field values were not supported. The intent is to ensure default behaviors for future-specified features.

A field value of TRUE shall always be interpreted as being equivalent to a numeric value of 1 (one), unless otherwise indicated. A field value of FALSE shall always be interpreted as being equivalent to a numeric value of 0 (zero), unless otherwise indicated.

3.8 Bit numbering and ordering

Data transfer sequences normally involve one or more cycles, where the number of bytes transmitted in each cycle depends on the number of byte lanes within the interconnecting link. Data byte sequences are shown in figures using the conventions illustrated by Figure 3.2, which represents a link with four byte lanes. For multi-byte objects, the first (leftmost) data byte is the most significant, and the last (rightmost) data byte is the least significant.

**Figure 3.2—Bit numbering and ordering**

Figures are drawn such that the counting order of data bytes is from left to right within each cycle, and from top to bottom between cycles. For consistency, bits and bytes are numbered in the same fashion.

NOTE—The transmission ordering of data bits and data bytes is not necessarily the same as their counting order; the translation between the counting order and the transmission order is specified by the appropriate reconciliation sublayer. In particular, SONET standards use different bit numbering and ordering conventions from those specified above.

3.9 Byte-sequential formats

Figure 3.3 provides an illustrative example of the conventions to be used for drawing frame formats and other byte-sequential representations. These representations are drawn as fields (of arbitrary size) ordered along a vertical axis, with numbers along the left sides of the fields indicating the field sizes in bytes. Fields are drawn contiguously such that the transmission order across fields is from top to bottom. The example shows that *field1*, *field2*, and *field3* are 1-, 1-, and 6-byte fields, respectively, transmitted in order starting with the *field1* field first. As illustrated on the right-hand side of Figure 3.3, a multi-byte field represents a

sequence of ordered bytes, where the first through last bytes correspond to the most significant through least significant portions of the multi-byte field, and the MSB of each byte is drawn to be on the left-hand side.

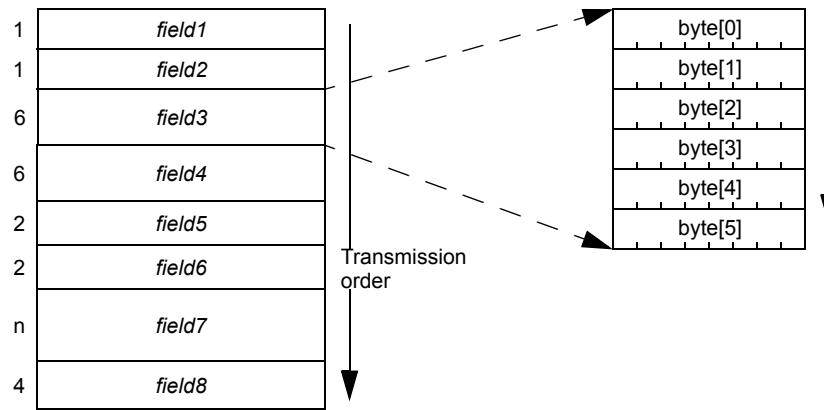


Figure 3.3—Byte-sequential field format illustrations

NOTE—Only the left-hand diagram in Figure 3.3 is required for representation of byte-sequential formats. The right-hand diagram is provided in this description for explanatory purposes only, for illustrating how a multi-byte field within a byte-sequential representation is expected to be ordered. The tag “Transmission order” and the associated arrows are not required to be replicated in the figures.

3.10 Left-to-right ordering

In many cases, bit fields within byte or multibyte objects are expanded in a horizontal fashion, as illustrated in the right side of Figure 3.4. The fields within these objects are illustrated as follows: Left-to-right is the byte transmission order; the left-through-right bits are the most significant through least significant bits, respectively.

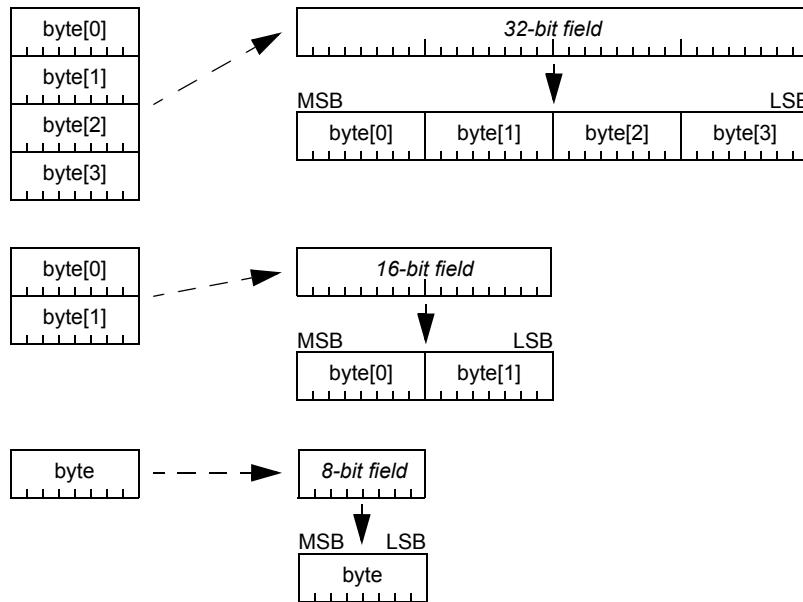


Figure 3.4—Multi-byte field illustrations

NOTE—Only the right-hand diagrams in Figure 3.4 are used within normative clauses. The left-hand diagrams are provided in this description for explanatory purposes only.

For consistency with bit and byte ordering conventions, bits are numbered in a left-to-right order. Depending on the interconnect, this RPR bit-numbering order may or may not differ from the physical layer bit transmission ordering.

3.11 Representation of MAC addresses

The format of MAC address fields within frames consists of OUI and *dependentID* fields and is illustrated for organization-specific frames in Figure 3.5.

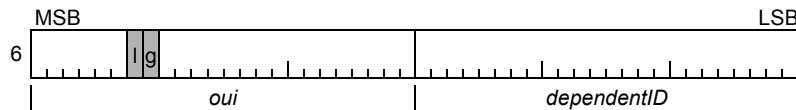


Figure 3.5—Organization-specific *controlDataUnit* field format

3.11.1 *oui*: A 24-bit organizationally unique identifier (OUI) field, used for the purpose of identifying the organization supplying the 24-bit *dependentID*. For clarity, the locations of the universally/locally administered and individual/group address bits are illustrated by the shaded *l* and *g* fields, respectively.

3.11.2 *dependentID*: A 24-bit field supplied by the *oui*-specified organization. Within this context, the *dependentID* is unique within a given organization; hence, the concatenation of the *oui* and *dependentID* provide a globally unique identifier.

An example of the mapping of actual values to the *oui/dependentID* fields is shown in Figure 3.6.

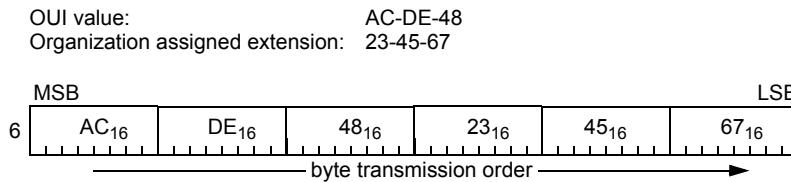
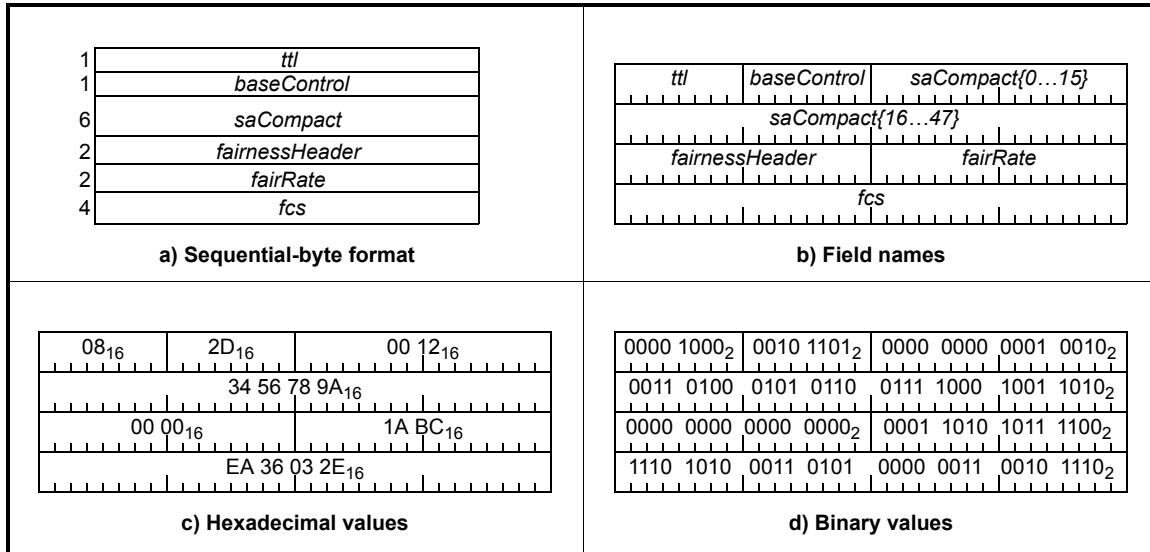


Figure 3.6—48-bit MAC address format

3.12 Mapping of numeric data values to fields

The mapping of hexadecimal and binary values to frame fields is illustrated in Figure 3.7.

The byte-sequential format of Figure 3.7-a is represented by Figure 3.7-b when transferred over a 32-bit data path. As an example, arbitrary numeric values have been assigned to the fields in Figure 3.7-b, and these values are represented in hexadecimal format in Figure 3.7-c and binary format in Figure 3.7-d.

**Figure 3.7—Illustration of fairness-frame structure**

3.13 Informative notes

Informative notes are used in this standard to provide guidance to implementers and to supply useful background material. Such notes never contain normative information, and implementers are not required to adhere to any of their provisions. An example of such a note follows.

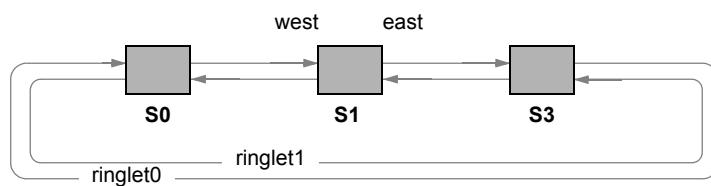
NOTE—This is an example of an informative note.

3.14 Conventions for C code used in state machines

Many of the state machines contained in this standard utilize C code functions, operators, expressions, and structures for the description of their functionality. Conventions for such C code can be found in Annex H.

3.15 Ringlet orientation conventions

By default, figures containing illustrations of RPR interfaces depict the east receive interface to ringlet0 on the upper left, the east transmit interface to ringlet0 on the upper right, the west receive interface to ringlet1 on the lower right, and the west transmit interface to ringlet1 on the lower left.

**Figure 3.8—Ringlet orientation conventions**

4. Abbreviations and acronyms

This standard contains the following abbreviations and acronyms:

BER	bit error ratio
CIR	committed information rate
CRC	cyclic redundancy check
dLOC	loss of continuity failure defect
EIR	excess information rate
EISS	enhanced internal sublayer service
FCS	frame check sequence
FDB	filtering database
FID	FDB identifier
FIFO	first in first out
FRTT	fairness round-trip time
FS	forced switch
GFP	generic framing procedure
GMII	gigabit media independent interface
GRS	GFP reconciliation sublayer
HDLC	high-level data link control
HEC	header error check
IANA	Internet Assigned Numbers Authority
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
ISS	internal sublayer service
ITU	International Telecommunication Union
LAN	local area network
LAPS	link access procedure—SDH
LCP	link control protocol
LLC	logical link control
LME	layer management entity
LOC	loss of continuity failure
LRTT	loop round-trip time
LSB	least significant bit
LVDS	low-voltage differential signals
LVTTL	low-voltage transistor-transistor logic
MAC	medium access control
MAN	metropolitan area network
MCFF	multi-choke fairness frame
MIB	management information base
MII	medium independent interface
MLME	MAC layer management entity
MS	manual switch
MSB	most significant bit
MSTI	Multiple Spanning Tree Instance
MTU	maximum transfer unit
OAM	operations, administration, and maintenance
OIF	Optical Internetworking Forum
OSI	open systems interconnect
PCS	physical coding sublayer
PDU	protocol data unit
PG	protection group

PGM	protection group member
PHY	physical layer
PICS	protocol implementation conformance statement
PIRC	protected inter-ring connection
PLME	physical layer management entity
PMA	physical medium attachment
PMD	physical medium dependent
PPM	parts per million
PPP	point to point protocol
PRS-1	1 Gb/s packetPHY reconciliation sublayer
PRS-10	10 Gb/s packetPHY reconciliation sublayer
PTQ	primary transit queue
RFC	request for comment
RPR	resilient packet ring
RS	reconciliation sublayer
RRTT	ring round-trip time
SAS	spatially aware sublayer
SCFF	single-choke fairness frame
SD	signal degrade
SDB	SAS database
SDH	synchronous digital hierarchy
SDU	service data unit
SF	signal fail
SME	station management entity
SNMP	simple network management protocol
SONET	synchronous optical network
SPI	system packet interface
SPI-3	System Packet Interface Level 3
SPI-4.1	System Packet Interface Level 4 Phase 1
SPI-4.2	System Packet Interface Level 4 Phase 2
SRS	SONET/SDH reconciliation sublayer
STQ	secondary transit queue
TP	topology and protection
VID	VLAN identifier
VLAN	virtual LAN
WAN	wide area network
WIS	WAN interface sublayer
WTR	wait to restore
XAUI	10 Gb attachment unit interface
XGMII	10 Gb media independent interface
XGXS	XGMII extender sublayer

5. Architecture overview

This clause contains summaries of the information contained in later normative clauses and annexes as an aid to understanding this standard. This clause does not specify any normative behavior; other clauses and normative annexes have precedence when apparent conflicts arise.

5.1 Terminology

The following terms have special meaning within the context of this standard.

5.1.1 aggressive: An algorithm that tends to exhibit an underdamped transient response.

5.1.2 congested: The indication from a local station that a threshold has been crossed for the declaration of a state of congestion in attempting to add fairness eligible traffic.

5.1.3 context: The topology and status database used by a source station to transmit a frame.

5.1.4 copy: The transfer of an inbound frame from the ring to the MAC sublayer. The copying of a frame does not imply its removal from the ring.

5.1.5 delay¹³: The time required to transfer information from one point to another.

5.1.6 domain: A set of contiguous stations associated with a common property, such as an indicated congestion condition or transit paths affiliated with a specific frame transfer.

5.1.7 edge: A span on which data frames are not allowed to pass. This is a span on which a wrap is present in a wrapping ring, or a span that is unused for data transmission due to a protection condition in a steering ring.

5.1.8 hop: The distance associated with the transfer of data from one station to the next station in its path. This distance between adjacent stations is described as “one hop.”

5.1.9 insert: The placement of a frame on a ringlet by the source station.

5.1.10 latency: *Syn: delay.*

5.1.11 neighbor: A station that is exactly one link distant from a given station on the network.

5.1.12 network: A set of communicating stations and the media and equipment providing connectivity among the stations.

5.1.13 normalize: To adjust a local value to allow a global interpretation when the value is communicated to other stations on the ringlet.

5.1.14 opportunistic: The property of an algorithm that seeks the ability to utilize capacity that is available, or that becomes available.

5.1.15 path: The specific sequence of stations and links traversed by a frame transferred between two stations on the network.

5.1.16 police: The enforcement of traffic parameters.

¹³*Delay* and *latency* are synonyms for the purpose of this standard. *Delay* is the preferred term.

5.1.17 receive: The transfer of an inbound frame from the ring to the medium access layer of the station.

5.1.18 span: The portion of a ring bounded by two adjacent stations. A span consists of a pair of unidirectional links transmitting in opposite directions.

5.1.19 steering: A protection method in which a source station redirects a unicast frame to the ringlet retaining connectivity to a destination station, or a multicast/broadcast frame to one or both ringlets retaining connectivity to destination stations.

5.1.20 strip: The removal of a frame from a ringlet.

5.1.21 transit: The passing of a frame through a station via the ring.

5.1.22 upstream: The direction opposite that of the data flow on a ringlet.

5.1.23 wrapping: The transmission of a frame on an opposing ringlet in order to route around a fault in a given ringlet.

5.2 Layer model

The RPR layer model and its relationship to the open systems interconnect (OSI) reference model is illustrated in Figure 5.1. The medium access control (MAC) control sublayer, MAC datapath sublayer, and reconciliation sublayers are specified within this document, as are the MAC service interface and PHY service interface supported by the sublayers.

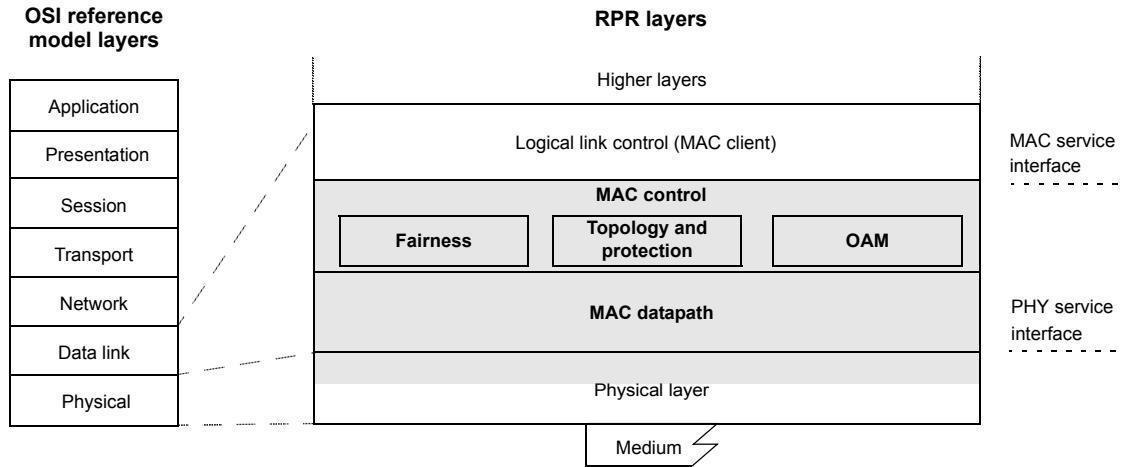


Figure 5.1—RPR service and reference model relationship to the ISO/IEC OSI reference model

The MAC service interface (see 6.4) provides service primitives used by MAC clients to exchange data with one or more peer clients, or to transfer local control information between the MAC and the MAC client. The MAC control sublayer (see 6.6) controls the datapath sublayer, maintains the MAC state and coordination with the MAC control sublayer of other MACs, and controls the transfer of data between the MAC and its client. The MAC datapath sublayer (see Clause 7) provides data transfer functions for each ringlet.

The PHY service interface is used by the MAC to transmit and receive frames on the physical media. Distinct reconciliation sublayers specify mapping between specific PHYs and (optional) medium independent interfaces (MIIs). This standard includes the definition of a reconciliation sublayer for each of the most commonly used PHYs and permits other reconciliation sublayers that conform to the requirements in Clause 8.

5.3 Ring structure

RPR employs a ring structure using unidirectional, counter-rotating ringlets. Each ringlet is made up of links with data flow in the same direction. The ringlets are identified as ringlet0 and ringlet1, as shown in Figure 5.2. The association of a link with a specific ringlet is not altered by changes in the state of the links or stations.

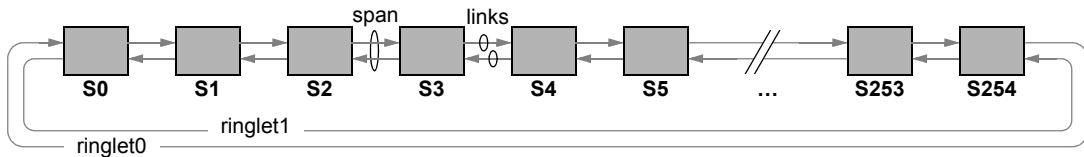


Figure 5.2—Dual-ring structure

Stations on the ring are identified by an IEEE 802 48-bit MAC address as specified in IEEE Std 802-2002. All links on the ring operate at the same data rate, but they may exhibit different delay properties.

The portion of a ring bounded by adjacent stations is called a span. A span is composed of unidirectional links transmitting in opposite directions.

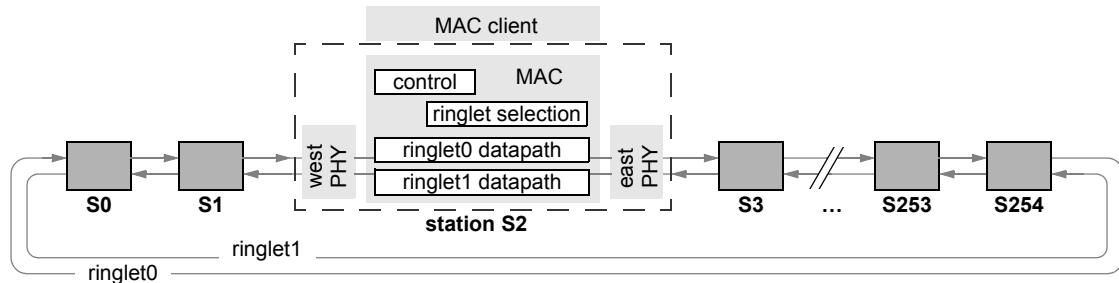
Station Y is said to be a downstream neighbor of station X on ringlet0/1 if the station X traffic becomes the receive traffic of station Y on the referenced ringlet. Thus, station S5 is the downstream neighbor of station S4 on ringlet0; similarly station S2 is the downstream neighbor of station S3 on ringlet1.

Station Y is said to be an upstream neighbor of station X on ringlet0/1 if the station Y traffic becomes the receive traffic of station X on the referenced ringlet. Thus, station S4 is the upstream neighbor of station S5 on ringlet0; similarly station S3 is the upstream neighbor of station S2 on ringlet1.

5.4 Station structure

A station is composed of one client entity, one MAC entity, and two PHY entities. Each PHY is associated with a span shared with a neighboring station. The MAC entity contains one MAC control entity, a ringlet selection entity, and two datapath entities (one datapath is associated with each ringlet). The PHY transmitting on ringlet0 and receiving on ringlet1 is identified as the east PHY, as illustrated in Figure 5.3. The PHY transmitting on ringlet1 and receiving on ringlet0 is identified as the west PHY. The ringlet0 datapath receives frames from the west PHY and transmits or retransmits frames on the east PHY. The ringlet1 datapath receives frames from the east PHY and transmits or retransmits frames on the west PHY.

NOTE—The MAC supports only a single service interface and assumes there is a single client entity attached to that service interface. Each client may contain multiple subclient entities (effectively becoming a multifunction station), but the definitions of such subclient entities and their sharing of the single service interface are beyond the scope of this standard.

**Figure 5.3—Station structure**

5.5 MAC architecture

5.5.1 Datapath connectivity

Figure 5.4 provides a single station view of the MAC architecture (see 6.6). The MAC entity associated with the station is shown to contain one instance of the MAC control sublayer entity and two instances of the MAC datapath sublayer entity. Each instance of the MAC datapath serves one of the two ringlets. The MAC client sends data frames to ringlet selection and receives data frames from each of the two MAC datapath instances. The MAC control entity sends control frames to ringlet selection and receives control frames from each of the two MAC datapath instances.

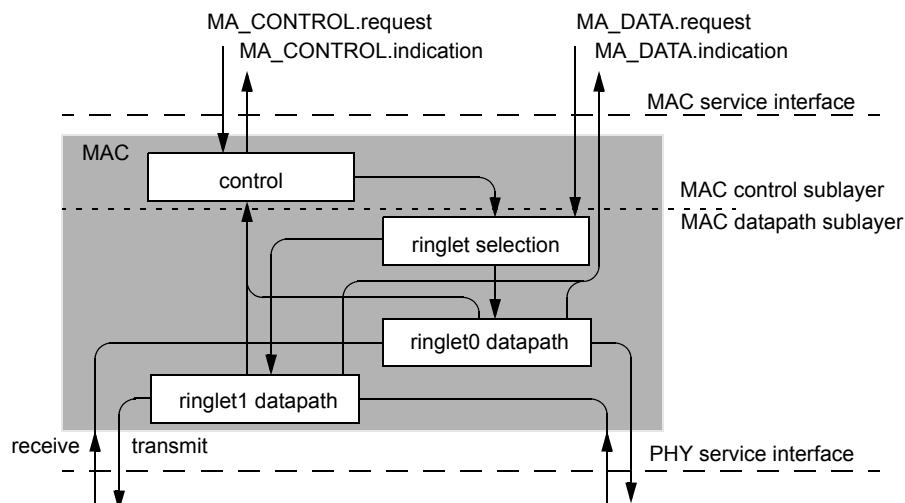
**Figure 5.4—Single station view of MAC architecture**

Figure 5.5 provides an end-to-end view of the MAC entities involved in a data transfer from station S1 to station S3 via transit-station S2. The MAC client interface functions are used only at those stations where frames are added to or copied from the ring. In the case of a unicast, this corresponds to the source and destination stations. In the absence of breaks in ringlet continuity, a frame is processed by the same MAC datapath entity in each station through which it passes (i.e., ringlet0 MAC datapath in each transit station or ringlet1 MAC datapath in each transit station).

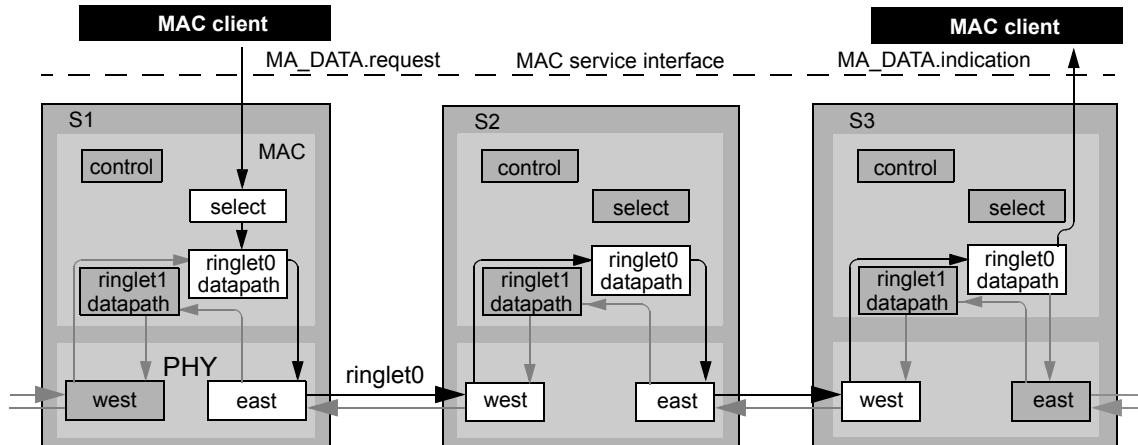


Figure 5.5—End-to-end view of MAC architecture

5.5.2 Ringlet selection

5.5.2.1 Destination addressing

The ringlet selection entity within the MAC datapath sublayer (see 7.7.1) checks the destination_address, source_address, and other parameters to select which ringlet(s) transmit the frame, whether and how to flood the frame, how to protect the frame, and which frame format (basic or extended) is used.

As part of ringlet selection, stations are capable of comparing the destination_address parameter against each of the MAC addresses associated with attached stations. These comparisons allow the station to determine the physical location of the destination, and that physical location knowledge allows the station to determine whether the destination is best reached through ringlet0 or ringlet1 (see 5.7.1 and 5.7.2.2).

5.5.2.2 Secondary addresses

In addition to its primary physical MAC address, each station can have one or two additional MAC addresses called secondary addresses (see Figure 5.6-a).

If secondary addresses are not supported in the transmitter, transmissions to secondary addresses are flooded, although only the actual destination (see Figure 5.6-b) copies the received frame to the client.

Source stations can translate secondary addresses (see Figure 5.6-c) to their corresponding primary MAC address before frame transmission. Such secondary-address source translation avoids additional flooding and receive station filtering. To facilitate source translation, stations can advertise their secondary addresses through ATD frames, for intended inclusion into the topology databases of other stations.

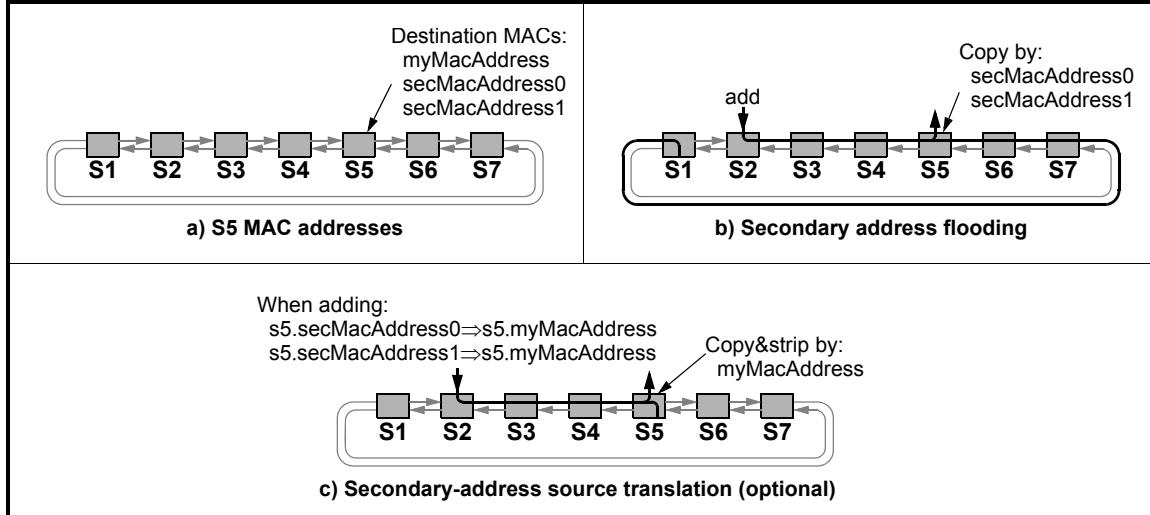


Figure 5.6—Secondary address transmissions

5.5.3 MAC datapath flows

The MAC datapath (see 7.4) provides logic that determines which frames are stripped and transit queues that hold received frames waiting to be retransmitted. All stations have a primary transit queue (PTQ), typically only a few MTUs in size. Stations have the option of providing another, typically much larger, secondary transit queue (STQ), as illustrated in Figure 5.7-a and Figure 5.7-b, respectively.

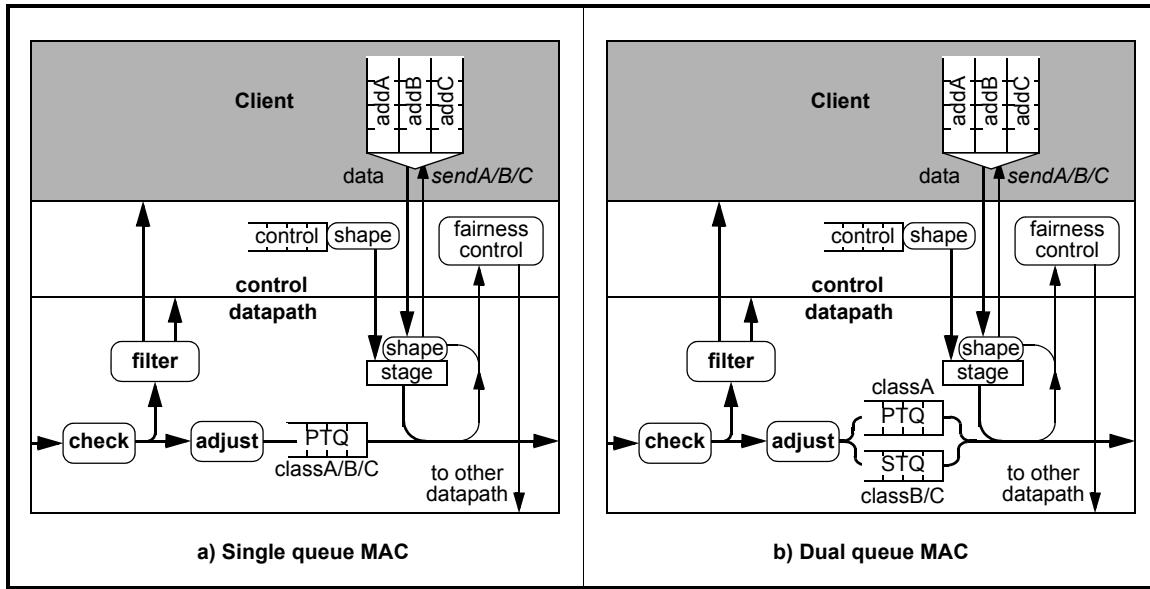


Figure 5.7—MAC data flows

Frames are transmitted from a MAC-resident stage queue, rather than transmitted directly from the client, to decouple MAC-to-client interface timings from the timings of the physical-layer interface. Flow control and prioritization protocols are therefore applied when the frames are accepted into the stage queue, rather than when the staged frames are actually transmitted. Thus, the stage-queue traffic is effectively on the ringlet

and is not affected by the bandwidth allocations, fairness weights, or fairness flow-control indications. The stage queue is small and has no shaper-enforced limits, so stage-queue frames are quickly sent and the stage-queue effect on flow-control latencies is minimal.

All traffic is shaped (see 7.5) before transmission. Control frames are shaped to limit the jitter impact on client-supplied classA transmissions. Client-supplied data frames are shaped to ensure conformance with pre-negotiated bandwidth allocations. Shaping involves monitoring transmitted frame bandwidths and sending {*sendA*, *sendB*, *sendC*} flow-control indications to the client.

The *sendC* indication (that controls classB-EIR and classC traffic flows) provides a hop-count distance, rather than a minimal stop/start indication. The intent is to avoid head-of-line blocking of near-in transmissions due to blocked far-out congestion conditions.

5.5.4 Receive rules

At a high level, the receive functionality (see 7.6) includes check, adjust, and filter functions. The check rules are responsible for discarding errored and expired frames as well as for counting receive flow statistics. The adjust rules are responsible for stripping frames at their intended destination, adjusting frame fields, and placing frames in the correct transit queue. The filter rules are responsible for deciding whether frames are copied to the client, MAC control sublayer, or neither.

5.6 MAC service

The MAC service is defined by the semantics of the MAC service interface (see 6.4) illustrated in Figure 5.8. A request is associated with the direction from the MAC client to MAC, whereas an indication is associated with the direction from the MAC to the MAC client. Control primitives are used by the local MAC client to request and receive control information from the local MAC. Data primitives are used by the local MAC client to exchange client-layer protocol data units (PDUs) with remote MAC clients.

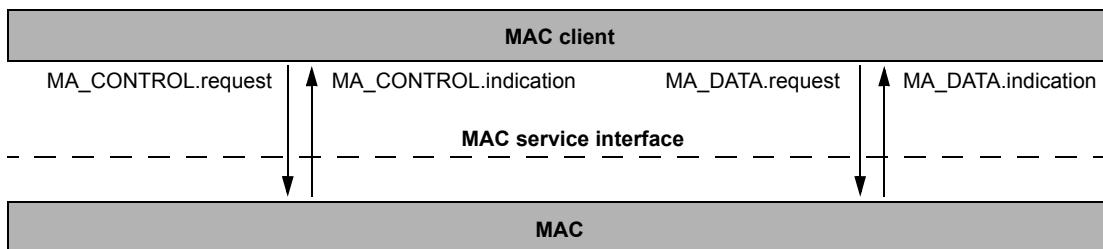
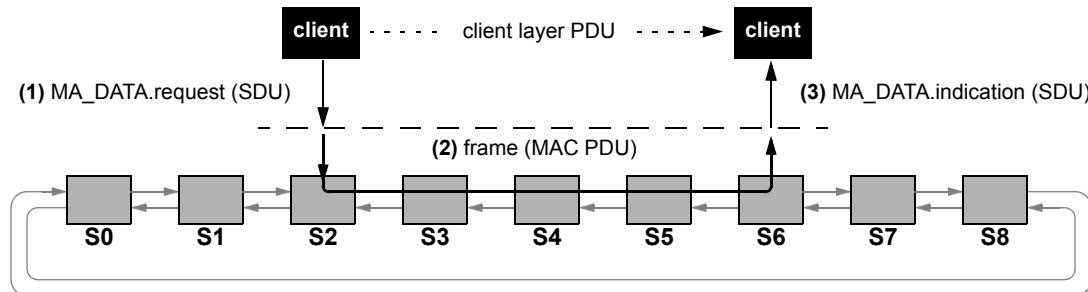


Figure 5.8—MAC service interface

5.6.1 MAC data primitives

Figure 5.9 illustrates the unicast transfer of a client-layer PDU between clients at station S2 and station S6. A MAC data request is (1) issued by the MAC client to the station S2 MAC. The client-layer PDU is carried as a MAC-layer service data unit (SDU), identified as a parameter of the data request primitive. The SDU is (2) encapsulated in a MAC header and is transferred from station S2 to station S6 as a frame (MAC-layer PDU). The MAC header is removed, and the remaining payload is transferred (3) to the local MAC client via a MAC data indication primitive; the primitive carries the SDU as a parameter. From the perspective of the client, these steps are equivalent to the transfer of a client-layer PDU from one client to the other, as shown by the dashed arrow in Figure 5.9.

**Figure 5.9—MAC data primitives**

5.6.2 Service classes

In addition to the service data unit, the MAC data primitive identifies a service class (A, B, or C) with which the data transfer is associated (see 6.3 and 7.3), as summarized in Table 5.1. The classA service provides low-jitter transfer of traffic (and therefore lower worst-case delays) up to its allocated rate. Traffic above the allocated rate is rejected. The classB service provides bounded delay transfer of traffic at or below the committed information rate (CIR) and a best-effort transfer of the excess information rate (EIR) data beyond the committed rate. The classC service provides best-effort data-transfer services.

Table 5.1—Service classes and their quality-of-service relationships

class of service			qualities of service				fairness eligible
class	examples of use	subclass	guaranteed bandwidth	jitter	type	subtype	
A	real time	subclassA0	yes	low	allocated	reserved	no
		subclassA1	yes	low	allocated	reclaimable	
B	near real time	classB-CIR	yes	bounded			yes
		classB-EIR	no	unbounded	opportunistic	reclaimable	
C	best effort	—					

The classA service is implemented as distinct subclassA0 and subclassA1 partitions. The subclassA1 partition is more efficient, but limited by the sizes of secondary transit queues (see Clause 7).

The ring capacity required to support the classA service and classB-CIR service is allocated via provisioning, and these services can be characterized as allocated services. The provisioning activity is expected to ensure that the aggregate service commitment on each link does not exceed that link's capacity. The allocation rates distributed by provisioning regulates access to these guaranteed services.

NOTE—Ring capacity has to be ensured to support classA and classB-CIR service guarantees. This could be done by allocating bandwidth through provisioning that could prevent over-provisioning the ring through an external service provisioning mechanism. The communication of allocation information must be phased so as to not to create a case where transitioning allocation levels temporarily cause cumulative allocations to exceed the link capacity. However, this service provisioning mechanism is beyond the scope of this standard.

5.6.3 MAC flow control primitives

The MAC control primitive is used by the MAC client to request and receive MAC control information or action from the local MAC. Examples of control information include station configuration, congestion status, send status, and ring topology.

One use of the MAC control primitive (see 7.5) is in providing flow-control indications to regulate client access to the ring. All traffic entering a ringlet from a client is subject to rate control by shaping. In the case of allocated services, shaping parameters are changed only when allocations change. In the case of opportunistic services, shaping parameters are dynamic and are adjusted when fair rate values are recomputed by the fairness algorithm.

The MAC enforces bandwidth constraints by refusing to allow its client to transmit more than the station's allowed rate or, if there is no allocated-rate limit, up to the ring's total bandwidth.

For all service classes, a status indication is sent from the MAC to the client indicating whether the client is, or is not, allowed to transfer data. For opportunistic service, the distance from the local station to the farthest allowed destination, if any, is also provided. The distance is represented as the number of links traversed, also known as the hop-count. This information allows the client to queue traffic distinctly for each destination, avoiding head-of-line blocking.

5.7 Frame transmissions

A ring supports the transmission of frames (see 7.7) from a source station to a destination station associated with an individual MAC address (unicast), to a set of bridges possibly associated with an individual MAC address (flooded), to a set of stations associated with a group MAC address (multicast), or to all stations (broadcast).

5.7.1 Unicast transmissions

A local unicast frame is transmitted on one of the ringlets, as illustrated in Figure 5.10-a. A unicast frame is added by the source station and is normally stripped at the destination station. For robustness, a unicast frame is also stripped from the ringlet based on a matching *sa* (source address) field or an expired *ttl* (time to live) field, whichever comes first. Within Figure 5.10, and other data-flow illustrations, the end-of-flow curves identify where the frames are stripped.

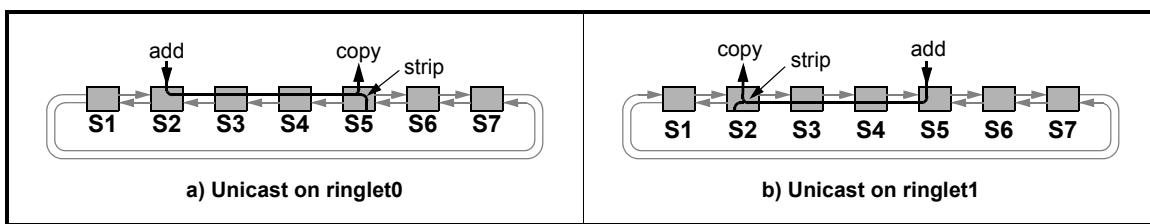


Figure 5.10—Unicast frame transmissions

The frame is normally copied at the destination station for delivery to the local MAC client or MAC control entity. If ringlet selection is based on shortest hop-count, a response frame is likely to take an opposing ringlet path, as illustrated in Figure 5.10-b.

5.7.2 Flooded transmissions

Transparent bridging relies on the ability to flood unknown-unicast frames (see 7.7.1.3 and 7.7.1.4) to all bridges in order to ensure that the frame can be delivered to all segments within the bridged LAN.

A flooded frame transits a sequence of stations and is normally stripped from the ring based on the expiration of the *ttl* (time to live) field, or stripped from the ring when returned to the source station. A flooded frame is also copied to the client bridge entity at each intermediate station. The intent is to mimic the behavior of a broadcast medium, for non-local frame transmissions, to facilitate transparent bridging applications.

The selected *fi* (flooding indication) field value within the *extendedControl* field of a frame (see 5.8) determines whether the frame is to be not-flooded, unidirectionally flooded (see 5.7.2.1), or bidirectionally flooded (see 5.7.2.2).

5.7.2.1 Unidirectional flooded transmissions

Unidirectional flooding involves sending the frame to all other stations in the ring via either ringlet0 (as illustrated in Figure 5.11) or ringlet1 (not illustrated). The frame can be stripped from the ring based on an expiration of the *ttl* field, or (less efficiently) stripped from the ring when returned to the source station, as illustrated in Figure 5.11-a and Figure 5.11-b, respectively.

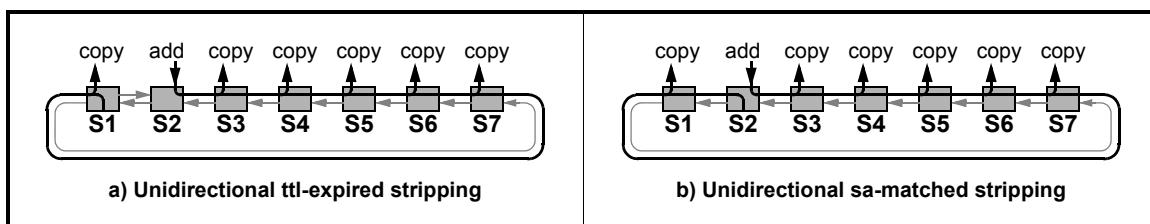


Figure 5.11—Flooded transmissions

5.7.2.2 Bidirectional flooded frames

Bidirectional flooding involves sending frames to all other stations in the ring via both ringlet0 and ringlet1. A bidirectional flooded frame is replicated, transmitted on both ringlets, and stripped based on the expiration of the distinct *ttl* fields at an agreed upon span, called the cleave point (see 7.7.2). This cleave point is illustrated by the dark dotted line in Figure 5.12-a and Figure 5.12-b. Unless the datapath is wrapped at its end points (see 5.14.1), a span failure causes the cleave point to be located at the failed-span location, as illustrated in Figure 5.12-b.

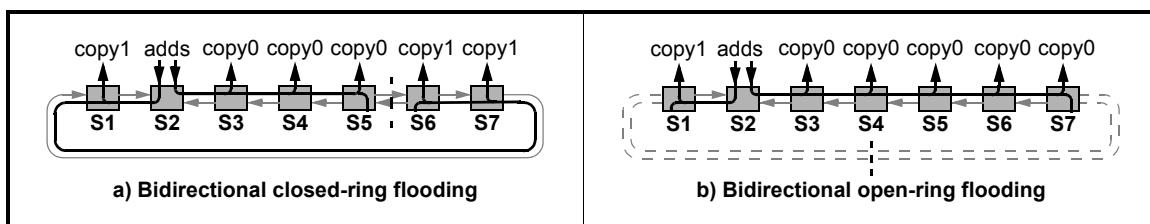


Figure 5.12—Cleave point migration

5.7.3 Multicast transmissions

A multicast frame is transmitted like an unknown-unicast flooded frame, in either a unidirectional or bidirectional fashion, as illustrated in Figure 5.11 and Figure 5.12, respectively. The frame is copied to the local client or MAC control entity at each intermediate station (not just bridge stations). Multicast group membership is identified by the *da* (destination address) field within the frame header.

5.8 Frame formats

The fields within the basic data frame (see Clause 9) are illustrated in Figure 5.13. The numbers to the left of each field indicate the field's size in bytes. End-to-end parameters (shaded white) are supplemented with additional information (shaded grey) necessary to transmit the frame over an RPR ring (see 9.2).

1	<i>ttl</i>	— Hop count to destination (time to live)
1	<i>baseControl</i>	— Frame type, service class, and baseline controls
6	<i>da</i>	— Destination client station (48-bit destination address)
6	<i>sa</i>	— Local RPR source station (48-bit source station address)
1	<i>ttlBase</i>	— Snapshot of <i>ttl</i> , for computing hop count to source
1	<i>extendedControl</i>	— Extended flooding and consistency checks
2	<i>hec</i>	— The 16-bit CRC for the header (header error check)
2	<i>protocolType</i>	— Form and function of the following <i>serviceDataUnit</i>
n	<i>serviceDataUnit</i>	— Data provided by the client (service data unit)
4	<i>fcs</i>	— The 32-bit CRC for <i>protocolType</i> & <i>serviceDataUnit</i> fields

Figure 5.13—Basic data frame format

5.9 Frame transmissions

On a broadcast medium, frames are directly applied to the medium and implicitly stripped when transmission ends. On an RPR ring, explicit mechanisms must be used to robustly strip frames, as listed below.

- a) Destination. The *da* address equals the station's *myMacAddress* and the flood indication is not set.
- b) Source. The *sa* address equals the station's *myMacAddress* value.
- c) Expired. The *ttl* field reaches zero, which indicates the frame has passed through the intended number of stations.

5.9.1 Local-source/local-destination transmissions

For source and destination stations attached to a common RPR ring, Figure 5.14 illustrates how client-supplied MA_DATA.request parameters (see 6.4.1) are mapped to an RPR frame, supplemented with additional RPR-specific fields, and sent by the source-station S1 MAC. Similarly, selected fields in an RPR frame are mapped to the MA_DATA.indication primitive (see 6.4.2) at the destination-station S7 MAC.

The rounded rectangles above the destination frame indicate how that frame is processed within stations. The copy condition indicates when the frame is copied to the client. The strip and discard conditions indicate when the frame is removed, based on normal and error conditions, respectively. The mini-box structures above stations S2, S4, and S6 indicate the bridged attachment of remote stations.

In general, the *da* field identifies the end destination station or multicast stations; the *sa* field identifies the local source station. Specifically within Figure 5.14, the *da* and *sa* fields identify the destination and source clients, respectively. The clients with source and destination stations are identified by a black rectangle within the station.

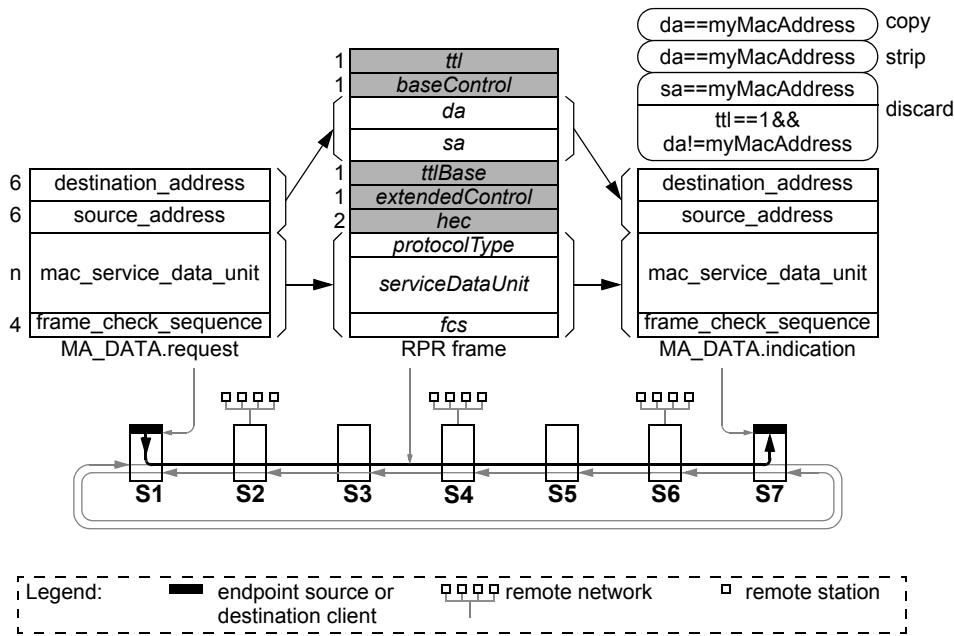


Figure 5.14—Local-source/local-destination transmissions

Similar mappings apply to multicast, remote destination, and remote-sourced frames, as illustrated in 5.9.2 through 5.9.5.

5.9.2 Local-source multicast transmissions

Client-supplied parameters with a multicast or broadcast destination_address parameter are mapped to an RPR frame, supplemented with additional RPR-specific fields, and transmitted by the MAC to the addressed stations, as illustrated in Figure 5.15. A flood indication within the *extendedControl* field of multicast frames causes them to be flooded to all stations and stripped when the *ttl* field has expired or the frame has returned to its *sa*-identified local-source station.

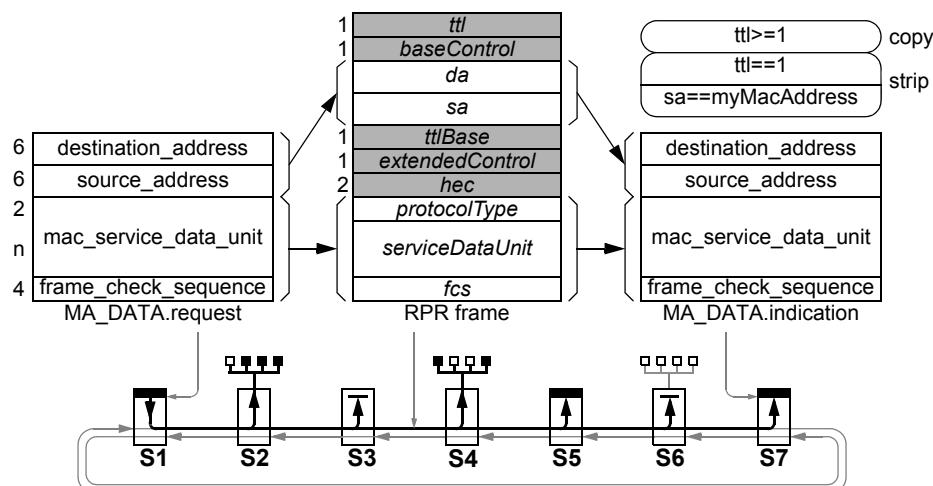


Figure 5.15—Local-source multicast transmissions

The *da* and *sa* fields identify the multicast-group and source-client addresses, respectively. When using bidirectional flooding (not illustrated), the transmitted *ttl* field in both frames must be set correctly to ensure completeness without duplication. For unidirectional flooding (as illustrated), the *ttl* can be set to strip the frame at the station's upstream neighbor or set to a larger value causing the frame to be stripped at its source.

5.9.3 Local-source unknown-unicast transmissions

Client-supplied local-source unknown-unicast service parameters are placed into an RPR frame (see Annex F), supplemented with additional RPR-specific fields, and transmitted by the MAC to the addressed stations, as illustrated in Figure 5.16. A flood indication within the *extendedControl* field of unknown-unicast frames causes them to be flooded. The unknown-unicast frames are treated the same as local-source multicast (see 5.9.2), except that the *da* field contains a unicast address.

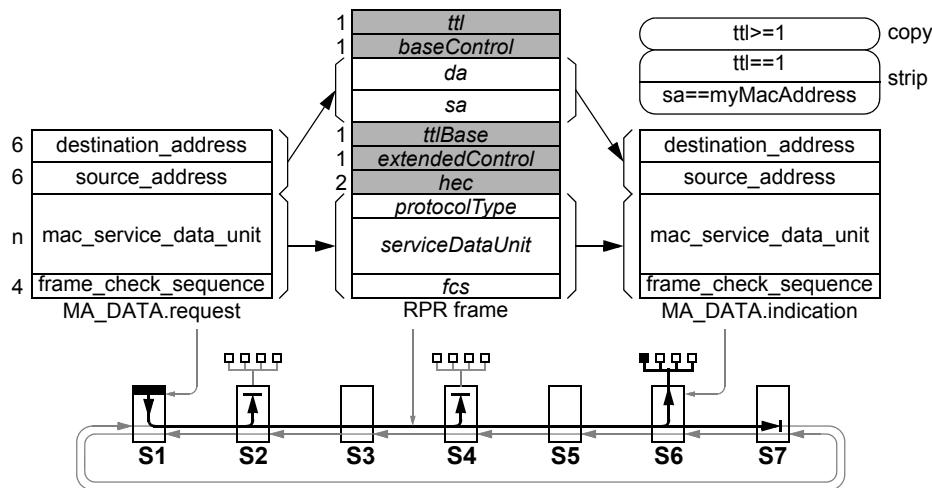


Figure 5.16—Local-source unknown-unicast transmission

Although these local-source unknown-unicast frames are flooded to all stations, a *da* address comparison prevents them from being passed to unrelated non-bridge station (S3, S5, and S7) clients. The learning state in bridge stations (S2, S4, and S6) determines whether these frames are flooded through bridge stations.

5.9.4 Remote-source unicast transmissions

Client-supplied remote-source unicast service parameters are placed into an RPR frame, supplemented with additional RPR-specific fields (see Annex F), and transmitted to the addressed stations, as illustrated in Figure 5.17. A flood indication within the *extendedControl* field of remote-source unicast frames (not illustrated) causes them to be flooded. Although the destination station may be a local (S7) station, these frames are flooded so that other bridge stations can learn the remote-source station address.

Although remote-source frames are flooded to all stations, a *da* address comparison prevents them from being passed to unrelated non-bridge station (S1, S3, S5, and S7) clients. The *sa* address ensures the frame will be stripped if it returns to its source. The learning state in bridge stations (S2, S4, and S6) determines whether these frames pass through bridge stations.

In Figure 5.17, the 48-bit *destinationAddress* and *sourceAddress* fields identify the remote end-station destination and end-station source MACs, respectively. Remotely sourced frames are passed as unmodified RPR payloads, thereby supporting end-to-end error checking of bridged frames with the same *fcs* specification.

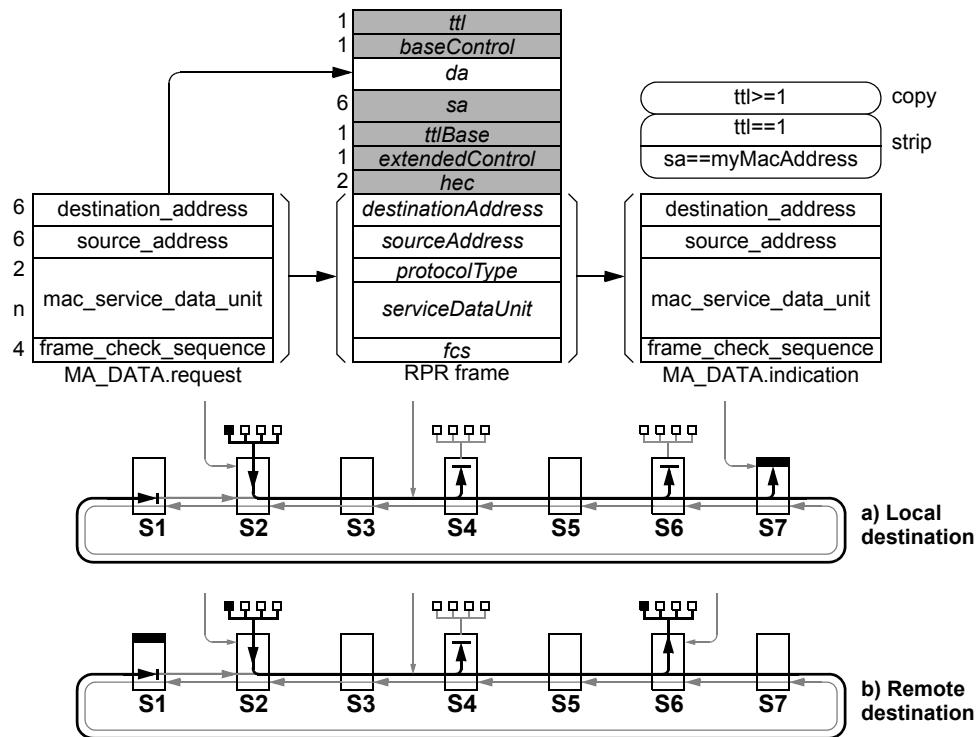


Figure 5.17—Remote-source unicast transmissions

5.9.5 Remote-source multicast transmissions

Figure 5.18 illustrates how remote-source broadcast/multicast frames are transmitted (see Annex F) by supplementing the header with additional RPR-specific fields (see 7.7.1). A flood indication within the *extendedControl* field of remote-source multicast frames causes them to be flooded. The contents of the header's *da* field are different, but the rules and formats are otherwise the same for remote-source unicast (see 7.7.2.1) and these multicast frames.

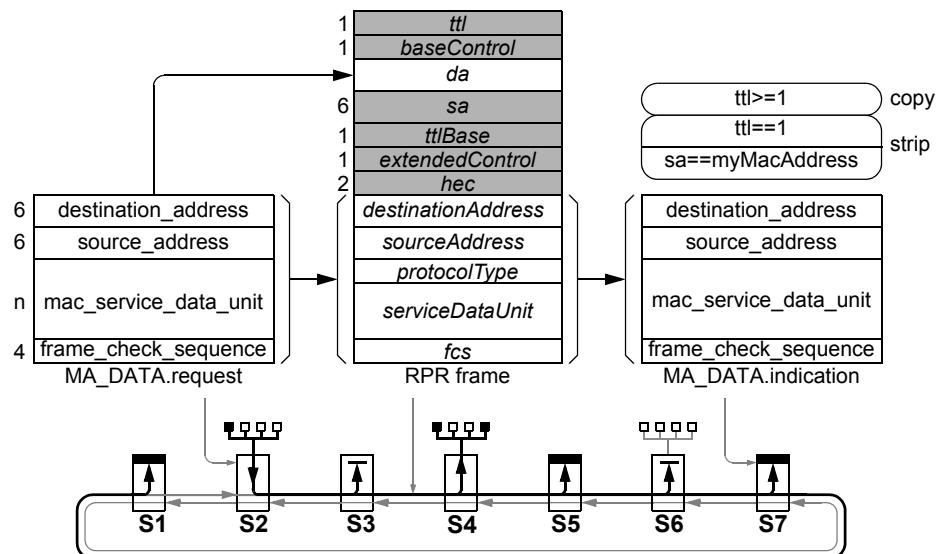


Figure 5.18—Remote-source multicast transmissions

5.10 Spatial reuse

The adding of individual frames is not synchronized between ringlets; the frame transmission event on any one link is independent of frame transmission events on other links. With the RPR ring topology, this allows per-link bandwidths to be utilized beyond that possible with other ring-based LAN technologies such as IEEE Std 802.5™-1998 Token Ring [B5]¹⁴ or ANSI FDDI-based protocols, as illustrated in Figure 5.19.

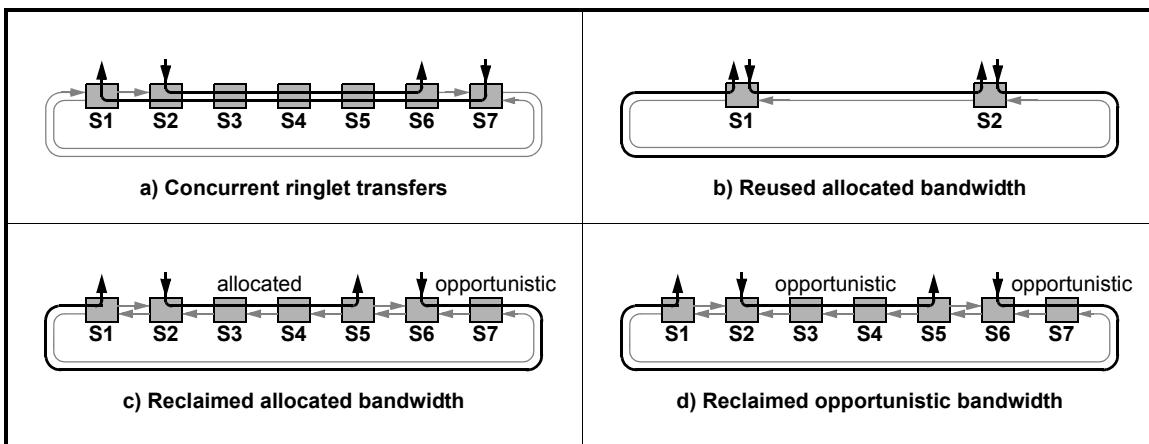


Figure 5.19—Bandwidth recycling mechanisms

By supporting concurrent per-ringlet transmissions (see Figure 5.19-a), the bandwidth available to stations on a ringlet exceeds the individual link capacity. The effective levels of allocated bandwidths (see Figure 5.19-b) are similarly improved, because allocated bandwidths can be independently committed on nonoverlapping segments.

The stripping of unicast frames at their destination station implies that allocated transfers use capacity only on links between their source and destination stations. Opportunistic bandwidth is therefore available on the remaining portion of the ringlet. This unused allocated bandwidth may therefore be reclaimed (see Figure 5.19-c) for opportunistic transfers. There are restrictions on this form of bandwidth reuse, in that some portions of the classA bandwidth (called subclassA0, see 7.3) can be safely reused on nonoverlapping segments, but cannot be reclaimed by other service classes.

The effective bandwidth utilization is further improved by allowing opportunistic bandwidth (see Figure 5.19-d) to be reclaimed on nonoverlapping segments of the ring. The fairness protocol automatically detects and resolves congestion point locations, avoiding the need to accurately predict opportunistic traffic patterns.

Spatial reuse for non-local frame transmission is supported by the optional spatially aware sublayer (see 5.18).

5.11 Bandwidth allocation

A single queue station processes all classA traffic as subclassA0. A dual station separates the classA traffic into subclassA0 and subclassA1 partitions. The subclassA1 partition is more efficient due to its reclaimable capabilities, but supportable levels of subclassA1 bandwidth are limited by the size of secondary transit queues (STQ) within the MAC (see 7.3). This partitioning is enforced within the MAC; it is not visible to the client, which is only constrained by the cumulative classA (subclassA0 plus subclassA1) bandwidth constraints.

¹⁴Numbers in brackets correspond to the numbers of the bibliography in Annex A.

5.11.1 Allocation enforcement

The committed subclassA0 bandwidths correspond to committed bandwidths that cannot be reclaimed by non-classA0 traffic. However, they can be statically reused in the sense that subclassA0 bandwidths can be recommitted on nonoverlapping source-to-destination segments. For nonoverlapping transfers, the subclassA0 bandwidths are effectively committed around the ring, even when not utilized, as illustrated by the dotted lines in Figure 5.20-a.

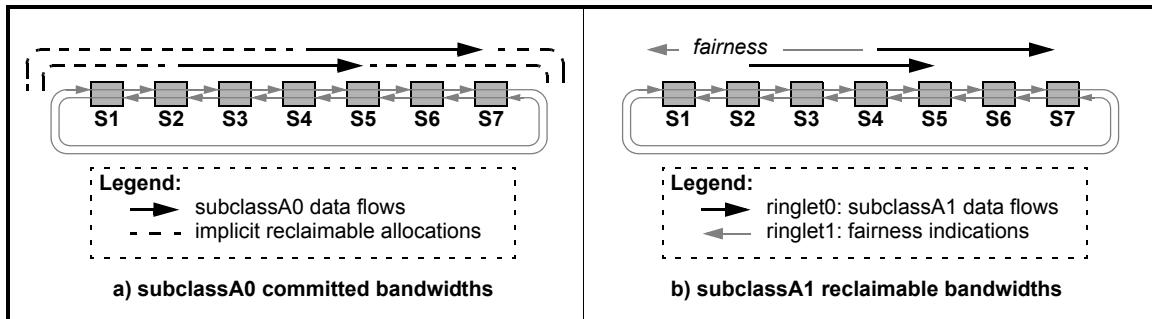


Figure 5.20—ClassA allocation enforcement

To maintain subclassA0 guarantees in the presence of long links and/or absence of STQs, each station throttles its other transmissions. This permits transmission of local subclassA0 frames and sustaining idles for one or more downstream stations.

The allocated subclassA1 bandwidths correspond to reclaimable bandwidths that can be dynamically reclaimed by non-classA traffic. When uncongested, stations send their allocated subclassA1 bandwidths without complaint (i.e., without indicating congestion in opposing-ringlet flow-control indications), as illustrated by the black arrows in Figure 5.20-b. In this illustration, ringlet0 data frame traffic is assumed.

When a congestion condition is detected, the congested station complains to upstream stations by means of fairness indication frames sent on the opposing ringlet, as illustrated by gray arrows in Figure 5.20-b. In this illustration, ringlet1 fairness frames provide indications of ringlet0 data-frame congestion.

Due to logic and transmission delays, a congested station S4 will observe delays between its assertion of a congestion indication and the indication-invoked reduction of incoming traffic. During this delay, station S4 continues to transmit its classA traffic; the classB/classC transit traffic is queued during these classA transmissions.

The size of the STQ and the round-trip signal propagation delays limit the levels of supportable subclassA1 traffic. Stations without STQs cannot send subclassA1 traffic (although they can transit subclassA1 traffic sent by others).

5.11.2 Allocation consistency

Bandwidth allocation implies the reservation of capacity on a ringlet. An allocation can be characterized as uniform or spatial. Uniform allocation uses a single rate per class for the entire ring. Spatial allocation uses independent rates per class for each link. Uniform allocation makes no distinction among traffic flows based on the number of spans traversed by their paths. Spatial allocation differentiates traffic levels on a per hop-count basis.

Flow-control protocols assume that each station is aware of its distribution of bandwidth allocations across each span of the ring, termed its bandwidth profile, as well as the summation of this same information for all

stations on the ring, termed the ringlet's cumulative bandwidth profile. Each station has default allocated-bandwidth values to enable plug-and-play without management configuration.

NOTE—This subclause discusses the use of uniform bandwidth allocations. The uniform allocation is suboptimal, in that the MAC enforces behaviors consistent with a worst-case spatial (furthest distance) traffic distribution. Correct operation is thus ensured without assuming MAC client compliance, but uniform allocation techniques do not take advantage of potential spatial reuse.

A uniform-allocation single queue station S1 has allocated levels of classA and classB traffic, as illustrated in Figure 5.21-a. In this simplest of allocation examples, a uniform worst-case traffic flow is assumed. Uniform allocation makes no distinction between classA traffic sent from S1-to-S2 and classA traffic sent from S1-to-S4. Similarly, uniform allocation makes no distinction between classB traffic sent from S1-to-S3 and classB traffic sent from S1-to-S4. For each station, the summation of each bandwidth profile forms a (“Class allocations” column) cumulative bandwidth profile of client-visible bandwidth allocations. Within a single queue MAC, the advertised bandwidth (“Subclass allocations” column) for the classA service consists of subclassA0 allocated bandwidth, because a second transit queue is necessary to support subclassA1.

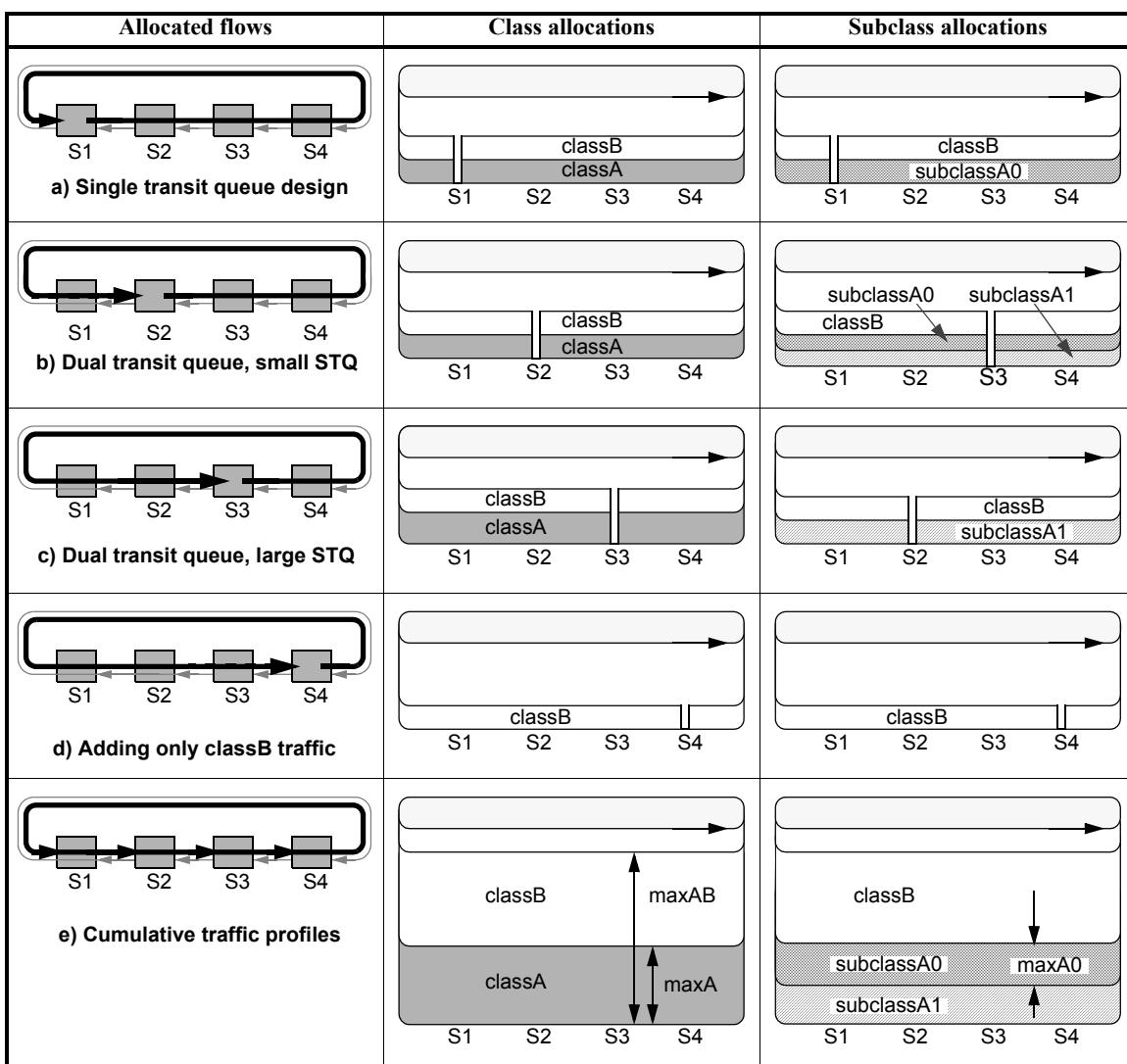


Figure 5.21—Single queue station S1 allocations

A uniform-allocation dual queue station S2 has allocated levels of classA and classB traffic, as illustrated in Figure 5.21-b. In this case, the levels of supportable subclassA1 bandwidths are proportional to the size of the secondary transit queue.

A uniform-allocation dual queue station S3 has allocated levels of classA and classB traffic, as illustrated in Figure 5.21-c. In this case, the levels of supportable subclassA1 bandwidths are sufficient to support the classA bandwidth allocations.

A uniform-allocation station S3 has allocated levels of classB traffic, as illustrated in Figure 5.21-d. In this case, allocations are unaffected by the presence and/or sizing of the secondary transit queue (STQ).

For uniform rate control, the cumulative class profile yields $\max A$ and $\max A0$, the worst-case allocated classA and subclassA0 segments, respectively, as illustrated in Figure 5.21-e. This information is used to rate-limit each station's classB and classC transmissions, with the intent of sustaining downstream classA transmissions. To ensure bandwidth guarantees, the cumulative $\max AB$ bandwidth, the sum of $\max A$ and $\max B$ bandwidths, must also be less than the capacity of an individual link.

5.12 Fairness

5.12.1 Equal-weighted fairness

Congestion occurs when the offered loads of the client (arrows from the top) exceed the capacity of the link (shaded boxes), as illustrated in Figure 5.22. A station that contributes to such congestion is not permitted to use more than its fair share of available capacity (see Clause 9) for the adding of fairness eligible traffic. This restriction prevents a station from using a disproportionate share of available capacity by virtue of its relative position on the ring. The rate restriction is enforced by a shaper within the MAC datapath sublayer.

Figure 5.22-a illustrates an example of unregulated access with upstream stations advantage. Stations S1 and S2 can utilize all available capacity during periods of congestion, preventing stations further downstream from adding opportunistic traffic.

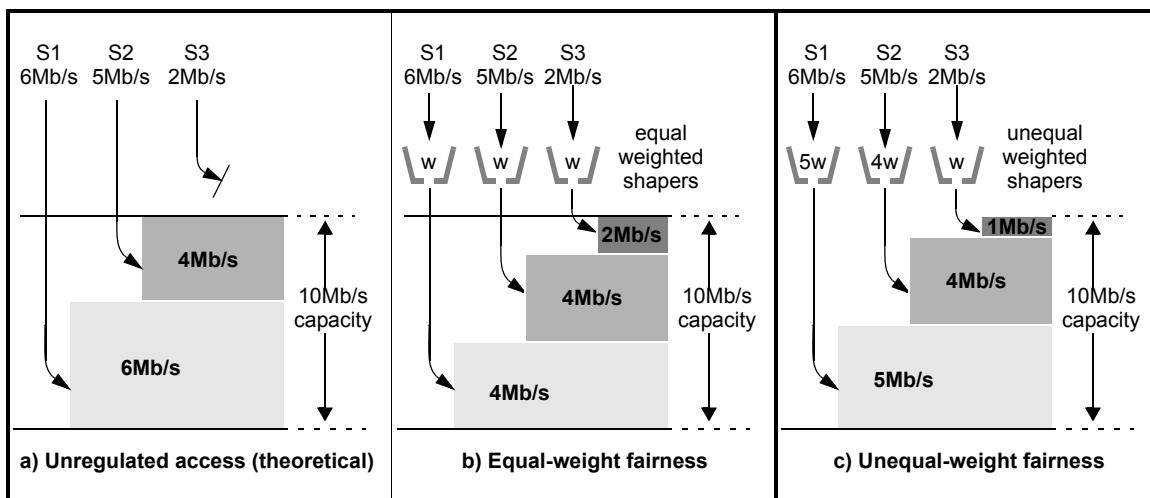


Figure 5.22—Unregulated vs. equal-weighted fair access on ringlet0

To eliminate the downstream disadvantage, each station applies fairness traffic shapers to restrict the use of available capacity during periods of congestion. With equal fairness weights, station traffic is distributed as illustrated in Figure 5.22-b.

Fairness is not necessarily based on equal bandwidths; it can be based on distinct fairness weights assigned to each station. The ratio of assigned bandwidth is approximately proportional to the ratios of the stations' fairness weights, as illustrated in Figure 5.22-c.

5.12.2 Fairness frame distribution

Shaping parameters for fairness eligible traffic are computed using a distributed fairness algorithm. The fairness algorithm for data traffic on a ringlet periodically sends fairness frames on the other ringlet, as illustrated in Figure 5.23. The basic fairness frame (called a single-choke fairness frame) is processed at each station and carries the identity of the station that is the most congested due to this station's transmissions, and the *fairRate* value reported by that station.

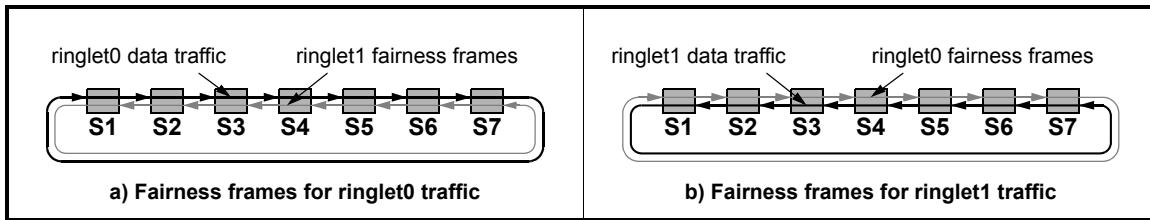


Figure 5.23—Fairness frames

5.12.3 Multi-choke fairness frame distribution

When enabled, a second type of fairness frame is broadcast periodically by all stations. This multi-choke fairness frame is passed to the fairness control unit for processing, and the relevant information is passed to the client to support multi-choke fairness. For efficiency, sending of the multi-choke fairness frame is automatically disabled when no multi-choke-capable stations are present.

5.13 Transit-queuing options

The terms cut-through and store-and-forward describe options (see 7.4) for the processing of transit traffic. Implementations can be based on either of these design options; store-and-forward is simpler; cut-through offers the possibility of improved performance. These implementation options differ in their processing of queued transit-queue frames, as illustrated by the black boxes within Figure 5.24.

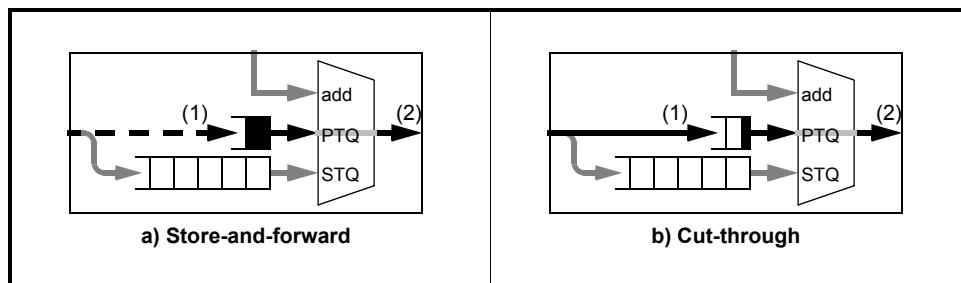


Figure 5.24—Transit-queuing options

In Figure 5.24, a frame transits through the PTQ of a dual queue station, with processing options as follows:

- a) Store-and-forward transiting frames are constrained as follows:
 - 1) A complete frame (including the payload) is accepted into the transit queue.
 - 2) A completely queued frame is selected for fixed-rate transmission to the downstream station.
- b) Cut-through transiting frames are constrained as follows:
 - 1) The header portion of a frame is accepted into the transit queue.
 - 2) A frame with a completely queued header is selected for fixed-rate transmission to the downstream station. The payload continues to be received while previously queued portions of the frame are transmitted.

NOTE—The operation of store-and-forward and cut-through options also applies to single queue stations or the secondary transit queue of a dual queue station.

The advantage of a store-and-forward implementation is simplicity: Frame processing can be deferred until after the entire frame has been received and its size is known. The advantage of a cut-through implementation is reduced latency: Frame forwarding can begin after only the header has been received, even though the data-payload size may be much larger and that size is unknown.

NOTE 1—Cut-through implementations also have to support store-and-forward behaviors, when transmission of the cut-through queue is temporarily blocked by a conflicting transmission or blocking flow-control indication.

NOTE 2—Although reception and transmission rates are closely matched, the output rate of the cut-through transit queue could be marginally higher than the input rate of this transit queue. In this situation, buffering of only the header could be insufficient to avoid underrun conditions.

5.14 Fault response methods

The dual-ring topology ensures that an alternate path between source station and destination station(s) is available after the failure of a single span or station. Fault response methods include protection and passthrough (see Clause 11). Protection involves either steering or wrapping and limits service interruption to 50 milliseconds. Passthrough (see 7.4.3) is optional.

Steering is supported by all stations. In the presence of a fault, steering stations direct protected frames to the ringlet that still has connectivity to the destination of the frame. In the presence of a fault, multicast and broadcast traffic is normally directed to both ringlets, so as to reach all stations on the ring.

Wrapping is an optional capability that is activated during a failure if a station supports wrapping and is configured to use wrapping as the protection method. Protected traffic is directed, at the point of failure, to the opposing ringlet. Wrapping is transparent to the source station, and connectivity to all stations is retained in the case of a single failure.

5.14.1 Fault response mechanisms

The effect of a station-detected failure depends on the failure condition and the state of other stations. The failure (see Figure 5.25-a) can become visible as a station-removal (see Figures 5.25-b, 5.25-c, and 5.25-d) or span-removal change to the operational topology (see Figure 5.25-e and Figure 5.25-f).

Stations detecting internal hardware/firmware corruption through consistency checks and heartbeat monitors have the option of entering a passthrough mode, as illustrated in Figure 5.25-b. A station in passthrough mode behaves as an active, otherwise-invisible repeater, maintaining a FIFO frame ordering.

Passthrough mode is an attractive alternative because communications with other stations are minimally affected by the resulting topology change. Furthermore, the remaining stations remain fully protected from other link and/or full-station failures.

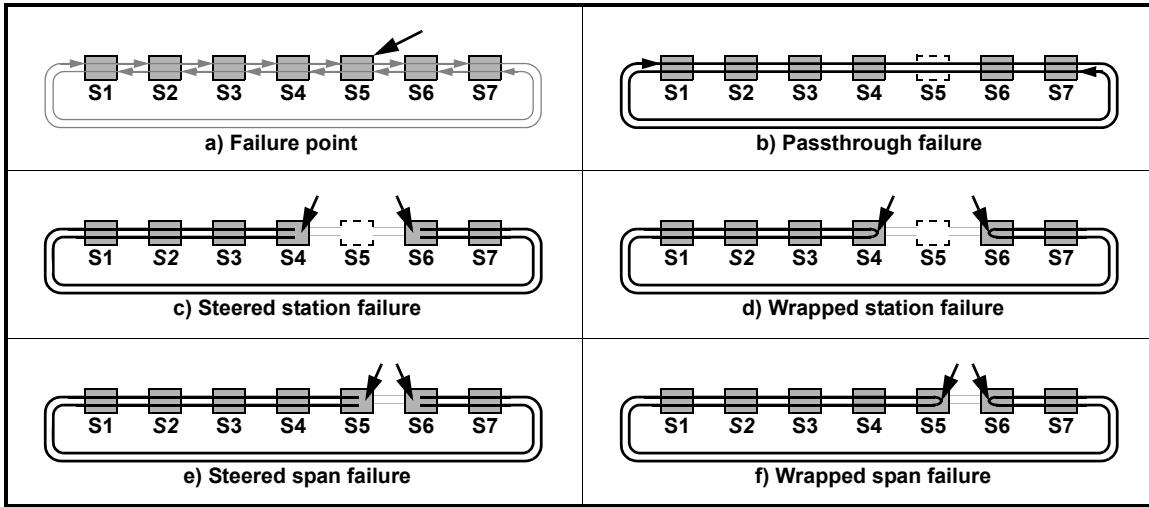


Figure 5.25—Fault response mechanisms

Passthrough mode is not possible when active repeating functionality is lost, due to a hardware failure, power loss, or card removal failure. Thus, recovery actions sometimes involve having adjacent stations isolate the failed station. Isolation can involve discarding endpoint traffic and steering traffic so that the endpoint is never reached (see Figure 5.25-c) or wrapping traffic at the edges (see Figure 5.25-d).

A span failure is also possible, due to laser-transmitter, laser-receiver, or fiber-cut failures. Fault isolation can involve discarding endpoint traffic and steering traffic so that the failed span is never reached (see Figure 5.25-e) or wrapping traffic on both sides of the failed span (see Figure 5.25-f).

5.14.2 Protection hierarchy

A single link may detect SIGNAL_FAILURE or SIGNAL_DEGRADE conditions. The MAC can be directed to generate a FORCED_SWITCH or MANUAL_SWITCH condition. The effect of the protection condition depends on the severity order of these protection conditions, listed below, where the top-through-bottom conditions are listed in order of most-through-least severity.

- FORCED_SWITCH—A management directive that forces a link to be deactivated.
- SIGNAL_FAILURE—A signal loss or major signal degradation that deactivates the link.
- SIGNAL_DEGRADE—A minor signal degradation that (if not overridden) deactivates the link.
- MANUAL_SWITCH—A management directive that (if not overridden) deactivates the link.
- WTR—After SIGNAL_FAILURE or SIGNAL_DEGRADE, the wait-to-restore timer improves stability in the presence of transient failures.

These protection conditions affect the topology, as illustrated in Figure 5.26. The left columns specify the P1-span and P2-span degradation conditions that lead to the protected topology figure in the right column. More severe P2 locations are not illustrated; these can be derived by interchanging the P1 and P2 span labels.

Only combinations of FORCED_SWITCH and SIGNAL_FAILURE can cause the ring to be severed at multiple span locations. Lower precedence conditions sever the most degraded span, but only when that most degraded span can be unambiguously identified.

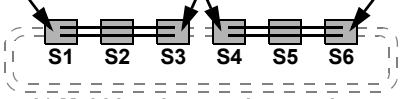
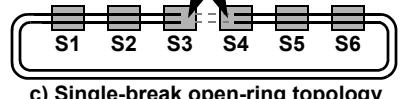
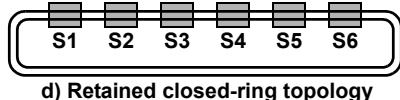
a) Protection conditions		
Condition at P1	Condition at P2	Resulting protection topology
$p1 > \text{SIGNAL_DEGRADE}$	$p2 > \text{SIGNAL_DEGRADE}$	 <p>b) Multi-break open-ring topology</p>
$p1 > p2$	$p2 \leq \text{SIGNAL_DEGRADE}$	 <p>c) Single-break open-ring topology</p>
$p1 == p2$	$p2 \leq \text{SIGNAL_DEGRADE}$	 <p>d) Retained closed-ring topology</p>

Figure 5.26—Protection topologies

5.14.3 Wrap then steer

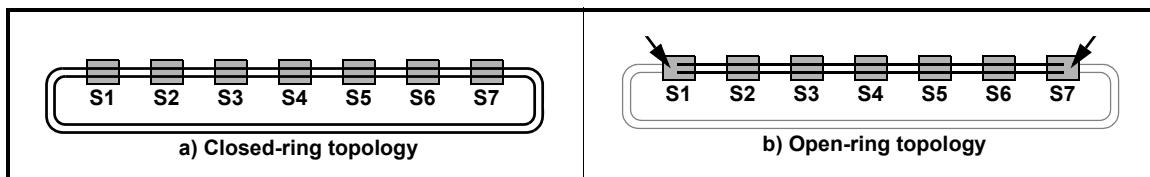
Implementations can pursue a two-phase protection strategy, completing step a) and then step b), as listed below. See Annex K for details.

- Wrap. Immediately wrap to reduce short-term frame losses.
- Steer. Eventually steer to improve long-term ringlet utilization efficiencies.

5.15 Topology discovery

The topology discovery algorithm (see Clause 11) describes rules for the broadcast of topology information contained in topology and protection (TP) frames on the ring. These frames contain information about the originating station and the configuration and capabilities making up the current topology image of that station. These frames are generated when the station becomes active on the ring, periodically, and on detection of a change in station or ring status.

The topology image represents a closed ring of stations (see Figure 5.27-a) or an open ring of stations (see Figure 5.27-b) resulting from a ring broken at one or more points.

**Figure 5.27—Closed-ring and open-ring topologies**

5.16 Frame ordering

5.16.1 Strict and relaxed transmissions

Three types of frame transmission are supported: permissive, relaxed, and strict. The adherence to the strict requirements (see 6.3 and 7.6.1) by the MAC is not only applicable to MACs servicing IEEE Std 802.1D-2004 and IEEE Std 802.1Q-2005 compliant clients (i.e., bridges), but also MACs servicing other clients (e.g., host, router). The requirements of strict transmissions are as follows:

- a) Duplication of user data frames is not permitted.
- b) Reordering of frames that have identical values of the following three parameters is not permitted:
 - 1) MAC address of the destination.
 - 2) MAC address of the source.
 - 3) Service class of the frame.

The complexity of supporting strict transmission is particularly burdensome during station or link failures. With relaxed and permissive mode, some reorder and/or duplication is possible, but frame losses can be decreased. With permissive mode, some reorder is also possible when the spatially aware sublayer (SAS) transitions between undirected and directed transmissions for a conversation.

The processing of strict, relaxed, and permissive transmissions is similar, but strict frames are discarded more frequently. The intent is to discard suspect frames, rather than risk the possibility of duplicating or misordering frames with distinct old-topology and new-topology heritages. Consistency checks and discard strategies handle passthrough (see Figure 5.25-b) and severed span (see Figure 5.25-c through Figure 5.25-f) failure scenarios.

5.17 Operations, administration, and maintenance (OAM)

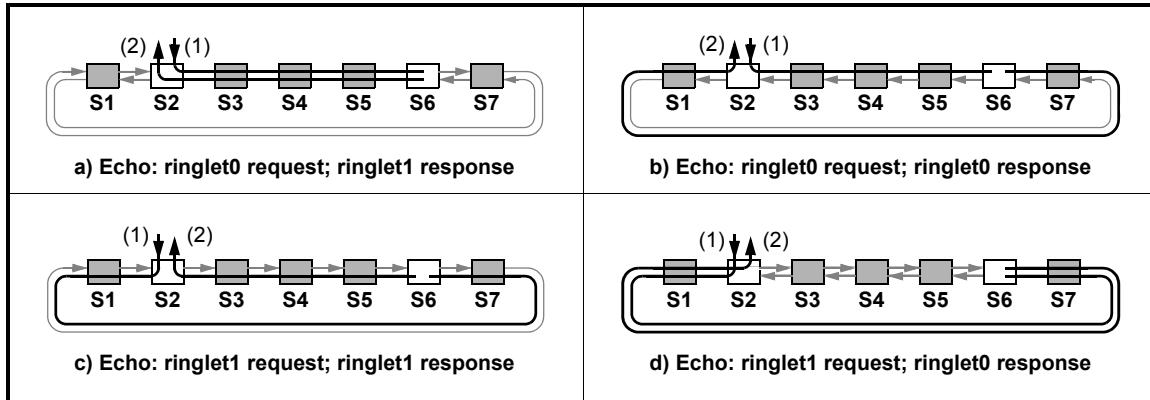
The operations, administration, and maintenance (OAM) entity of RPR (see Clause 12) provides a set of control functions and indications to support configuration management, fault management, and performance management.

Special control frames enable the detection and isolation of failures at the ring layer. These frames can be used either during service provisioning or continuously to minimize the correction time of abnormal operation.

5.17.1 Echo operations

An echo operation may be used to verify link integrity. The client can request an echo operation (called an echo request command) to a specified destination with the intent of checking the reachability of that destination. This echo request generates an echo response, and (depending on the echo-request specified routing) the response can return on the same ringlet or opposing ringlet. The client has the option of sending echo frames using classA, classB, or classC service classes.

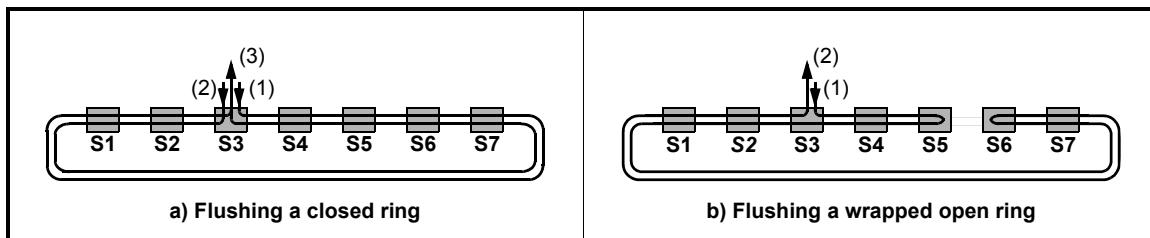
The path for the response (ringlet0 or ringlet1) depends on the arrival path and a code within the echo request, allowing the echo response to be returned on either ringlet, as illustrated in Figure 5.28. A code within the echo request directly or indirectly specifies the ringlet on which the response is returned, with the following selections: ringlet0, ringlet1, default, and reverse. For the default option, the echo-responder's ringlet-selection mechanism (see 7.7.1) selects the returned path; for the reverse option, the response returns on the ringlet not used by the echo request, as illustrated in Figure 5.28-a and Figure 5.28-b.

**Figure 5.28—Echo operation, ringlet selection possibilities**

5.17.2 Flushing previously sourced traffic

A flush is expected to be used when changing the ringlet selection algorithm, when revised ringlet selection protocols are necessary to access all stations (for steer-protection), or to improve bandwidth utilization (for wrap protection).

A flush ensures the removal of related (i.e., previously sourced from the same station) traffic, by sending a marker behind related frames. The return of that marker thus confirms the removal of related data traffic. The transmission path of a flush frame is illustrated in Figure 5.29. Within the selected traffic class, the marker maintains its position behind related traffic. Nothing forces the removal of previously sourced traffic, but removals occur as a defined side effect of the frame-transfer protocols.

**Figure 5.29—Flushing previously sourced traffic**

Two flushes are normally necessary to confirm the removal of related traffic on a closed ring, as illustrated in Figure 5.29-a. On an open wrapped ring, one flush is sufficient to confirm the removal of related traffic, as illustrated in Figure 5.29-b.

5.17.3 Management information base (MIB)

This standard defines a management information base (MIB) for use with network management applications using the simple network management protocol (SNMP) for managing RPR interfaces (see Annex D).

The MIB defines the set of attributes that are supported and operations that may be performed on an RPR MAC by network management. For example, the MIB can be queried to discover a variety of MAC-resident management information, such as the topology of the ring.

5.18 Spatially aware sublayer

SAS is a sublayer of the MAC datapath sublayer. SAS functionality is optional and provides spatial reuse for frame transmissions other than local unicast (e.g., through bridges on the ring), by using directed transmissions or multicast scoping. If SAS functionality is not implemented, *enableSas* (see 6.2.4) is always FALSE, resulting in no spatial awareness for frame transmissions other than local unicast.

5.19 Protected inter-ring connection

Protected inter-ring connection (PIRC) is an optional sublayer of the MAC datapath sublayer. PIRC functionality provides 50 ms protection of traffic between interconnected rings through dual-station homing. The PIRC sublayer is applicable only to interconnect stations. Non-interconnect stations require no changes in order for PIRC to function correctly.

6. Medium access control (MAC) service and reference model

6.1 Overview

This clause provides an overview of the MAC sublayer and the reconciliation sublayer (see shaded portions of Figure 6.1); specifies the services provided by the MAC sublayer, including the MAC control sublayer and the MAC datapath sublayer; and provides a reference model for the MAC sublayer. Higher layer clients can include the logical link control (LLC) sublayer, bridge relay entity, or other users of ISO/IEC LAN international standard MAC services. The services are described in an abstract way and do not imply any particular implementations or any exposed interfaces. There is not necessarily a one-to-one correspondence between the primitives and formal procedures and the interfaces in any particular implementation.

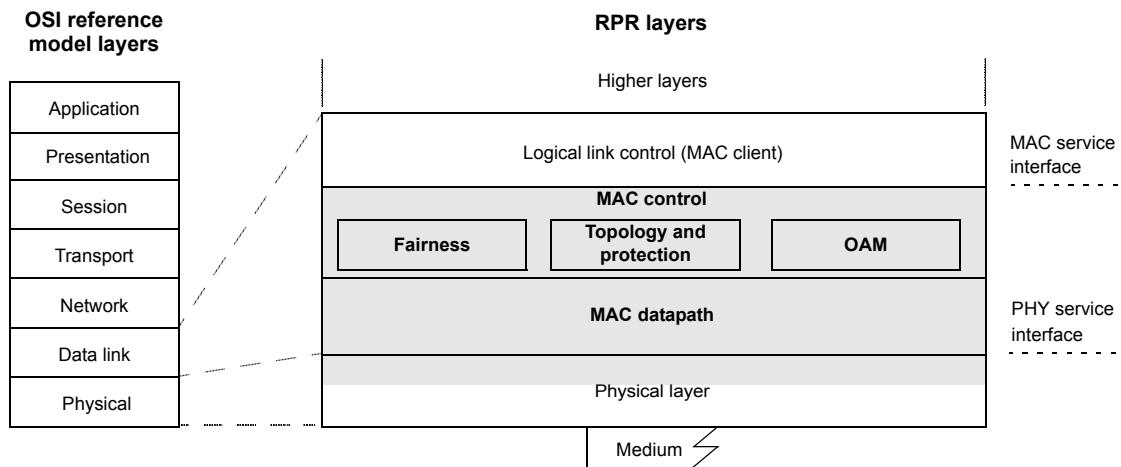


Figure 6.1—RPR service and reference model relationship to the ISO/IEC OSI reference model

6.2 Terminology and variables

This clause defines the following terms and variables for use throughout this standard:

6.2.1 *copyBadFcs*: A boolean variable determining if frames with bad *fcs* values are copied to the client. A value of TRUE causes all data frames destined to the client to be copied to the client regardless of the validity of the *fcs* field. A value of FALSE causes the MAC to not copy any data frames with invalid *fcs* values to the client. The method of configuring *copyBadFcs* is not standardized.

6.2.2 *enablePirc*: A boolean variable determining whether PIRC functionality is enabled. A value of FALSE causes all data frames to bypass PIRC and ignores all received PIRC status messages.

6.2.3 *enableSas*: A boolean variable determining whether SAS processing is enabled. A value of TRUE enables processing of the *request_sas* parameter of the MA_DATA.request primitive, and learning of address associations in SAS receive state tables. A value of FALSE causes all data frames to bypass SAS. The method of configuring *enableSas* is not standardized.

6.2.4 *optionSasMcastScope*: A boolean variable that controls the optional SAS multicast scoping functionality. A value of TRUE indicates that SAS multicast scoping is active. A value of FALSE indicates that SAS multicast scoping is not active. A value of TRUE can be set only if *enableSas* is set to TRUE. The default value is FALSE. The method of configuring *optionSasMcastScope* is not standardized.

6.2.5 *remapRxAddrs*: A boolean variable that controls the mapping of extended addresses in the MAC receive path. A value of TRUE means the MAC receive path sets source_address to source_address_extended, and destination_address to destination_address_extended. A value of FALSE indicates that the MAC receive path does not modify the address parameters. The default value is TRUE. The method of configuring *remapRxAddrs* is not standardized.

6.2.6 *ringlet0*: The datapath that receives frames from the west PHY and transmits or retransmits frames on the east PHY.

6.2.7 *ringlet1*: The datapath that receives frames from the east PHY and transmits or retransmits frames on the west PHY.

6.2.8 *stdCleave*: A boolean variable indicating whether a SAS-capable station should use bidirectional flooding and the standard cleave point calculation. A value of TRUE causes all flooded data frames to use the standard cleave point calculation per 7.7.2.1. A value of FALSE indicates that neither the standard cleave point calculation nor bidirectional flooding need to be applied by the MAC. The method of configuring *stdCleave* is not standardized.

6.2.9 *transmissionControl*: Determines the transmission service provided by the MAC.

STRICT—The MAC uses the strict transmission service for all frames unless the client sets the strict_order parameter to FALSE; in which case, the frame uses the relaxed transmission service.

RELAXED—The MAC uses the relaxed transmission service for all frames unless the client sets the strict_order parameter to TRUE; in which case, the frame uses the strict transmission service.

PERMISSIVE—The MAC uses the permissive transmission service for all frames unless the client sets the strict_order parameter to TRUE; in which case, the frame uses the strict transmission service. PERMISSIVE can be set only if *enableSas* is set to TRUE.

The default is STRICT. The method of configuring *transmissionControl* is not standardized.

6.3 Overview of MAC services

The services provided by the MAC sublayer allow the following:

- a) The local client layer in an end station to exchange data with peer client layer entities.
- b) The local client layer in an end station to exchange parameters with the local MAC entity.
- c) The relay entity in a bridge to exchange data with local MAC entities in the bridge.

6.3.1 Service types

The RPR MAC provides three types of frame transmission service, described below. (See also 7.6.1, Table 7.20, and Table 7.29.)

- a) Strict: Adheres to the 802.1 frame reorder, duplication, and loss requirements, as follows:
 - 1) There is no guarantee that all service data units (SDUs) are delivered.
 - 2) Reordering of frames with a given user priority for a given combination of destination address and source address is not permitted.
 - 3) Duplication of user data frames is not permitted.

In general, strict transmission requires special processing only during station or link failure events.

- b) Relaxed: Adheres to the 802.1 frame reorder, duplication, and loss requirements, except while recovering from failures on the ring. During ring protection events, a minimal amount of reorder and/or duplication can be encountered; however, the chance of delivery is increased.

Scenarios where a negligible amount of frame reorder or duplication can occur include the following:

- 1) After ring (link or station) restoration events.
 - 2) Topology and status database of stations on the ring are not synchronized.
 - 3) Station failure resulting in passthrough behavior. That is, frames are sent through the transit path without *ttl* decrement or any other frame processing/stripping rules being applied.
 - 4) Compound ring (link or station) failures resulting in segmented open rings.
 - 5) Rapid cascading ring failures.
- c) Permissive: Permissive has the same attributes as the relaxed transmission service, except that some reordering can occur during transitions between undirected and directed transmission for a conversation.

NOTE—Relaxed and permissive modes are more efficient (i.e., fewer frames are discarded) and are used when strict mode is not required.

The MAC sublayer presents a service interface for the exchange of MAC service data units between MAC client entities. The MAC service interface supports service classes denoted classA, classB, and classC (see 7.3 and 9.6.4).

For all service classes, the MAC service interface provides per-ringlet indications to the MAC client that indicate whether traffic can or cannot currently be accepted. For service class classC, the MAC service interface of a station lying within a congestion domain also provides the number of hops to the head of the congestion domain in which that station lies. Each service class is rate controlled to prevent the client from transmitting more traffic than was allocated by station management or allowed by fairness, as applicable. When a client does not obey the indication of the MAC with regard to flow control on a given service class, the MAC is free to ensure correct ring operation by either dropping the offending client requests, or by head-of-line blocking until the request can be properly handled.

The allocation, advertisement, and shaping of allocated bandwidth is done on a per-class, per-ringlet basis.

The MAC does not provision bandwidth. This is a higher level management function beyond the scope of this standard.

The information that flows between the MAC and the client layer is specified formally in the primitives (see 6.4).

6.3.2 Service classes

The RPR MAC provides three classes of frame transmission service (see 7.3).

ClassA service provides an allocated, guaranteed data rate and a low end-to-end delay and jitter bound. Within this class, the MAC uses two internal subclasses, subclassA0 for reserved bandwidth and subclassA1 for reclaimable bandwidth (see 7.3). ClassA traffic is not subject to the fairness algorithm at ingress to the ring or when transiting through the ring.

ClassA traffic has precedence over classB and classC traffic at ingress to the ring, and during transit through the ring (for dual queue stations). ClassA traffic moves through the primary transit path in each station as it propagates around the ring.

Description and operation of the primary and secondary transit paths, and descriptions of single queue and dual queue models, are provided in 7.4.

ClassB service provides an allocated, guaranteed data rate, and bounded end-to-end delay and jitter for the traffic within the allocated rate and access to additional best effort data transmission that is not allocated, guaranteed, or bounded, and is subject to the fairness algorithm. Within this class, the MAC uses fairness eligibility markings to differentiate the committed information rate portion of classB (classB-CIR) and the excess information rate portion of classB (classB-EIR).

ClassB traffic (including classB-EIR) has precedence over classC traffic at ingress to the ring. In a single queue implementation, all classB traffic moves through the primary transit path. In a dual queue implementation, classB traffic moves through the secondary transit path, regardless of whether the frame is marked fairness eligible.

ClassC service provides a best-effort traffic service with no allocated or guaranteed data rate and no bounds on end-to-end delay or jitter. ClassC traffic is always subject to the fairness algorithm.

In a single queue implementation, classC traffic moves through the primary transit path. In a dual queue implementation, classC traffic moves through the secondary transit path.

6.4 MAC services to the client layer

The services of a layer or sublayer are the set of capabilities that it offers to the next higher (sub)layer. The services specified in this standard are described by abstract service primitives and parameters that characterize each service. This definition of a service is independent of any particular implementation.

The following four service primitives are defined for the non-bridge client interfaces and shall be implemented. (For additional interface requirements for bridge clients, see Annex F.)

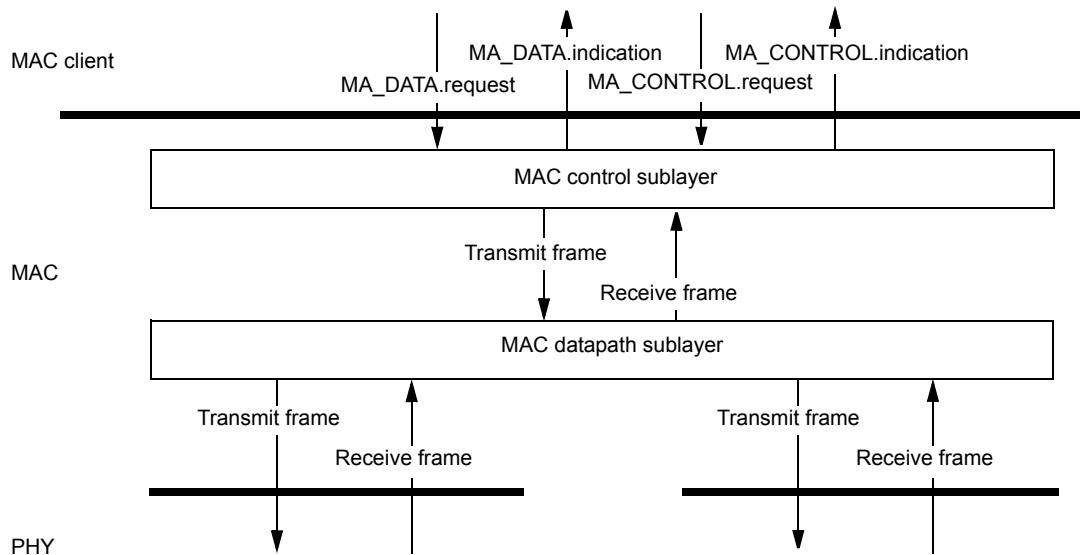
- MA_DATA.request
- MA_DATA.indication
- MA_CONTROL.request
- MA_CONTROL.indication

The service primitives are illustrated in context in Figure 6.2.

6.4.1 MA_DATA.request

6.4.1.1 Function

The MA_DATA.request primitive defines the transfer of data from a MAC client entity to a single peer entity, or to multiple peer entities in the case of group addresses.

**Figure 6.2—MAC service model**

6.4.1.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```

MA_DATA.request
(
    destination_address,
    source_address,          // optional
    mac_service_data_unit,
    frame_check_sequence,   // optional
    service_class,
    ringlet_id,              // optional
    mac_protection,          // optional
    mark_fe,                 // optional
    strict_order,             // optional
    destination_address_extended, // optional
    source_address_extended,  // optional
    flooding_form,           // optional
    request_sas              // optional
)
  
```

The parameters of the MA_DATA.request are described as follows:

destination_address

Specifies either an individual or group MAC address, different from the local MAC address, to be used to create the *da* (destination MAC address) field or *daExtended* (destination MAC address, extended) field of the transmitted frame (see 6.4.1.5 and 9.2.2.3).

source_address

If present, and different from the station's MAC address, specifies an individual MAC address to be used to create the *saExtended* (source MAC address, extended) field of the transmitted frame (see 9.2.2.9), and to determine basic or extended data frame format (see 6.4.1.5 and 7.7.1). The *source_address* parameter is ignored if the *source_address_extended* parameter is provided.

mac_service_data_unit

Provides the payload to be delivered. Specifically, the *mac_service_data_unit* parameter specifies the MAC service data unit to be transmitted by the MAC sublayer entity in the *serviceDataUnit* field of the transmitted frame (see 9.2.2.11). Sufficient information is associated with *mac_service_data_unit* for the MAC sublayer entity to determine the length of the data unit.

frame_check_sequence

Provides the value of the *fcs* field of the transmitted frame (see 9.2.2.12). If the *frame_check_sequence* parameter is omitted, the MAC calculates the *fcs* value (see E.2).

service_class

Indicates the class of service requested by the MAC client, as described in Table 6.1, which is used by the MAC entity to select the value of the *sc* field (see 9.6.4), and to indicate the requested MAC treatment of the transmitted frame (see 7.7.4).

Table 6.1—service_class values

service_class value	Corresponding <i>sc</i> value	Description
SC_CLASS_A	CLASS_A0 or CLASS_A1	classA service class
SC_CLASS_B	CLASS_B	classB service class
SC_CLASS_C	CLASS_C	classC service class

ringlet_id

Indicates the ringlet choice of the client, as described in Table 6.2 (see also 7.7.1), which is used by the MAC entity to select the value of the *ri* (ringlet identifier) field (see 9.6.1) of the transmitted frame (before any protection-based change).

Table 6.2—ringlet_id values

ringlet_id value	Corresponding <i>ri</i> value	Description
RI_0	RINGLET_0	ringlet0 preferred
RI_1	RINGLET_1	ringlet1 preferred
RI_DEFAULT	—	default ringlet
(null)	—	default ringlet

mac_protection

Indicates a choice of whether the MAC provides protection for the frame (see 7.7.1).

TRUE—The MAC provides protection for the frame.

FALSE—The MAC does not provide protection for the frame.

(null)—The default value is TRUE.

mark_fe

Indicates a request to mark and treat a frame as fairness eligible regardless of how it would have been marked or treated otherwise, guiding the MAC entity on how to set the *fe* (fairness eligible) field (see 9.6.2). The *mark_fe* parameter is valid only for requests that include a request of classB for the *service_class* and is ignored for all other requests. This is provided to allow a client to choose which of the classB marked service data units presented to the MAC are considered fairness eligible, such as when handling multiple flows of traffic, each with their own client-based rate allocations (see 7.7.4).

TRUE—The MAC forces the (classB) frame to be fairness eligible.

FALSE—The MAC determines whether the frame is fairness eligible based on *sendB*.

(null)—The default value is FALSE.

strict_order

Indicates a request to mark and treat a data frame as strict, which is used by the MAC entity to select the value of the *so* (strict order) field (see 9.7.4).

TRUE—The MAC sets *so* to 1.

FALSE—The MAC sets *so* to 0.

(null)—The default value is TRUE when the *transmissionControl* variable is set to STRICT, otherwise FALSE.

destination_address_extended

Specifies either an individual or group MAC address, different from the local MAC address, to be used to create the *daExtended* field of the transmitted frame (see 9.2.2.8). If the destination_address_extended parameter is provided, the MAC uses the extended data frame format (see 7.7.1). Whenever the destination_address_extended parameter is provided, the source_address_extended parameter is also required to be provided (see 6.4.1.5), and the flooding_form parameter is also required to be provided. Use of this parameter's effect on bridging is discussed in F.1.7. This parameter is ignored when SAS processing is selected.

source_address_extended

Specifies an individual MAC address, to be used to create the *saExtended* field of the transmitted frame (see 9.2.2.9). If the source_address_extended parameter is provided, the MAC uses the extended data frame format (see 7.7.1). Whenever the source_address_extended parameter is provided, the source_address parameter is ignored, the destination_address_extended parameter is also required to be provided (see 6.4.1.5), and the flooding_form parameter is also required to be provided. Use of this parameter's effect on bridging is discussed in F.1.7. This parameter is ignored when SAS processing is selected.

flooding_form

Indicates a request to use a particular flooding form (see 7.7.1), as specified in Table 6.3, and to set the *fi* (flooding indication) field accordingly (see 9.7.2). A value of FI_NONE directs the MAC to not flood the frame. A value other than FI_NONE directs the MAC to flood the frame regardless of the determination that would have been made otherwise by ringlet selection, and to use the flooding form selected for the frame. A value of FI_NONE is ignored for frames whose destination address has the group bit set. If flooding_form is omitted, the MAC uses its default flooding behavior. The flooding_form parameter is required to be provided if the destination_address_extended and source_address_extended parameters are provided. Use of this parameter's effect on bridging is discussed in F.1.7. This parameter is ignored when SAS processing is selected and a SAS database (SDB) entry is found, or when *stdCleave* is enabled.

Table 6.3—flooding_form values

flooding_form value	Corresponding <i>fi</i> value	Description
FLOOD_NONE	FI_NONE	no flood
FLOOD_UNIDIR	FI_UNIDIR	unidirectional flood
FLOOD_BIDIR	FI_BIDIR	bidirectional flood
(null)	default flooding behavior selected by the MAC	

request_sas

Indicates a request to provide SAS processing. If the *enableSas* variable is set to TRUE (see 6.2.3), this parameter controls the selection of SAS processing. Otherwise, SAS processing is disabled and this parameter is ignored.

TRUE—Request SAS processing for this frame.

FALSE—Bypass SAS for this frame.
(null)—The default value is the setting of the *enableSas* variable.

6.4.1.3 When generated

The MA_DATA.request primitive is invoked by the client entity whenever data are to be transferred to a peer entity or entities.

6.4.1.4 Effect of receipt

The receipt of the MA_DATA.request primitive causes the MAC entity to create a data frame or an extended data frame, fill in the fields whose values are given or determined by the parameters of this request, and pass the properly formed frame to the transmit state machines by placing it on Q_TX_PIRC (see Figure 6.2), for transfer to the peer MAC sublayer entity or entities.

6.4.1.5 Additional comments

The MAC does not reflect frames back to the client. If a client issues an MA_DATA.request primitive with a destination_address value equal to its local MAC address, the request is rejected.

The MAC does not accept requests from the client when the appropriate send indication is not present. The details of this interaction are implementation specific.

For frames not processed by SAS, the combinations of source_address and source_address_extended are summarized in Table 6.4. In the case of any ambiguity between this summary and the RingletSelection and SasTransmit state machines (see 7.7.1 and 14.5.3.1), the state machines shall take precedence.

Table 6.4—Source address value combinations

source_address_extended	source_address	Resulting frame actions
not provided	not provided	// basic data frame sa = myMacAddress;
	= myMacAddress	// basic data frame sa = myMacAddress;
	≠ myMacAddress	// extended data frame sa = myMacAddress; saExtended = source_address; da = destination_address; daExtended = destination_address;
provided	—	// extended data frame sa = myMacAddress; saExtended = source_address_extended; da = destination_address; daExtended = destination_address_extended;

6.4.2 MA_DATA.indication

6.4.2.1 Function

The MA_DATA.indication primitive defines the transfer of data from the MAC sublayer entity to the MAC client entity.

6.4.2.2 Semantics of the service primitive

The semantics of the primitive are as follows:

```
MA_DATA.indication
(
    destination_address,
    source_address,
    mac_service_data_unit,
    frame_check_sequence,
    reception_status,
    service_class,
    ringlet_id,
    fairness_eligible,
    strict_order,
    extended_frame,
    destination_address_extended,
    source_address_extended
)
```

The parameters of the MA_DATA.indication are described below.

destination_address

Indicates the value of the *da* (see 9.2.2.3) or *daExtended* (see 9.2.2.8) field of the incoming frame, as described in 14.5.3.1.

source_address

Indicates the value of the *sa* (see 9.2.2.4) or *saExtended* (see 9.2.2.9) field of the incoming frame, as described in 14.5.3.1.

mac_service_data_unit

Provides the MAC service data unit, as specified by the *serviceDataUnit* field of the incoming frame (see 9.2.2.11). There is sufficient information associated with *mac_service_data_unit* for the MAC client entity to determine the length of the data unit.

frame_check_sequence

Indicates the value of the *fcs* field in the frame header (see 9.2.2.12).

reception_status

Indicates the status of the received frame to the MAC client entity, as described in Table 6.5. *reception_status* can take the value of RECEIVE_FCS_ERROR only if the variable *copyBadFcs* is set to TRUE.

Table 6.5—reception_status values

reception_status value	Description
RECEIVE_OK	The frame had no errors.
RECEIVE_FCS_ERROR	The frame had an FCS error (either invalid FCS or stomped FCS).

service_class

Indicates the service class at which the frame was sent (see 7.17.3), using the values provided in Table 6.1.

ringlet_id

Indicates the setting of the *ri* bit in the frame header (see 9.6.1), using the values provided in Table 6.2.

`fairness_eligible`

Indicates the setting of the *fe* bit in the frame header (see 9.6.2).

`strict_order`

Indicates the setting of the *so* bit in the frame header (see 9.7.4).

`extended_frame`

Indicates the setting of the *ef* bit in the frame header (see 9.7.1).

`destination_address_extended`

If `extended_frame` is TRUE, indicates the value of the *daExtended* field of the incoming frame (see 9.2.2.8).

`source_address_extended`

If `extended_frame` is TRUE, indicates the value of the *saExtended* field of the incoming frame (see 9.2.2.9).

6.4.2.3 When generated

The `MA_DATA.indication` is passed from the MAC sublayer entity (through the MAC control sublayer) to the MAC client entity or entities to indicate the arrival of a frame to the local MAC sublayer entity that is destined for the MAC client. Such frames are reported only if they are validly formed, and their destination address designates the local MAC entity (local station address, group address, or flooded). A client's handling of a received frame, including one with a `reception_status` value other than `RECEIVE_OK`, is beyond the scope of this standard.

6.4.2.4 Effect of receipt

The effect of receipt of the `MA_DATA.indication` primitive by the MAC client is unspecified. There is no requirement or implication that the MAC client processes any of the parameters.

6.4.2.5 Additional comments

The MAC does not reflect frames back to the client. If a MAC receives a frame with a *sa* value of the local MAC address, it does not cause an `MA_DATA.indication` primitive to be sent to the originating client (see 7.6.3).

6.4.3 MA_CONTROL.request

The `MA_CONTROL.request` primitive defines the transfer of control requests from the MAC client to the MAC control sublayer. This primitive does not provide a direct means for a client to transmit a control frame from the local MAC onto any ringlet, although control frames (for example, echo or flush) can be indirectly generated as a result of this request.

If a client level entity requires setting or getting status or configuration parameters of the MAC, it directs or queries the MLME to set or obtain this information. Examples include topology and status database, single queue versus dual queue, manual switch, and forced switch.

6.4.3.1 Function

The `MA_CONTROL.request` primitive defines the transfer of control commands from a MAC client entity to the local MAC control sublayer entity.

6.4.3.2 Semantics of the service primitive

The semantics of the `MA_CONTROL.request` primitive are as follows:

```
MA_CONTROL.request
(
    opcode,
    request_operand_list
)
```

The opcode parameter, described in Table 6.6, indicates the control operation requested by the MAC client entity.

Table 6.6—Control request opcodes

opcode value	Meaning	Operands	Specified in
OAM_ECHO_REQ	Request to transmit an echo request frame	echo request payload and parameters	12.4.1
OAM_FLUSH_REQ	Request to transmit a flush frame	flush payload and parameters	12.4.3
OAM_ORG_REQ	Request to transmit an organization-specific OAM frame	organization-specific payload and parameters	12.4.5
OAM_SAS_NOTIFY	Notification of a network topology change event that affects SAS forwarding.	SAS notify specific payload and parameters	12.4.7

6.4.3.3 When generated

The MA_CONTROL.request primitive is generated by a MAC client whenever it wishes to use the services of the MAC control sublayer entity.

6.4.3.4 Effect of receipt

The effect of receipt of the MA_CONTROL.request primitive by the MAC control sublayer is opcode-specific.

6.4.4 MA_CONTROL.indication

6.4.4.1 Function

The MA_CONTROL.indication primitive defines the transfer of control status indications from the MAC control sublayer to the MAC client.

6.4.4.2 Semantics of the service primitive

The semantics of the MA_CONTROL.indication primitive are as follows:

```
MA_CONTROL.indication
(
    opcode,
    indication_operand_list
)
```

The elements of the indication_operand_list parameter are specific to each opcode parameter and specified in Table 6.7.

Table 6.7—Control indication opcodes

opcode value	Meaning	Operands	Specified in
OAM_ECHO_IND	Receipt of an echo reply frame	echo payload and parameters	12.4.2
OAM_FLUSH_IND	Receipt of a flush frame	flush payload and parameters	12.4.4
OAM_ORG_IND	Receipt of an organization-specific OAM frame	organization-specific payload and parameters	12.4.6
OAM_SAS_NOTIFY_IND	Receipt of a SAS notify specific OAM frame	SAS notify payload and parameters	12.5
TOPO_CHANGE	Topology change	topology and status database	11.5
PROT_CHANGE	Protection change	topology and status database	11.5
SENDA	<i>sendA</i> change	TRUE/FALSE, ringlet_id	7.5.2
SENDB	<i>sendB</i> change	TRUE/FALSE, ringlet_id	7.5.2
SENC	<i>sendC</i> change	<i>hopsToCongestion</i> , ringlet_id	7.5.2
SINGLE_CHOKE_IND	Expiration of <i>agingInterval</i> (when <i>singleChokeIndOption</i> is TRUE)	<i>allowedRate</i> , <i>allowedRateCongested</i> , <i>hopsToCongestion</i> , ringlet_id	10.4.2
MULTI_CHOKE_IND	Receipt of multi-choke fairness frame (MCFF)	<i>frame.fairRate</i> , <i>frame.saCompact</i> , <i>frame.ttl</i> , <i>frame.ri</i>	10.4.8

6.4.4.3 When generated

The MA_CONTROL.indication primitive is generated by the MAC control sublayer under conditions specific to each MAC control operation.

6.4.4.4 Effect of receipt

The effect of receipt of the MA_CONTROL.indication primitive by the MAC client is unspecified.

6.4.4.5 Additional comments

The client's usage of the indications provided by the MA_CONTROL.indication is beyond the scope of this standard; however they are made available to allow a client to perform more complex actions beyond the capability of the MAC, for example, implementing a more efficient frame scheduling algorithm based on the knowledge of choke points reported via the MULTI_CHOKE_IND.

The indications and the operands of the indications are not necessarily physically passed to the client. They can be merely made available to the client. The means of indicating or otherwise making available is a local implementation detail.

6.5 MAC compliance test points

There are three types of test points, described in Table 6.8, at which a system-level implementation of this standard can claim compliance.

Table 6.8—Compliance test point types

Test point type	Standardization of implementation	Compliance
logical test point	Implementation is logically defined	No physical compliance is specified or testable, but abstract compliance can be required
standard test point	Implementation can be standardized, but is not required to be testable in a complete system	Physical compliance can be testable for components, but is not necessarily available in a complete system
conformance test point	Implementation is standardized	Physical compliance is specified and tested in a complete system

This standard has three test points: the client interface, the reconciliation sublayer to physical layer interface, and the physical interface, as illustrated in Figure 6.3.

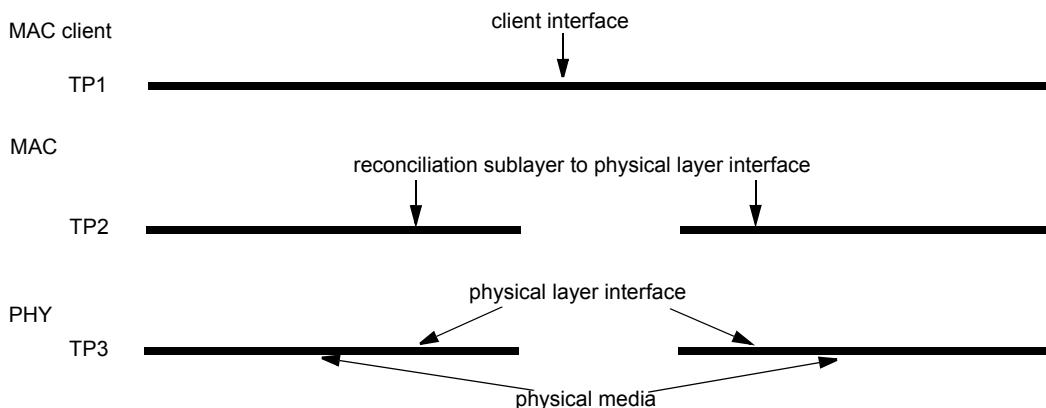


Figure 6.3—MAC test points

The client interface, shown as TP1 in Figure 6.3, is a logical test point that provides only a logical description of the service primitives used to interface with the client. This interface comprises the MA_DATA.request, MA_DATA.indication, MA_CONTROL.request, and MA_CONTROL.indication service primitives. This standard does not define any physical conformance for this interface.

The reconciliation sublayer to physical layer interface, shown as TP2 in Figure 6.3, is a logical test point that provides primarily a logical description of the service primitives used to interface between the reconciliation sublayer and the physical layer. Detailed logical descriptions of the mapping of the reconciliation sublayer primitives to the standardized electrical interfaces specified in this (or other) standards are provided. Recommendations are provided for optional physical mappings, but no standard physical mapping of this interface is required. This interface comprises the PHY_DATA.request, PHY_DATA.indication, PHY_LINK_STATUS.indication, and PHY_READY.indication service primitives. The physical implementation of this interface is of concern to component providers and system implementers and does not affect system conformance.

The physical layer interface, shown as TP3 in Figure 6.3, is the only conformance test point for this standard. This test point provides the detailed mapping of how the frame appears on the physical media¹⁵ types supported by the MAC. The physical media whose reconciliation sublayers are specified in this standard are PacketPHY and SONET/SDH.

6.5.1 1 Gb/s PacketPHY

Conforms with the physical coding sublayer (PCS) and physical medium attachment (PMA) (see Annex B).

6.5.2 10 Gb/s PacketPHY

Conforms with the physical coding sublayer (PCS) and physical medium attachment (PMA) (see Annex B).

6.5.3 SONET/SDH

Conforms with the physical layer, framing specifications, and virtual concatenation specifications (see Annex C).

6.6 MAC reference model

As illustrated in Figure 6.1, the MAC comprises the MAC control sublayer and the MAC datapath sublayer. The MAC datapath sublayer comprises the protected inter-ring connection sublayer, spatially aware sublayer, the ringlet selection entity, and the datapaths for the two ringlets. These components and their interconnections are illustrated in Figure 6.4 within the context of a single station view of the MAC architecture.

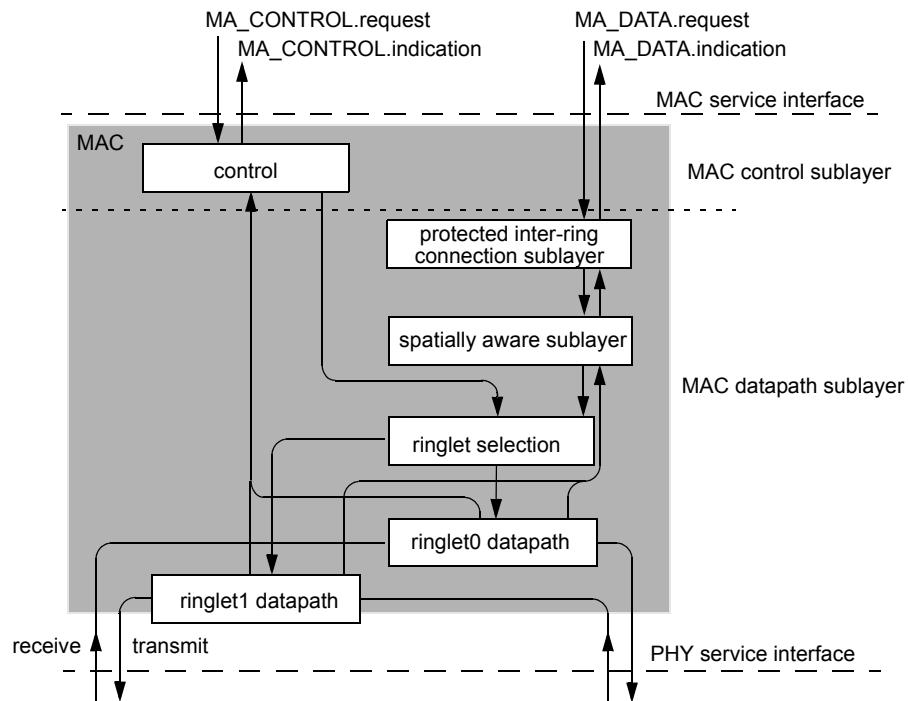


Figure 6.4—Single station view of MAC architecture

¹⁵Note that the “physical media” can be a virtually concatenated SONET/SDH channel, i.e., not truly a physical medium.

6.6.1 MAC control sublayer

The MAC control sublayer, illustrated in Figure 6.5, supports control activities necessary to maintain the state of the MAC and datapath activities not identified with a particular ringlet. The control activities are distributed among stations on the ring in order to survive any single point of failure. Control entities in a station communicate with peer control entities in other stations using the services of the MAC datapath sublayer. The activities of the MAC control sublayer include the following:

- control service interface
- fairness algorithm and protocol
- protection database and protocol
- topology database and protocol
- operations, administration, and maintenance (OAM) functionalities
- control frame send and receive

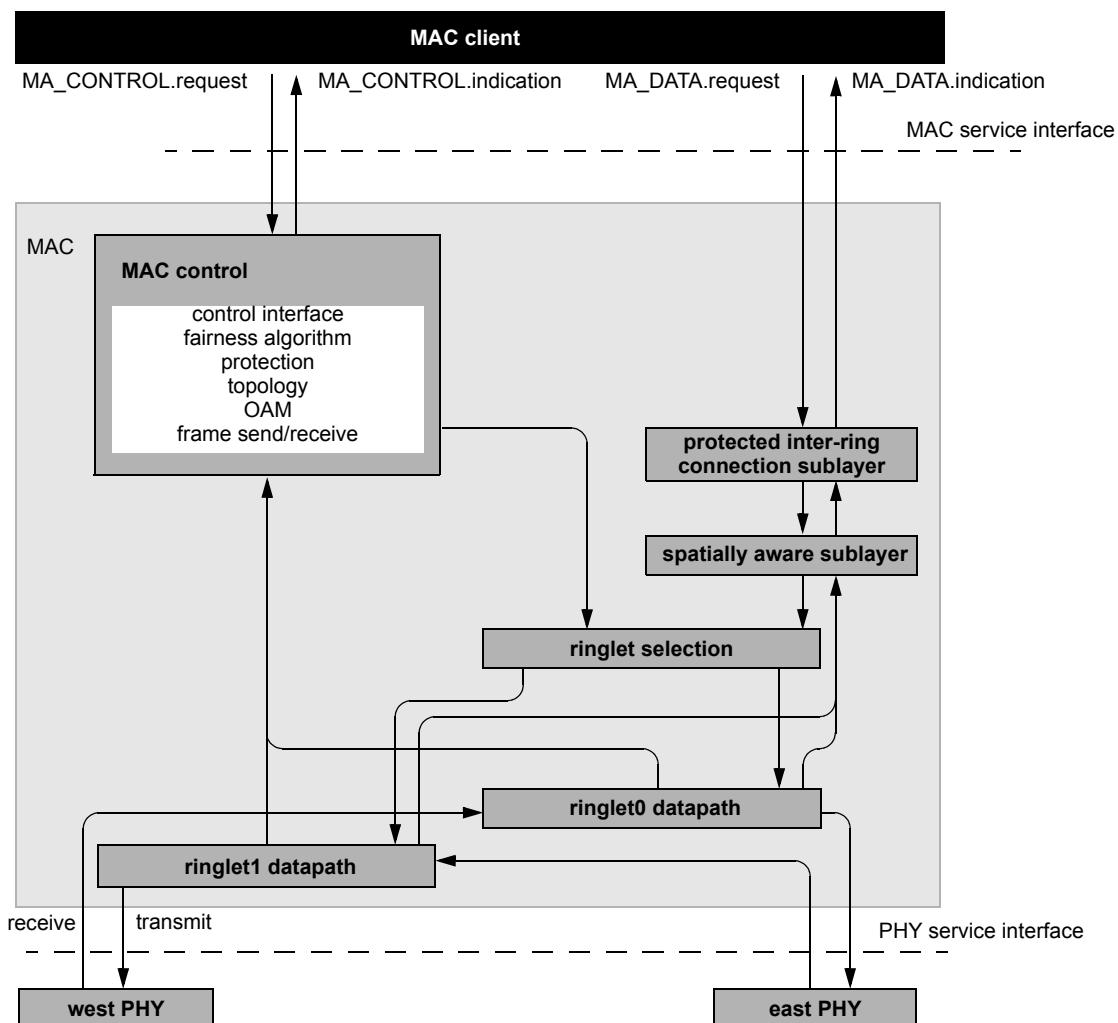


Figure 6.5—MAC control architecture

6.6.2 MAC datapath sublayer

The MAC datapath sublayer includes a single ringlet selection entity and two distinct instances of ringlet-specific datapaths, as illustrated in Figure 6.6.

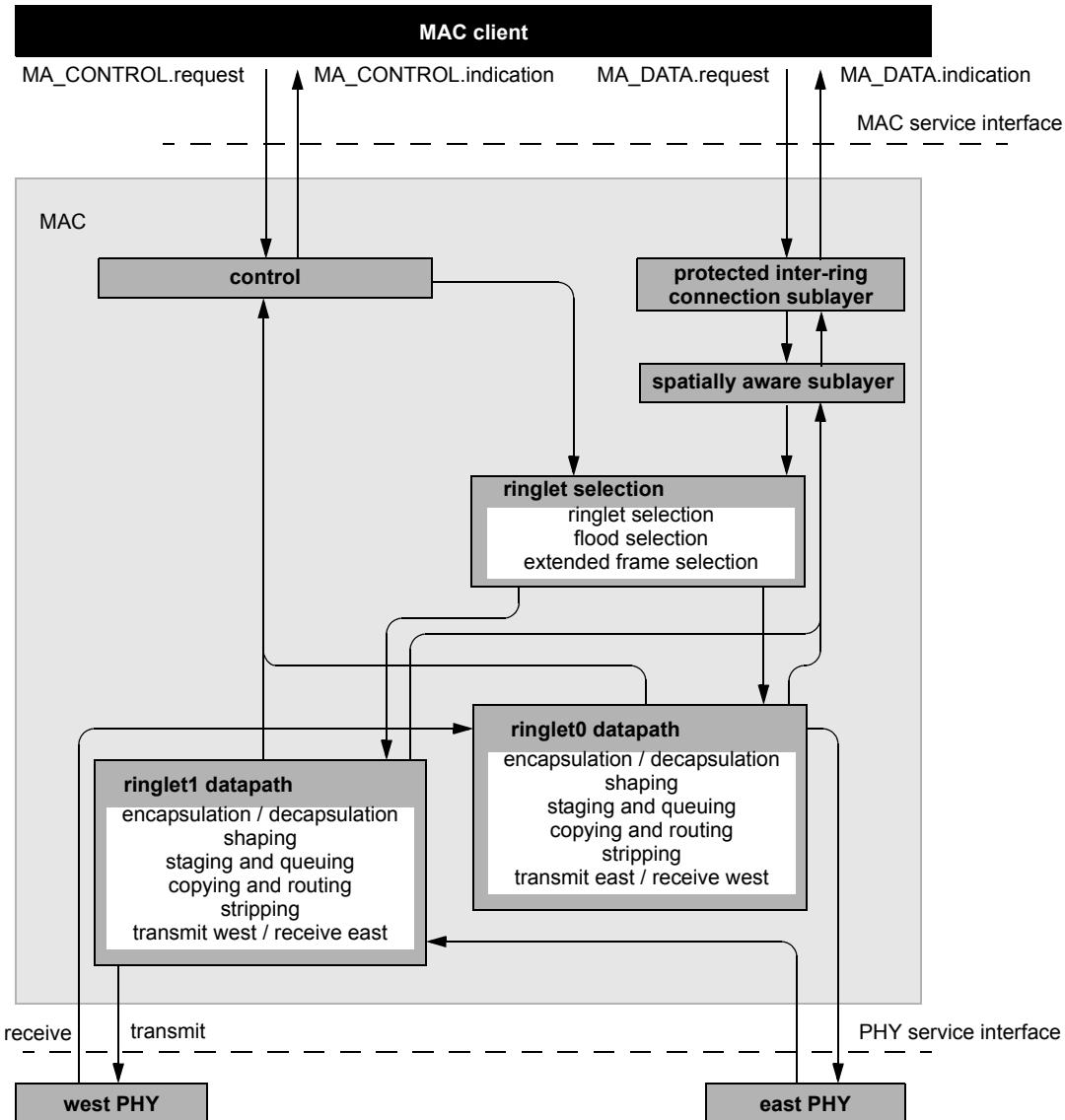


Figure 6.6—MAC datapath architecture

The ringlet selection entity uses the `destination_address`, the `source_address`, and the topology and protection database to determine the following:

- The ringlet to use to transmit the frame.
- Whether and how to flood the frame.
- Whether to use the basic frame format or the extended frame format.

The ringlet-specific datapath components provide the following functions:

- a) Encapsulation and decapsulation of client data frames (including insertion and extraction of header and trailer fields) on transmit and receive.
- b) Per service class traffic shaping to regulate access to the shared ring medium.
- c) Staging and queueing of frames at their source and queueing of transit frames.
- d) Copying and routing received frames to the MAC client and MAC control sublayer.
- e) Stripping frames from the ring on error or expiration.
- f) Transmission and reception of frames.

6.6.3 Flow of data within the MAC

Stations can support steering, center wrapping, or edge wrapping protection schemes (see 7.4.4). The paths through which data flows in steering systems, and in wrapping systems that are not wrapped, are illustrated in Figure 6.7.

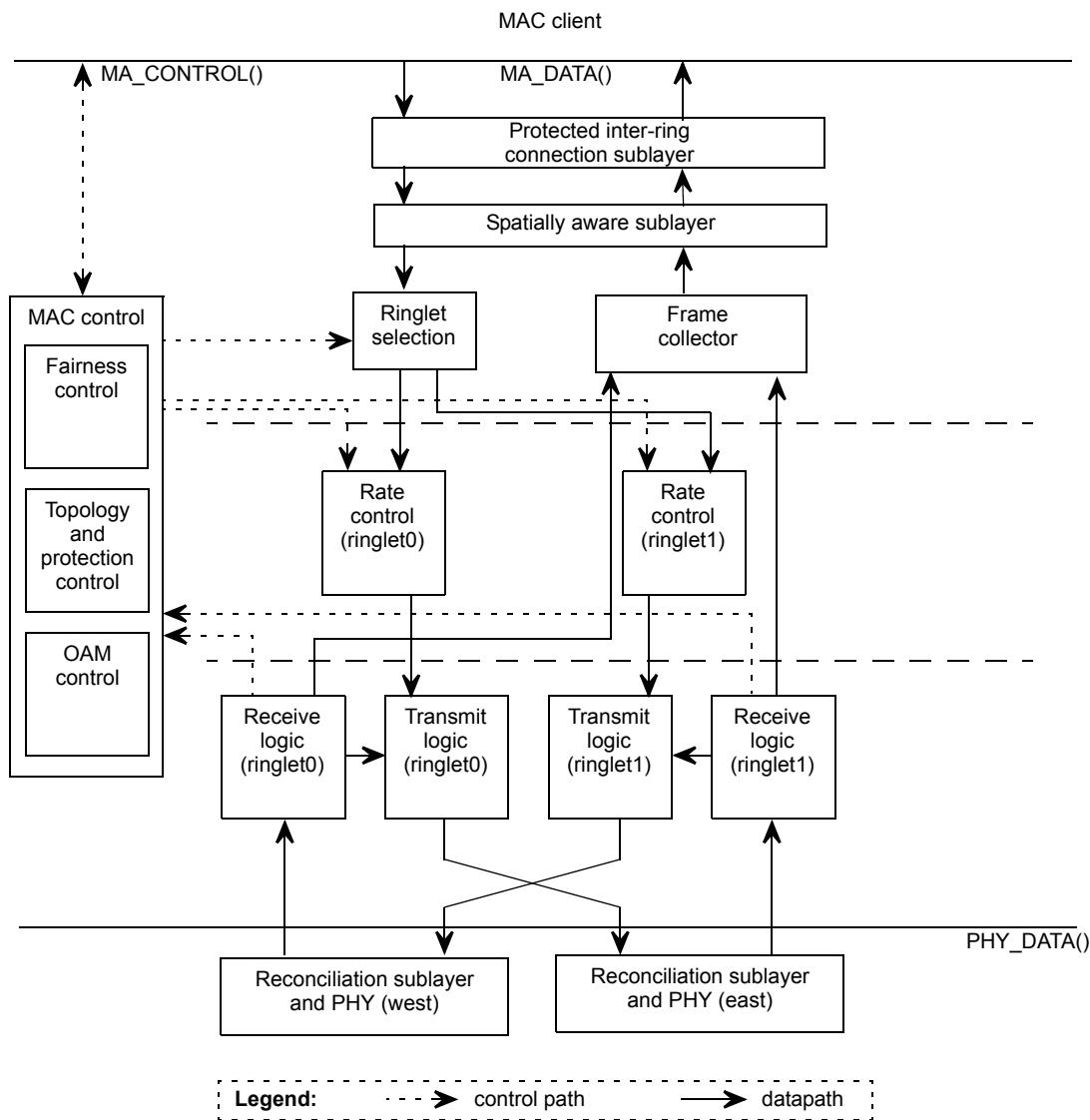


Figure 6.7—MAC reference model, showing internal MAC functions and data paths

The paths through which data flows in wrapping systems that use center wrapping are illustrated in Figure 6.8. For center wrapping systems, the unwrapped data path and the wrapped data path are not active at the same time. During normal conditions, the unwrapped data path is used and the wrapped data path is inactive. During local fault conditions, the wrapped data path is used and the unwrapped data path is inactive.

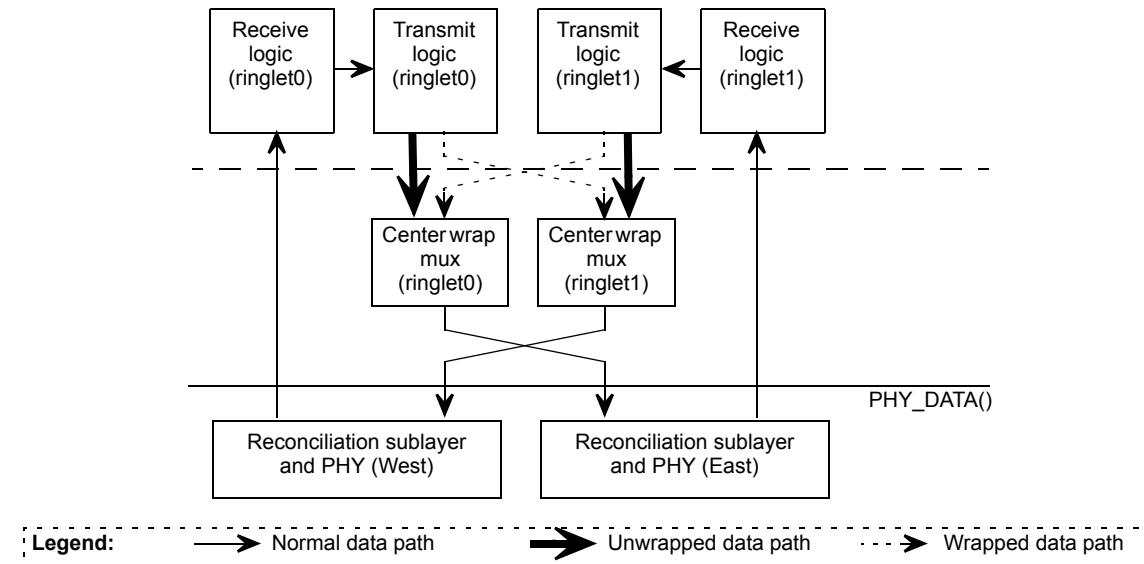


Figure 6.8—MAC reference model continued, showing center wrap data paths

The paths through which data flows in wrapping systems that use edge wrapping are illustrated in Figure 6.9. For edge wrapping systems, the unwrapped data path and the wrapped data path are sometimes active at the same time. During normal conditions, the unwrapped data path is used and the unwrapped data path is inactive. During local fault conditions, both the wrapped and the unwrapped data paths are active.

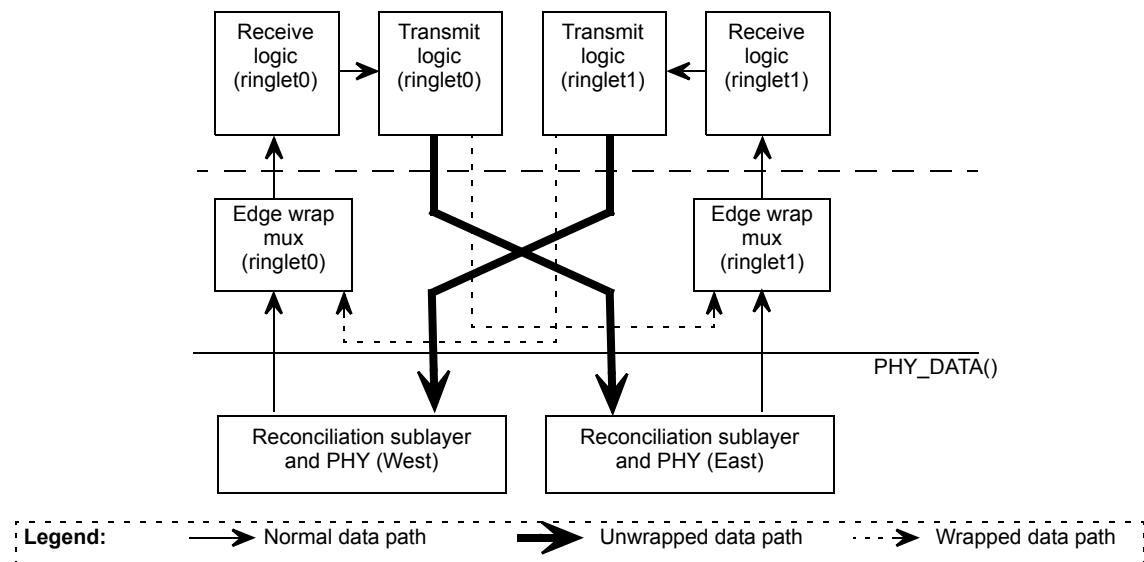


Figure 6.9—MAC reference model continued, showing edge wrap data paths

6.6.4 Reconciliation sublayer

The reconciliation sublayer (see Clause 8) is part of the physical layer and provides a uniform, reconciled service interface to the MAC layer. There is one reconciliation sublayer entity for each physical layer interface. Specific reconciliation sublayers are provided in Annex B and Annex C.

6.6.5 Medium access control

The RPR medium access control sublayer provides the access control for the physical layer medium. It also controls the transit path(s) through the MAC. Its functions include receiving frames, transmitting frames, ringlet selection, mapping of frames to the correct transit queue, rate control, fairness, protection, and topology discovery.

6.6.5.1 Receive

The receive entity (see 7.6) receives frames from the reconciliation sublayer and copies them to one or more of the MAC client, the MAC control sublayer, and the designated transit queue, as appropriate for each frame.

6.6.5.2 Transmit

The transmit entity (see 7.7) transmits frames to the reconciliation sublayer from the transit queue(s), the MAC control sublayer, and the MAC client.

6.6.5.3 Ringlet selection

Ringlet selection (see 7.7.1) can be specified entirely by the client, specified by the client but with the option for the MAC to override it for protection, or left entirely to the MAC.

6.6.5.4 Rate control

The rate control entity (see 7.5) governs the rate at which frames are transmitted from the client and from the transit queues, and it coordinates this control with the other MAC sublayers on the ring. It controls access by the MAC client to each service in order to ensure that rules for medium access and bandwidth allocation are obeyed.

Add traffic is shaped using per-service-class token buckets. These shapers are used to indicate to the MAC client (via the resultant *sendA*, *sendB*, and *sendC* indications) to cease making MA_DATA.requests for a particular service class.

Dynamic control of bandwidth to accommodate bursty traffic is accomplished through the fairness algorithm's bandwidth control mechanism (see Clause 10). This mechanism provides results to the fairness eligible shaper (see 7.5.7.5) to enable correct setting of the values for the *sendC* indication. In addition, the optional multi-choke mechanism enables the client to provide service data units to the MAC at an optimal rate even when more than one link is congested.

6.6.5.5 Fairness

Bandwidth management is done to maintain fairness for fairness eligible frames (those without or beyond allocated bandwidth), with mechanisms to assure that all stations receive their fair share of ring capacity across the links being used by the stations, where the fair share is not necessarily the same for all stations. The fairness algorithm (see Clause 10) ensures weighted dynamic distribution of available link bandwidths to source stations using those links.

6.6.5.6 Protection

By default, traffic is protected from failures in the ring and ring equipment (see 7.7.1 and Clause 11). The MAC client can choose not to protect a given frame. The protection entity provides the protection state machines and manages the protection database for the local MAC and coordination of this control with the other MAC sublayers on the ring.

6.6.5.7 Topology

The topology entity of the MAC sublayer (see Clause 11) manages the topology database. It also provides the topology state machines for the local MAC and coordination of this control with the other MAC control sublayers on the ring.

An RPR network consists of dual, counter-rotating ringlets. The MAC can present two views of the network to the MAC client: a flat view of the network, in which the MAC sublayer hides the dual-ringlet-based topology from the client, or a topology-aware view, which allows the MAC client to make data and control requests for specific ringlets. Topological information is collected via a MAC sublayer entity process known as topology discovery and is made available to the client via a MA_CONTROL.indication.

6.6.6 Operations, administration, and maintenance (OAM)

The Operations, administration, and maintenance (OAM) entity (see Clause 12) provides a set of control frames and indications to support configuration, fault localization, and ring maintenance.

6.6.7 MAC layer management entity (MLME)

The MAC layer management entity (MLME) (see Clause 13) is an independent entity that resides outside of the MAC layer in a separate management plane. The MLME contains the management information base for the MAC layer and provides get and set operations on the MIB to MLME SAP user-entities. The MLME reads from and writes to the MAC layer as a result of the invocation of MLME primitives.

6.7 Protocol Implementation Conformance Statement (PICS) proforma for Clause 6¹⁶

6.7.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 6, Medium access control (MAC) service and reference model, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

6.7.2 Identification

6.7.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

6.7.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, Medium access control (MAC) service and reference model
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

¹⁶*Copyright release for PICS proformas:* Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

6.7.3 PICS tables for Clause 6

6.7.3.1 Frame transmission services

Item	Feature	Subclause	Value/Comment	Status	Support
FTS1	Strict frame transmission	6.3	Able to generate and transmit strict frames	M	Yes []
FTS2	Relaxed frame transmission	6.3	Able to generate and transmit relaxed frames	M	Yes []

6.7.3.2 Service primitives

Item	Feature	Subclause	Value/Comment	Status	Support
SP1	MA_DATA.request	6.4.1	MA_DATA.request is implemented	M	Yes []
SP2	MA_DATA.indication	6.4.2	MA_DATA.indication is implemented	M	Yes []
SP3	MA_CONTROL.request	6.4.3	MA_CONTROL.request is implemented	M	Yes []
SP4	MA_CONTROL.indication	6.4.4	MA_CONTROL.indication is implemented	M	Yes []

7. Medium access control datapath

7.1 Datapath overview

This clause describes the datapath sublayer of the RPR MAC as shown in the shaded region of Figure 7.1. The selection of the datapath and the operations done on the selected datapath are together treated as the MAC datapath.

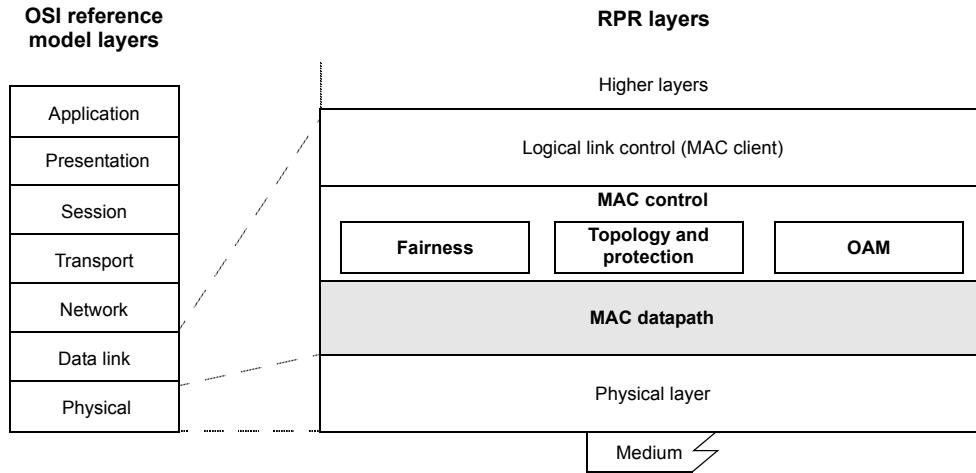


Figure 7.1—MAC datapath sublayer relationship to the ISO/IEC OSI reference model

The RPR MAC datapath sublayer provides the interaction between the client and the physical layer and the communication between peer datapath sublayers in other MACs on the same ring.

The datapath sublayer comprises:

- a) Frame reception logic (see 7.6), which further comprises the following:
 - 1) Checking: erroneous frames are discarded.
 - 2) Counting: frames are counted for traffic-monitoring statistics purposes.
 - 3) Filtering: frames are filtered to determine if they are to be copied to the client or to the control sublayer.
 - 4) Stripping: frames are stripped when they reach their destination.
 - 5) Updating: frames are modified to achieve proper next-hop behaviors.
 - 6) Queuing: frames are queued in the transit queues.
- b) Frame transit logic (see 7.4).
- c) Frame transmit logic (see 7.7), which further comprises the following:
 - 1) Ringlet selection.
 - 2) Stage queue processing.
 - 3) Service classes and per-service class shaping.
 - 4) Frame transmit selection.
 - 5) Frame wrapping.

The various pieces of the datapath sublayer are tied together through logical queues. For example, an invocation of MA_DATA.request places input into a queue leading to the PircTransmit state machine. As this and each subsequent state machine processes the input, it takes the input off of one queue and then places it onto another queue. This continues until the last queue leads to a PHY_DATA.request. Similarly, invocation of PHY_DATA.indication (by the reconciliation sublayer) provides a frame to each of the receive

state machines in turn, via logical queues, and then eventually to one of the transmit state machines, via the transit queue(s), and to MA_DATA.indication. These relationships are illustrated in a simplified overview in Figure 7.2.

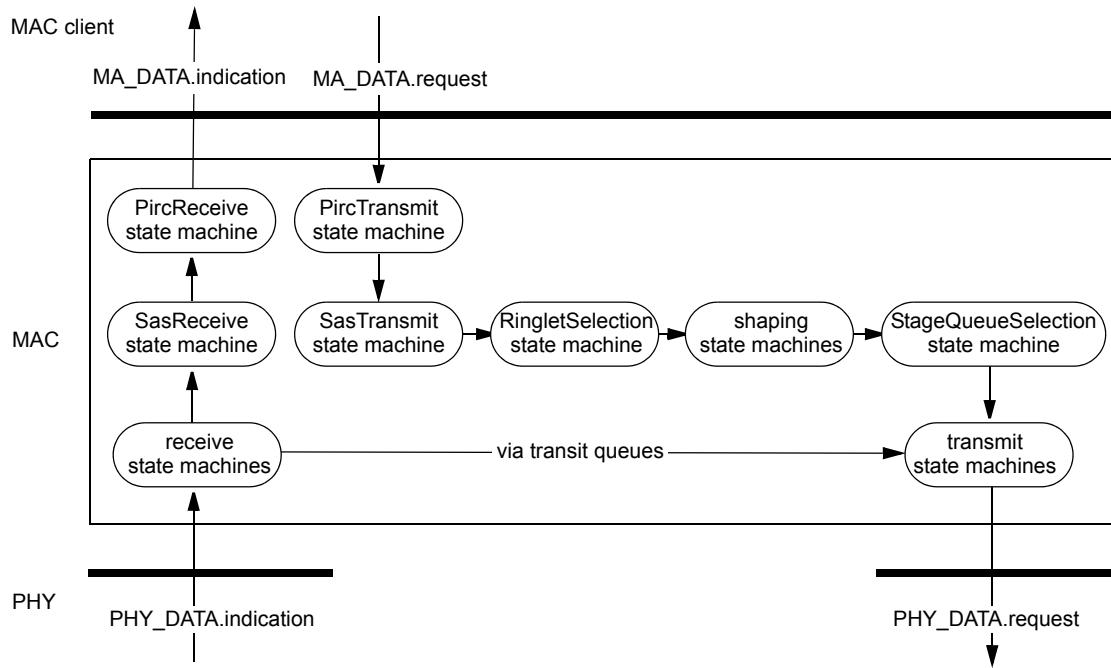


Figure 7.2—MAC datapath relationships

Dual instances of all but the RingletSelection state machine exist for each of the two ringlets. With the exception of ringlet selection, this clause describes per-ringlet behavior, with the assumption that the appropriate ringlet instance is in use.

7.2 Terminology and variables

7.2.1 Common state machine definitions

The following state machine inputs are used multiple times within this clause.

GROUP_BIT

A constant value derived from IEEE Std 802-2001, Figure 8, and specified by Equation (7.1).

$$(((\text{uint64_t})1) \ll 40) \quad (7.1)$$

NONE

A constant that indicates the absence of a value being passed to a routine.

NULL

A constant that indicates the absence of a value and (by design) cannot be confused with a valid value.

queue values

Enumerated values used to specify shared queue structures. Each queue provides a FIFO behavior, except for Q_RX_SS. One instance of each queue type exists in each datapath, except for Q_TX_RS.

- Q_RX_ADJUST—The queue identifier associated with the receiving adjuster.
 - Q_RX_ATD—The queue identifier associated with received ATD frames.
 - Q_RX_CHECK—The queue identifier associated with the receive checking.
 - Q_RX_CLIENT—The queue identifier associated with the receiver's client queue.
 - Q_RX_COUNT—The queue identifier associated with the receive link flow counters.
 - Q_RX_ECHO_REQ—The queue identifier associated with received echo request frames.
 - Q_RX_ECHO_RSP—The queue identifier associated with received echo response frames.
 - Q_RX_FAIR—The queue identifier associated with received fairness frames.
 - Q_RX_FDD—The queue identifier associated with received FDD frames.
 - Q_RX_FILTER—The queue identifier associated with the receiving filter.
 - Q_RX_FLUSH—The queue identifier associated with received flush frames.
 - Q_RX_F_C_CNT—The queue identifier associated with the receive filter control counters.
 - Q_RX_F_D_CNT—The queue identifier associated with the receive filter data counters.
 - Q_RX_LRTT_REQ—The queue identifier associated with received LRTT request frames.
 - Q_RX_LRTT_RSP—The queue identifier associated with received LRTT response frames.
 - Q_RX_ORG—The queue identifier associated with received organization-specific frames.
 - Q_RX_PHY—The queue identifier associated with the receiving PHY.
 - Q_RX_SAS—The queue identifier associated with SAS receive processing of frames.
 - Q_RX_SAS_NOTIFY—The queue identifier associated with received SAS notify frames.
 - Q_RX_STRIP—The queue identifier associated with the receiving stripper.
 - Q_RX_TC—The queue identifier associated with received TC frames.
 - Q_RX_TP—The queue identifier associated with received TP frames.
 - Q_TX_C_COUNT—The queue identifier associated with the control add flow counters.
 - Q_TX_COUNT—The queue identifier associated with the transmit link flow counters.
 - Q_TX_D_COUNT—The queue identifier associated with the data add flow counters.
 - Q_TX_FAIR—The queue identifier associated with the transmitter's fairness frame queue.
 - Q_TX_IDLE—The queue identifier associated with the transmitter's idle frame queue.
 - Q_TX_OAM—The queue identifier associated with control frame transmit requests.
 - Q_TX_PHY—The queue identifier associated with the transmitting PHY.
- (The PHY is not actually reached when transmitting into an edge by a wrapped station.)
- Q_RX_PIRC—The queue identifier associated with PIRC receive processing of frames.
 - Q_RX_PIRC_STATUS—The queue identifier associated with received PIRC status frames.
 - Q_TX_PIRC—The queue identifier associated with PIRC transmit processing of frames.
 - Q_TX_PTQ—The queue identifier associated with the primary transit queue.
 - Q_TX_ROUTE—The queue identifier associated with the transmit route selection.
 - Q_TX_RS—The queue identifier associated with the transmitter ringlet selection.
 - Q_TX_SAS—The queue identifier associated with SAS transmit processing of frames.
 - Q_TX_SS—The queue identifier associated with the transmitter stage queue selection.
 - Q_TX_STAGE—The queue identifier associated with the transmitter stage.
 - Q_TX_STQ—The queue identifier associated with the secondary transit queue.

SAS_GROUP_ADDRESS

The reserved group MAC address used by SAS (see 14.5.3.1).

Value: 01-80-C2-00-00-04.

STOMP

A constant value specified by Equation (7.2).

$$(0xFFFFFFFF) \quad (7.2)$$

TICK

The amount of time between shaper updates.

Range: [1 byte transmit time, 1024 bytes transmit time]

Default: 1 byte transmit time

7.2.2 Common state machine variables

One instance of each variable specified in this clause exists in each datapath, unless otherwise noted.

badFcsFrames

A counter for data or control frames where the *fcs* value did not match the expected *fcs* value. This includes data frames passed to the client as a result of *copyBadFcs* being set to TRUE.

classAAccessDelayTimer

Indicates the amount of time any classA frames were waiting to be transmitted and were not able to be transmitted.

classAAccessDelayTimerExpired

Indicates whether the access delay timer for classA add traffic has exceeded the *classAAccessDelayTimerThreshold*.

TRUE—Access delay timer for classA traffic has expired.

FALSE—(Otherwise.)

classAAccessDelayTimerThreshold

Indicates the maximum amount of time any classA add traffic can wait to be transmitted before indicating a defect.

Range: [100 microseconds, 25.5 milliseconds], in 100-microsecond increments

Default: specified by Equation (7.18)

classBAccessDelayTimer

Indicates the amount of time any classB frames were waiting to be transmitted and were not able to be transmitted.

classBAccessDelayTimerExpired

Indicates whether the access delay timer for classB add traffic has exceeded the *classBAccessDelayTimerThreshold*.

TRUE—Access delay timer for classB traffic has expired.

FALSE—(Otherwise.)

classBAccessDelayTimerThreshold

Indicates the maximum amount of time any classB add traffic can wait to be transmitted before indicating congestion.

Range: [100 microseconds, 25.5 milliseconds], in 100-microsecond increments

Default: 1.00 milliseconds

classCAccessDelayTimer

Indicates the amount of time any fairness eligible frames were waiting to be transmitted and were not able to be transmitted.

classCAccessDelayTimerExpired

Indicates whether the access delay timer for classC add traffic has exceeded the *classCAccessDelayTimerThreshold*.

TRUE—Access delay timer for classC traffic has expired.

FALSE—(Otherwise.)

classCAccessDelayTimerThreshold

Indicates the maximum amount of time any classC add traffic can wait to be transmitted before indicating congestion.

Range: [100 microseconds, 25.5 milliseconds], in 100-microsecond increments

Default: 10.00 milliseconds

containedFrames

Counter for frames discarded due to context containment.

currentTime

A value representing the current time, with a precision equivalent to at least 1 TICK. There is one instance of this variable for the MAC. Within the state machines of this standard, the *currentTime* value and other time comparison values are assumed to be so large that they would never overflow within the lifetime of any conforming equipment.

lineRate

The data rate of the ringlet identified by *myRi*, in units of bits per second.

myDualQueueStation

Indicates whether dual queue station capabilities are implemented. There is one instance of this variable for the MAC.

TRUE—Indicates a dual queue implementation.

FALSE—Indicates a single queue implementation.

myEdgeState

Indicates the adjacent edge condition.

NORMAL—The datapath is operating without wrapping or passthrough.

INTO_EDGE—The transmit side of the datapath is associated with a failed span.

FROM_EDGE—The receive side of the datapath is associated with a failed span.

PASSTHROUGH—The datapath is in passthrough.

myProtectMethod

The type of protection being employed by this station. There is one instance of this variable for the MAC.

CENTER_WRAP—This station uses center wrapping when it needs to protect.

EDGE_WRAP—This station uses edge wrapping when it needs to protect.

STEER—This station uses steering when it needs to protect.

myRi

The ringlet identifier associated with this datapath entity. The enumerated values are the same as those for the *ri* field (see 9.6.1).

RINGLET_0—The datapath is transmitting onto ringlet0.

RINGLET_1—The datapath is transmitting onto ringlet1.

myRxFilter

The type of filtering done when receiving frames destined to a client.

RX_BASIC—Filter data frames for a host client.

RX_FLOOD—Filter data frames for a bridge client.

passA0

The indication to the MAC of whether transmission of subclassA0 traffic is allowed.

TRUE—Indicates permission to transmit subclassA0 traffic.

FALSE—(Otherwise.)

passA1

The indication to the MAC of whether transmission of subclassA1 traffic is allowed.

TRUE—Indicates permission to transmit subclassA1 traffic.

FALSE—(Otherwise.)

passAddFe

A boolean value indicating whether fairness eligible traffic can be added to the ringlet.

passAddFeCongested

A boolean value indicating whether fairness eligible traffic bound for a destination beyond the congestion point can be added to the ringlet.

passAddFeOrStq

A boolean output value indicating whether a classB/classC frame from the STQ or a fairness eligible add frame is allowed by the source shaper.

scffErrors

Counter for single-choke fairness frames (SCFFs) received with parity errors or FCS errors. (See also 12.6.1.2.)

sendA

The indication to the client or control sublayer of whether it can send classA traffic.

TRUE—Indicates permission to transmit classA traffic.

FALSE—(Otherwise.)

sendB

The indication to the client or control sublayer of whether it can send allocated classB traffic.

TRUE—Indicates permission to transmit classB-CIR traffic.

FALSE—(Otherwise.)

sendC

The indication to the client or control sublayer of whether and how far it can send fairness eligible traffic. The value indicates the maximum number of hops that the datapath has permission to transmit fairness eligible traffic.

sendD

The indication to the datapath sublayer or control sublayer of whether a frame using unreserved bandwidth is allowed by the downstream shaper.

TRUE—Indicates permission to transmit unreserved traffic.

FALSE—(Otherwise.)

sendI

The indication to the control sublayer of whether an idle frame is allowed by the idle shaper.

TRUE—Indicates permission (and requirement) to transmit an idle frame.

FALSE—(Otherwise.)

sendM

The indication to the control sublayer of whether a control frame is allowed by the MAC control shaper.

TRUE—Indicates permission to transmit control frames.

FALSE—(Otherwise.)

sizePtq

The size, in bytes, of the primary transit queue.

sizeStq

The size, in bytes, of the secondary transit queue. The minimum value is $6*mtuSize$ bytes, as constrained by the ranges for *stqLowThreshold*, *stqMedThreshold*, *stqHighThreshold*, and *stqFullThreshold*.

sourceCheck

A subset of the topology and status database containing just each entry's MAC address. The *sourceCheck* database is generated individually for each ringlet, with each entry being the station that is the same hops away (on the specified ringlet) as the index into the table. It is computed only at the point when the topology becomes stable and valid (see 11.6.6). During periods when the topology is unstable or invalid, source checking is done using the *sourceCheck* database computed for the last valid topology.

stqDepth

The number of occupied bytes in the secondary transit queue (*sizeStq* - *SpaceInStq()*).

stqFullThreshold

A level of STQ occupancy at or above which the STQ is considered too full to allow transmit frames to have precedence over transit frames. Defined only for dual queue implementations.

Range: $[stqHighThreshold + mtuSize, sizeStq - mtuSize]$

Default: *sizeStq* - *mtuSize*

tossWrongRingletIDs

An indication that opposing-ringlet frames are to be discarded.

ttlExpiredFrames

Counter for frames received with expired *ttl* values.

wrongRingletSensed

Set in ReceiveCheck state machine by a control frame or data frame received from the opposing ringlet. Cleared in the WrongRinglet state machine.

7.2.3 Common state machine routines

Broadcast(macAddress)

Indicates whether the supplied address is the broadcast address, as specified by Equation (7.3).

TRUE—The address is the broadcast address.

FALSE—(Otherwise.)

$$(\text{macAddress} == \text{0xFFFFFFFFFFFF}) \quad (7.3)$$

ConsistentHopCount(frame)

Indicates whether a received frame has traveled the number of hops that is consistent with the receiving station's topology and status database, as specified by Equation (7.4).

TRUE—The frame traveled a consistent number of hops.

FALSE—(Otherwise.)

$$(\text{sourceCheck}[\text{myRi}] [\text{frame.ttlBase} - \text{frame.ttl} + 1] == \text{frame.sa}) \quad (7.4)$$

Crc16(frame)

Returns the CRC16 of the data frame header or control frame header (see 9.2.2.7 and 9.3.2.7).

Crc32(frame)

Returns the CRC32 of the data frame or control frame (see 9.2.2.12 and 9.3.2.11).

Crc32Compact(frame)

Returns the CRC32 of the compact frame (see 9.4.2.6 and 9.5.2.5).

DefaultRinglet()

Assigns a value to the ringlet_id parameter based on the destination MAC address.

RINGLET_0—The first choice for the given destination address is ringlet0.

RINGLET_1—The first choice for the given destination address is ringlet1.

Dequeue(queue)

Returns the next available frame from the specified queue within the datapath identified by myRi, if a frame is available. If the requested queue is Q_TX_RS, myRi is ignored, and the only instance of Q_TX_RS is used.

frame—The next available frame.

NULL—No frame available.

DequeueRi(ringlet, queue)

Returns the next available frame from the specified queue within the datapath of the identified ringlet, if a frame is available. If the requested queue is Q_TX_RS, ringlet is ignored, and the only instance of Q_TX_RS is used.

frame—The next available frame.

NULL—No frame available.

EdgeAllowedFrame(frame)

Indicates whether frame is a type of frame that is allowed to be transmitted into an edge, as specified by Equation (7.5).

TRUE—The frame is allowed to be transmitted into an edge.

FALSE—(Otherwise.)

$$((\text{frame.ft} == \text{FT_FAIRNESS} \&\& \text{frame.ffType} == \text{SINGLE_CHOKE}) \mid\mid \\ (\text{frame.ft} == \text{FT_CONTROL} \&\& \\ \text{frame.controlType} == \text{CT_TOPO_PROT} \&\& \\ \text{frame.ttl} == \text{MAX_STATIONS})) \quad (7.5)$$

Enqueue(queue, frame)

Places the frame at the tail of the specified queue within the datapath identified by myRi. If the requested queue is Q_TX_RS, myRi is ignored, and the only instance of Q_TX_RS is used.

EnqueueRi(ringlet, queue, frame)

If the requested queue is Q_TX_RS, ringlet_id is set to the value of ringlet, and the only instance of Q_TX_RS is used.

EntryInQueue(queue)

Indicates if an entry (a full header for cut-through or a full frame for store-and-forward) is available from the specified queue within the datapath identified by *myRi*.

TRUE—An entry is available in the specified queue.

FALSE—(Otherwise.)

ExamineQueue(queue, frameType, serviceClass)

Looks inside the specified queue within the datapath identified by *myRi* for the next available frame of the specified frame type and the specified service class. If the frame type is specified as NONE, no frame type constraint is applied. If such a frame is available, it is returned, without being removed from the queue.

frame—The next available frame meeting the specified criteria.

NULL—No frame available meeting the specified criteria.

FairnessEligible(frame)

Indicates whether the frame is fairness eligible (see 9.6.2), as specified by Equation (7.6).

TRUE—The frame is fairness eligible.

FALSE—(Otherwise.)

$$(((\text{frame.sc} == \text{CLASS_B}) \&\& \text{frame.fe}) \mid\mid \text{frame.sc} == \text{CLASS_C}) \quad (7.6)$$

Max(value1, value2)

Returns the numerically larger of the two values.

Min(value1, value2)

Returns the numerically smaller of the two values.

Multicast(macAddress)

Indicates whether the supplied address is a multicast (or broadcast) address, as specified by Equation (7.7).

TRUE—The address is a multicast (or broadcast) address.

FALSE—(Otherwise.)

$$((\text{macAddress} \& \text{GROUP_BIT}) != 0) \quad (7.7)$$

MyEdgeRinglet()

Returns the *ri* value for the ringlet transmitting into an edge, which is the ringlet whose *myEdgeState* variable has the value of INTO_EDGE.

Other(ringletIdentifier)

Returns the ringlet not identified by *ringletIdentifier*, as specified by Equation (7.8).

$$(\text{ringletIdentifier} == \text{RINGLET_0} ? \text{RINGLET_1} : \text{RINGLET_0}) \quad (7.8)$$

Parity(frame)

Returns the parity (see 9.4.2) of the referenced bits in idle or fairness frames.

Provided(parameter)

To indicate whether an optional parameter was provided in the MA_DATA.request service primitive invocation or by a sublayer of the MAC.

TRUE—The parameter was provided.

FALSE—(Otherwise.)

Reachable(address, ringlet)

To indicate whether the topology database (see 11.5) shows that the specified destination MAC address is reachable via the specified ringlet. The reachability is determined from topoEntry[*Other(ringlet)*][*index*].reachable, where *index* is the index value of the station entry with topoEntry[*Other(ringlet)*][*index*].macAddress equal to *address*.

TRUE—The destination address is reachable via the specified ringlet.

FALSE—(Otherwise.)

Replicate(frame)

Makes two copies of the supplied frame with the names *frame0* and *frame1*.

Round100(timeAmount)

Returns the closest integral multiple of 100 microseconds that is greater than or equal to the *timeAmount*.

SameSide(frame)

Indicates whether the frame behaves as if it were on this ringlet, as specified by Equation (7.9).

TRUE—The frame is on the ringlet of the current context or behaves as if it is.

FALSE—(Otherwise.)

$$(\text{frame.r}i == \text{myR}i \quad || \quad (\text{myProtectMethod} == \text{CENTER_WRAP} \quad \&\& \quad \text{myEdgeState} == \text{INTO_EDGE})) \quad (7.9)$$

SizeOf(frame)

Returns the number of bytes in the frame. The entire frame, from header through trailer, is included. If the frame is an extended data frame, the extended fields are also included.

SpaceInPtq()

Returns the number of bytes remaining in the primary transit queue.

SpaceInStq()

Returns the number of bytes remaining in the secondary transit queue.

Stomp(frame)

Returns the stomped value of the CRC32 of the frame (see E.2.3).

Unicast(macAddress)

Indicates whether the supplied address is a unicast address, as specified by Equation (7.10).

TRUE—The address is a unicast address.

FALSE—(Otherwise.)

$$((\text{macAddress} \quad \& \quad \text{GROUP_BIT}) \quad == \quad 0) \quad (7.10)$$

7.2.4 Variables and literals defined in other clauses

This clause references the following parameters, literals, and variables defined in Clause 6:

```
destination_address
destination_address_extended
enableSas
FLOOD_BIDIR
FLOOD_NONE
FLOOD_UNIDIR
flooding_form
mac_protection
mark_fe
optionSasMcastScope
remapRxAddrs
request_sas
RI_0
RI_1
ringlet_id
source_address
source_address_extended
strict_order
transmissionControl
```

This clause references the following literals and routines defined in Clause 8:

```
PHY_READY.indication()
READY
```

This clause references the following literals defined in Clause 9:

CLASS_A0
CLASS_A1
CLASS_B
CLASS_C
CONTROL_MAX
CONTROL_MIN
CT_FDD
CT_LRTT_REQ
CT_LRTT_RSP
CT_OAM_ECHO_REQ
CT_OAM_ECHO_RSP
CT_OAM_FLUSH
CT_OAM_ORG
CT_OAM_PIRC_STATUS
CT_OAM_SAS_NOTIFY
CT_STATION_ATD
CT_TOPO_CHKSUM
CT_TOPO_PROT
DATA_MIN
EXT_HDR_SIZE
FAIRNESS_SIZE
FI_BIDIR
FI_NONE
FI_UNIDIR
FT_CONTROL
FT_DATA
FT_FAIRNESS
FT_IDLE
IDLE_SIZE
RINGLET_0
RINGLET_1

This clause references the following literals, variables, and routines defined in Clause 10:

addRateCongestedOK
addRateOK
advertisingInterval
ageCoef
allowedRate
allowedRateCongested
FeAddByteAvailable()
FeFwByteAvailable()
hopsToCongestion
LINK_RATE
MULTI_CHOKE
SINGLE_CHOKE
stqHighThreshold

This clause references the following literals and variables defined in Clause 11, where the local reference to per-ringlet arrays assumes the array element for the current datapath (e.g., *keepaliveTime[0]* is used for the ringlet0 datapath instance and *keepaliveTime[1]* is used for the ringlet1 datapath instance):

badFcsUser, referred to as *ringInfo.badFcsUser* in Clause 11
 CLOSED_RING
containmentActive
keepaliveTime
 MAX_STATIONS
mtuSize, referred to as *ringInfo.mtuSize* in Clause 11
myMacAddress, referred to as *myTopoInfo.macAddress* in Clause 11
numStations, referred to as *ringInfo.numStations* in Clause 11
 OK_IN_USE
 OPEN_RING
protConfig, referred to as *myTopoInfo.protConfig* in Clause 11
secMac, referred to as *myTopoInfo.secMac* in Clause 11
 STEERING
topoEntry
topoType, referred to as *ringInfo.topoType* in Clause 11
 WRAPPING

7.2.5 Flow count variables

fromClientBcastFrames
 Counter for broadcast frames added by the client.
fromClientMcastFrames
 Counter for multicast (but not broadcast) frames added by the client.
fromClientMcastClassABytes
fromClientMcastClassAFrames
fromClientMcastClassBCirBytes
fromClientMcastClassBCirFrames
fromClientMcastClassBEirBytes
fromClientMcastClassBEirFrames
fromClientMcastClassCBytes
fromClientMcastClassCFrames
 Counters for multicast (and broadcast) bytes and frames added by the client for each service class.

fromClientUcastClassABytes
fromClientUcastClassAFrames
fromClientUcastClassBCirBytes
fromClientUcastClassBCirFrames
fromClientUcastClassBEirBytes
fromClientUcastClassBEirFrames
fromClientUcastClassCBytes
fromClientUcastClassCFrames
 Counters for unicast bytes and frames added by the client for each service class.

fromCtrlFrames
 Counter for frames added by the MAC control sublayer.
fromCtrlFairnessFrames
fromCtrlFddFrames
fromCtrlLrttReqFrames
fromCtrlLrttRspFrames
fromCtrlOamEchoReqFrames
fromCtrlOamEchoRspFrames
fromCtrlOamFlushFrames
fromCtrlOamOrgFrames
fromCtrlOamPircStatusFrames

fromCtrlOamSasNotifyFrames
fromCtrlTopoATDFrames
fromCtrlTopoSumFrames
fromCtrlTopoTPFrames

Counters for frames added by the MAC control sublayer for each *controlType*.

rxMcastClassABytes
rxMcastClassAFrames
rxMcastClassBCirBytes
rxMcastClassBCirFrames
rxMcastClassBEirBytes
rxMcastClassBEirFrames
rxMcastClassCBytes
rxMcastClassCFrames

Counters for multicast (and broadcast) bytes and frames received for each service class.

rxUcastClassABytes
rxUcastClassAFrames
rxUcastClassBCirBytes
rxUcastClassBCirFrames
rxUcastClassBEirBytes
rxUcastClassBEirFrames
rxUcastClassCBytes
rxUcastClassCFrames

Counters for unicast bytes and frames received for each service class.

toClientBcastFrames

Counter for broadcast frames copied to the client.

toClientMcastFrames

Counter for multicast (but not broadcast) frames copied to the client.

toClientMcastClassABytes
toClientMcastClassAFrames
toClientMcastClassBCirBytes
toClientMcastClassBCirFrames
toClientMcastClassBEirBytes
toClientMcastClassBEirFrames
toClientMcastClassCBytes
toClientMcastClassCFrames

Counters for multicast (and broadcast) bytes and frames copied to the client for each service class.

toClientUcastClassABytes
toClientUcastClassAOctets
toClientUcastClassBCirBytes
toClientUcastClassBCirFrames
toClientUcastClassBEirBytes
toClientUcastClassBEirFrames
toClientUcastClassCBytes
toClientUcastClassCFrames

Counters for unicast bytes and frames copied to the client for each service class.

toCtrlFrames

Counter for frames copied to the MAC control sublayer.

toCtrlFairnessFrames

toCtrlFddFrames
toCtrlLrttReqFrames
toCtrlLrttRspFrames
toCtrlOamEchoReqFrames
toCtrlOamEchoRspFrames
toCtrlOamFlushFrames
toCtrlOamOrgFrames
toCtrlOamPircStatusFrames
toCtrlOamSasNotifyFrames
toCtrlTopoATDFrames
toCtrlTopoSumFrames
toCtrlTopoTPFrames

Counters for frames copied to the MAC control sublayer for each *controlType*.

txMcastClassABytes
txMcastClassAFrames
txMcastClassBCirBytes
txMcastClassBCirFrames
txMcastClassBEirBytes
txMcastClassBEirFrames
txMcastClassCBytes
txMcastClassCFrames

Counters for multicast (and broadcast) bytes and frames transmitted for each service class.

txUcastClassABytes
txUcastClassAFrames
txUcastClassBCirBytes
txUcastClassBCirFrames
txUcastClassBEirBytes
txUcastClassBEirFrames
txUcastClassCBytes
txUcastClassCFrames

Counters for unicast bytes and frames transmitted for each service class.

7.3 Service classes

Client data are classified into three service classes, as listed in Table 7.1. ClassA traffic is allocated with a committed information rate (CIR) and provides the lowest MAC end-to-end delay and jitter¹⁷ bounds. ClassB traffic is allocated with a CIR and provides bounded MAC end-to-end delay and jitter for the amount of traffic within the profile of the CIR. Any classB traffic amount beyond the allocated CIR is referred to as excess information rate (EIR) classB traffic. ClassC traffic is not allocated. ClassB-EIR and classC traffic is marked as fairness eligible (by setting the *fe* field) by the MAC and provides no MAC end-to-end delay or jitter bounds. The guarantees of bandwidth and delay or jitter bounds are the service guarantees for each service class.

NOTE—The service class guarantees can be guaranteed only within the bounds of the setting of the shaper parameters.

¹⁷MAC end-to-end delay and jitter are defined to be measured from the time when a frame is requested to be transmitted via MA_DATA.request until it is received via MA_DATA.indication.

Table 7.1—Service classes

Class of service			Quality of service			
Name	Example use	Subclass	Guaranteed bandwidth	Delay/jitter	Bandwidth type	Bandwidth subtype
classA	real time ^a	subclassA0	yes	low	allocated	reserved
		subclassA1				reclaimable
classB	near real time ^b	classB-CIR	yes	bounded	allocated	
		classB-EIR	no	unbounded	opportunistic	
classC	best effort ^c	—				

^aReal-time services need as close to immediate servicing as possible. Examples include interactive telephony.

^bNear real-time services need short, but not immediate, time frames for servicing. Examples include noninteractive video.

^cBest effort services are not time dependent, and they do not need a minimum amount of bandwidth within any particular time frame. Examples include mail.

On ingress, classA traffic has higher precedence than classB traffic, which has higher precedence than classC traffic. On transit, classA traffic has higher precedence than classB or classC traffic for dual queue designs. No distinction is made between classB and classC traffic on transit for dual queue designs. No distinction is made between any of the classes on transit for single queue designs. Strict precedence on ingress and transit is used to ensure that frames are not reordered within a service class and to provide the bandwidth guarantees and delay and jitter bounds specified for each service class.

Control traffic shares the same service classes as used by client data, with ingress precedence at each service class over client data with the same service class.

7.3.1 Service class classA

ClassA service provides an allocated, guaranteed data rate, a low end-to-end delay, and jitter bounded by an order of $numStations * mtuSize$. ClassA traffic has precedence over classB and classC traffic at ingress to the ring, and during transit through the ring (for dual queue stations).

ClassA traffic is not subject to the fairness algorithm at ingress to the ring or when transiting through the ring. Therefore, the *fe* (fairness eligible) bit in the RPR header is always set to 0 on classA traffic.

ClassA traffic moves through the primary transit path in each station as it propagates around the ring (see 7.4).

Internal to the MAC, classA traffic is partitioned into two subclasses: subclassA0 and subclassA1. This partitioning is done to increase the ability of the ring to reclaim unused classA traffic. The MAC client requests classA traffic, not one of the internal subclasses, which are not visible to the client. The MAC is configured for a total classA amount, from which it determines how much is subclassA0 and how much is subclassA1. The division of classA is based on ring circumference and the size of the secondary transit queue (STQ) in that station. Single queue implementations always allocate 100% of classA traffic to subclassA0 and 0% to subclassA1. The MAC advertises, via the station ATD frame, a reserved bandwidth allocation equal to the internal subclassA0 amount. The reclaimable bandwidth allocated to subclassA1 can be easily reclaimed by traffic of classB-EIR and classC when not being used by the station originating the classA traffic being reclaimed.

The amount of a station's classA traffic that can be sent as subclassA1 should be determined by how much classB and classC transit traffic can be queued by the local station while it is signaling upstream stations to decrease their excess traffic. Based on an implementation's STQ size, the default amount of subclassA1 and classB-CIR add traffic that can be supported can be estimated by Equation (7.11). Equation (7.11) is intended to represent both aggressive and conservative fairness modes (see Clause 10), is based on several simplistic assumptions, and further elaborated in G.1. (*addRateA0*, *addRateA1*, and *addRateB* are specified in 7.5.5, 7.5.5, and 7.5.6, respectively. *sizeStq* is specified in 7.2.2. *stqHighThreshold* is specified in 10.2.2. *responseTime* is estimated in G.1.)

$$\text{addRateA1} \leq ((\text{sizeStq} - \text{stqHighThreshold}) / \text{responseTime}) - \text{addRateB} \quad (7.11)$$

How the actual split of classA into subclassA1 and subclassA0 is done is an implementation decision. If the decision does not follow Equation (7.11), it is possible that the bandwidth, delay, and jitter of classA is not guaranteed.

Subtracting *addRateA1* from the total allocation for classA yields the amount of classA traffic to be reserved as *addRateA0*, which the MAC advertises for each ringlet via the ATT_STATION_BW value (see 11.6.11). The subclassA0 advertisements are used to determine the *reservedRate* for a ringlet (see 11.2.6). Subtracting the *reservedRate* from the LINK_RATE yields the *unreservedRate* (see 11.2.4).

7.3.2 Service class classB

ClassB service provides an allocated, guaranteed data rate, and bounded end-to-end delay and jitter for the traffic within the allocated rate, bounded on the order of a ring round-trip time (RRTT). ClassB also provides access to additional best effort data transmission that is not allocated, guaranteed, or bounded and is subject to the fairness algorithm. ClassB traffic (including classB-EIR) has precedence over classC traffic at ingress to the ring.

ClassB service has similarities to classA service, described above, in that frame transmission rates within the allocated rate profile (known as classB committed information rate, or classB-CIR) are guaranteed a bounded delay and jitter, although with higher bounds than for classA frames, and are not marked as fairness eligible. Traffic within the allocated rate profile is not subject to the fairness algorithm at ingress to the ring or when transiting through stations on the ring.

ClassB traffic also has similarities to classC service, described below, in that traffic beyond the allocated rate profile (known as classB excess information rate, or classB-EIR) is subject to the fairness algorithm, and is marked by the MAC as such with the *fe* bit in the RPR header prior to transmission on the ring. Fairness eligible frames are counted as part of the RPR fairness algorithm both at ingress to the ring and while transiting stations on the ring.

Internal to the MAC, classB traffic is partitioned into classB-CIR and classB-EIR at ingress to the ring through the use of the *fe* field. The client can either let the MAC choose based on the presence or absence of *sendB* (see 7.5.6) or it can force a classB frame to be considered only for classB-EIR by setting the *mark_fe* parameter (see 7.7.4). ClassB-EIR traffic receives a higher quality of service than classC because all classB traffic, including classB-EIR traffic, receives ingress precedence over classC traffic.

NOTE—As classB-EIR has higher precedence than classC, classB-EIR can starve out classC. As the submission of classB-EIR and classC traffic is made by the client, the client has control over the relative amounts of classB-EIR and classC traffic.

In a single queue implementation, all classB traffic moves through the primary transit path. In a dual queue implementation, classB traffic moves through the secondary transit path, regardless of whether the frame is marked fairness eligible (see 7.4).

7.3.3 Service class classC

ClassC service provides a best-effort traffic service with no allocated or guaranteed data rate and no bounds on end-to-end delay or jitter. ClassC traffic has the lowest precedence for ingress to the ring.

ClassC traffic is always subject to the fairness algorithm and is marked by the MAC as such with the *fe* bit in the RPR header prior to transmission on the ring. ClassC frames are counted as part of the RPR fairness algorithm both at ingress to the ring and while transiting stations on the ring.

In a single queue implementation, classC traffic moves through the primary transit path. In a dual queue implementation, classC traffic moves through the secondary transit path.

Fairness eligible traffic (classB-EIR and classC) is opportunistic rather than allocated, in that it uses bandwidth available from the unallocated bandwidth and the unused reclaimable bandwidth, as described below. A weighted fairness algorithm (see Clause 10) is used to partition fairness eligible bandwidth among contending stations.

7.3.4 Reclamation

Allocated bandwidth can be reused, or reclaimed, by a lower priority service class whenever the reclamation does not effect the service guarantees of any equal or higher priority class(es) on the local station or on any other station on the ring. A small reclamation-caused effect on any equal or higher service class that does not violate the service guarantees of that class (i.e., a minor priority inversion) is allowed. Any reclamation-caused effect that violates any equal or higher class' guarantee (i.e., a major priority inversion) is not allowed. Reclamation of the unused bandwidth of all but the subclassA0 traffic can be done by any station that determines through the fairness algorithm that it is permitted to add fairness eligible traffic (see 7.5.2). The fairness algorithm and the datapath shapers guarantee that fairness eligible traffic can be added in amounts that cause no more than minor priority inversions.

NOTE—Additional traffic can be reclaimed, beyond that reclaimed by the fairness algorithm, when a station can do so without impacting the service guarantees of the service classes whose traffic is being reclaimed. Any mechanisms for accomplishing this are beyond the scope of this standard.

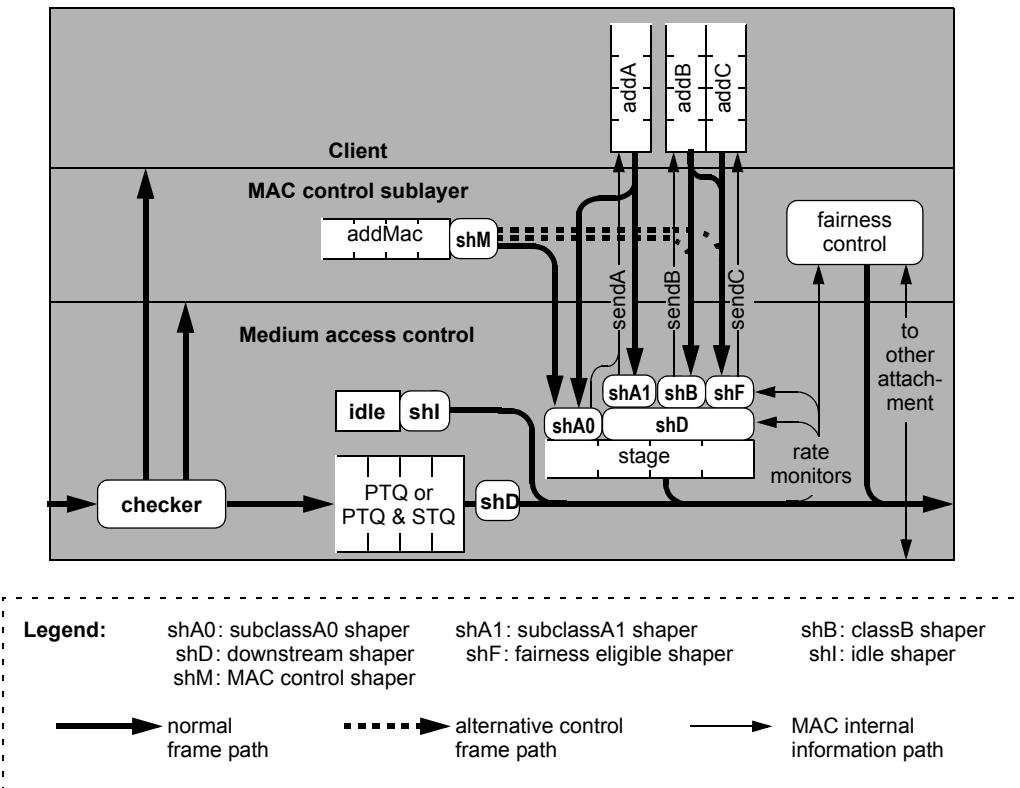
7.4 Datapaths

The MAC supports several paths along which frames move. Frames added directly by the MAC or by the MAC indirectly from the client are transmitted via the add paths, which include the add queues. Frames received and intended to be retransmitted are transmitted via the transit path(s), which include the transit queue(s). These paths collectively form the datapaths of the MAC and are illustrated in Figure 7.3, and they are further described in 7.4.1–7.4.4. Each set of datapaths is duplicated per ringlet.

7.4.1 Add paths

The client's labeling of its frames as classA, classB, or classC informs the MAC which shaper to apply to the added frames (see 7.5). The classA client add traffic flows through the subclassA0 shaper or the subclassA1 shaper, as determined by the MAC. The classB client add traffic flows through the classB shaper or the fairness eligible shaper, as determined by the MAC. The classC client add traffic flows through the fairness eligible shaper.

Accepted client traffic is placed into a stage queue, one frame at a time (see 7.7.4). The stage queue is a logical construct that might or might not correspond to any physical structure.

**Figure 7.3—MAC datapaths**

Frames added by the MAC control sublayer are shaped by the MAC control shaper. They are usually labeled subclassA0 and shaped by the subclassA0 shaper, but they can be labeled classB or classC (e.g., see 12.1.4 and 12.1.5) and directed to the classB shaper or the fairness eligible shaper, respectively, as illustrated by the dotted lines in Figure 7.3. Shaping by the MAC control shaper is applied first and in addition to the other shapers.

All shapers other than the downstream shaper are applied only to add traffic. The transit traffic is limited only by the downstream shaper. The transit traffic is monitored for the purpose of adjusting the fairness eligible shaper and the downstream shaper in addition to their updates based on add traffic.

NOTE—To the extent that an implementation does not present the highest priority control traffic at the head of the addMac queue, the MAC could experience adverse head-of-line blocking of its control traffic.

All unreserved add and transit traffic flows through the downstream shaper (see 7.5.8).

Optionally, rate synchronization idle frames are added by the MAC and shaped by the idle shaper (see 7.5.3).

7.4.2 Transit paths

A MAC transits frames that do not originate or terminate at that MAC. There are two types of MAC transit queueing designs: single queue and dual queue (see 7.7.7 and 7.7.8). The single queue design places all transit traffic into a primary transit queue (PTQ). The dual queue design places classA transit traffic into a higher precedence primary transit queue and classB and classC transit traffic into a lower-precedence secondary transit queue (STQ). Neither path supports preemption of either the transit or ingress frames.

Once a frame has begun transmission, its transmission cannot be interrupted by the transmission of another frame.

The correspondence between the services provided by the MAC and the transit paths used is summarized in Table 7.2, for implementations using single or dual transit queues.

Table 7.2—Service mapping

Implementation type	Service classes	Transit queue
single queue	classA, classB, and classC	PTQ
dual queue	classA	PTQ
	classB and classC	STQ

The delay through an unloaded transit path (a transit path with no frames in the transit queue(s) and no frames being added by the local station) shall be no more than the value that would correspond to the transmission of four frames with length of *mtuSize*. The delay is specified from first bit in to first bit out. The rest of the delays are not specified.

7.4.3 Passthrough mode

When a station determines (e.g., through self-consistency checks or heartbeat monitors) that its MAC sublayer is no longer able to reliably perform the procedures specified for the MAC, the station can enter the passthrough mode, where it acts similarly to a repeater. If a station is operating in a passthrough mode, it is no longer recognized as a station on the ring. The means by which a MAC enters passthrough, if at all, is a local implementation issue. Passthrough mode is an optional behavior.

NOTE—The transit path and the PHY should be believed to be operational in order for a station to enter passthrough. What constitutes reason for entry into passthrough mode is an implementation decision, within the bounds that the station must still be able to perform the actions required to support the transit path.

When in passthrough mode, the following behaviors shall be mandatory:

- a) If a station is wrapped, it unwraps when it enters passthrough mode.
- b) A station can not wrap while in passthrough mode.
- c) All data, control, and fairness frames received on a given ringlet are retransmitted without any modification on the same ringlet. This has the following implications:
 - 1) *ttl* is not decremented.
 - 2) The *ps* bit is not set.
 - 3) Frames are not checked for errors and are not discarded.
- d) Idle frames continue to be discarded when received.
- e) If the idle function is supported, idle frames continue to be added.
- f) Data, control, and fairness frames are not added.
- g) Transited frames continue to be queued into the appropriate transit queue and dequeued in priority order (not necessarily in strict FIFO order as would happen with a repeater).

7.4.4 Protection datapaths

Stations can support steering, center wrap, or edge wrap protected datapaths, as illustrated in Figure 7.4. Both the receive and transmit portions of stations are affected by the type of protection that a station

supports (*myProtectMethod*) and the current protection state of the span being received from or transmitted to (*myEdgeState*). The effects of the different protection options are summarized in this subclause.

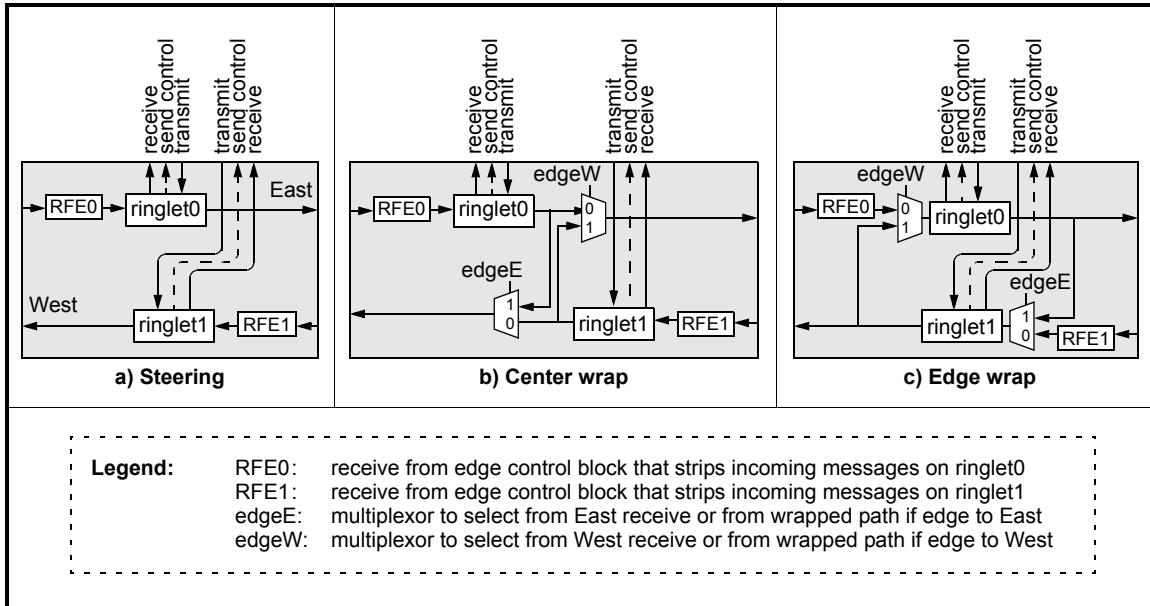


Figure 7.4—Wrap methods

Steering of frames is controlled by the existence of an edge between the source and the destination. Frames are steered away from the intended datapath if the station is a steering station (as determined by *protConfig*—see 11.2.5), an edge exists between the source and the destination (as determined by *Reachable()*—see 7.7.1.6.3), and the frame is eligible to be resteered (as determined by the *mac_protection* parameter—see 6.4.1).

Wrapping of frames is controlled by several conditions. Frames are wrapped if the station is a wrapping station (as determined by *protConfig*—see 11.2.5), the local station is wrapped (as determined by *myEdgeState*—see 7.4.4.6), and the frame is eligible to be wrapped (as determined by the *we* field—see 9.6.5).

Each wrapping-capable station has wrappable datapaths that allow frames to loop back to the opposing ringlet after link failures are detected. There are two methods of implementing wrapping, each with different effects on client add traffic behavior, and each providing different forms of possible redundancy. Wrapping systems can choose either method.

Frames are not transmitted across an edge, with two exceptions. The exceptions are fairness frames and topology and protection frames transmitted from a station on an edge to the neighbor on the other side of the edge span. These two frame types are explicitly sent into the edge to allow correct workings of the topology and protection protocols. If these frames succeed in crossing the edge span, they are stripped at the neighbor station (see 7.6.3.6). All other frames are either discarded (if not wrap eligible) or wrapped (if wrap eligible) at an edge.

7.4.4.1 Steering datapaths

For steering implementations, protection is provided at the source, not at the edge. Protection is provided by steering frames away from any edges that they could encounter. The steering decision is made as part of ringlet selection (see 7.7.1).

No steering-specific adjustments are made to frames during transit.

The receive from edge control block of a steering datapath is actively stripping incoming frames only when the remainder of the datapath is not being used. Its function could be implemented by the datapath implementation on the same ringlet, but it is illustrated as a distinct block for clarity.

Steering has no effect on the local transmission of topology and protection frames and fairness frames, which continue to be sent out both PHYs even when one PHY is transmitting into an edge.

7.4.4.2 Center wrap datapaths

A center wrapping station is a station that implements wrapping in the center of the datapath. For center wrap implementations, illustrated in Figure 7.4-b, frames are looped from the ringlet heading into the edge to the opposite ringlet's PHY RS. This has the effect of isolating the opposite ringlet's datapath from the client interface.

For wrapped center-wrap stations, add traffic is resteeled via ringlet selection, as is done for a steering node (see 7.7.1).

The receive from edge control block of a center wrap datapath is actively stripping incoming frames only when the remainder of the datapath is not being used. Its function could be implemented by the datapath implementation on the same ringlet, but it is illustrated as a distinct block for clarity.

Center wrapping has no effect on the local transmission of topology and protection frames and fairness frames, which continue to be sent out both PHYs even when one PHY is transmitting into an edge.

7.4.4.3 Center wrap nonuniformities

Center wrap nonuniformities require the wrap and rewrap adjustments illustrated in Table 7.3.

7.4.4.4 Edge wrap datapaths

An edge wrapping station is a station that implements wrapping on the edges of the datapath (as opposed to an edge station, which is any station adjacent to an edge, as defined in 5.1.7). For edge wrap implementations, illustrated in Figure 7.4-c, the data stream is looped back to the opposite ringlet, and participates in the opposite-ringlet datapath processing. This has the effect of allowing the client to access both datapaths without any ringlet selection adjustments.

The receive from edge control block of an edge wrap datapath is actively stripping incoming frames while the remainder of the datapath is also being used. Its function could not be implemented by the datapath implementation on the same ringlet.

Edge wrapping has no effect on the local transmission of topology and protection frames, which continue to be sent out both PHYs even when one PHY is transmitting into an edge. When transmitting into an edge, edge wrapped stations send one copy of any single-choke fairness frame (SCFF) into the edge and transmit another copy of the same frame into the wrap.

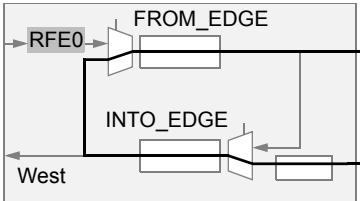
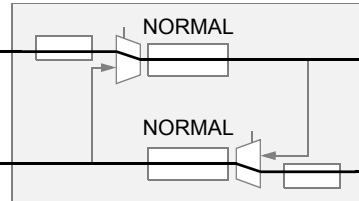
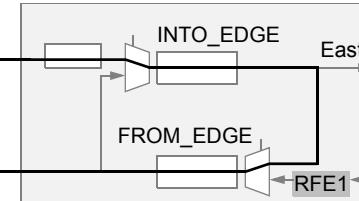
Table 7.3—Center wrap adjustments

Rewrap	Intermediate stations	Wrap
// Special discard rules discard = frame.ri != myRi && myEdgeState == INTO_EDGE && frame.sa != myMacAddress && frame.ps == 0	// Special discard rules —	// Special discard rules discard = frame.ri == myRi && myEdgeState == INTO_EDGE && frame.sa != myMacAddress && frame.ps == 1
// Special copy rules copy = frame.da == myMacAddress Multicast(frame.da) (myRxFilter == RX_FLOOD && frame.fi != FI_NONE)	// Special copy rules —	// Special copy rules copy = frame.da == myMacAddress Multicast(frame.da) (myRxFilter == RX_FLOOD && frame.fi != FI_NONE)
// Special adjust rules // Frames returning to source are // stripped at the wrap/rewrap points, // hence no need to set the PS bit in a // transit frame	// Special adjust rules frame.ps = frame.ri != myRi && myEdgeState == NORMAL && sa == myMacAddress	// Special adjust rules // Frames returning to source are // stripped at the wrap/rewrap points, // hence no need to set the PS bit in a // transit frame
// Special transmit rules if (frame.so) { frame.ri = ringlet_id; frame.ps = (frame.ri == MyEdgeRinglet()); } else { frame.ri = Other(MyEdgeRinglet()); frame.ps = 0; }	// Special transmit rules frame.ps = 0	// Special transmit rules if (frame.so) { frame.ri = ringlet_id; frame.ps = (frame.ri == MyEdgeRinglet()); } else { frame.ri = Other(MyEdgeRinglet()); frame.ps = 0; }
// Special wrong ringlet rules SameSide = (myProtectMethod == CENTER_WRAP && myEdgeState == INTO_EDGE) frame.ri == myRi		

7.4.4.5 Edge wrap nonuniformities

Edge wrap nonuniformities require the wrap and rewrap adjustments illustrated in Table 7.4. The boxes labeled “RFE0” and “RFE1” represent the receive from edge (RFE) functionality illustrated in Figure 7.4 and specified in 7.6.3.6. They are shown explicitly here to emphasize that unlike the steering or center wrap implementations, the receive from edge checks are not done in conjunction with the normal receive processing on the datapath.

Table 7.4—Edge wrap adjustments

Rewrap	Intermediate stations	Wrap
		
// Special discard rules discard = frame.ri != myRi && myEdgeState == INTO_EDGE && frame.sa != myMacAddress && frame.ps == 0	// Special discard rules —	// Special discard rules discard = frame.ri == myRi && myEdgeState == INTO_EDGE && frame.sa != myMacAddress && frame.ps == 1
// Special adjust rules		
frame.ps = myEdgeState == NORMAL && frame.ri != myRi && sa == myMacAddress		
// Special transmit rules		
frame.ps = 0		
// Special wrong ringlet rules		
SameSide = (frame.ri == myRi)		

7.4.4.6 *myEdgeState* setting

The frame receive operation state machines require knowledge of the relative position of any local fault. This knowledge is maintained by the *myEdgeState* variable, which indicates the edge status that the receive operation state machines are operating in. The values of the *myEdgeState* variable and the conditions that lead to the values are specified in Table 7.5.

Table 7.5—*myEdgeState* setting

Failure location or type	Resulting <i>myEdgeState</i> value	
	ringlet0 datapath	ringlet1 datapath
EAST	INTO_EDGE	FROM_EDGE
WEST	FROM_EDGE	INTO_EDGE
passthrough	PASSTHROUGH	PASSTHROUGH
—	NORMAL	NORMAL

7.5 Rate control

All add traffic (and some transit traffic) is rate controlled to maintain service class guarantees. This also has the effect of limiting the strict precedence of transmit decisions such that each service class gets its fair share of transmissions.

7.5.1 MAC shaper overview

The shapers described in this clause generate the *sendA* (see 7.5.5), *sendB* (see 7.5.6), and *sendC* (see 7.5.7.5) indications that are supplied to the client, and the *sendI* (see 7.5.3), *sendM* (see 7.5.4), and *sendD* (see 7.5.8) indications supplied to the MAC control sublayer and the MAC datapath sublayer. All shapers and indications operate on a per-ringlet basis.

Although multiple shapers are used within this standard, the behaviors of most shapers can be characterized by a common algorithm with instance-specific parameters. The shapers' credits are adjusted down or up, as illustrated in Figure 7.5. The decrement and increment values typically represent sizes of a transmitted frame and of credit increments in each update interval, respectively.

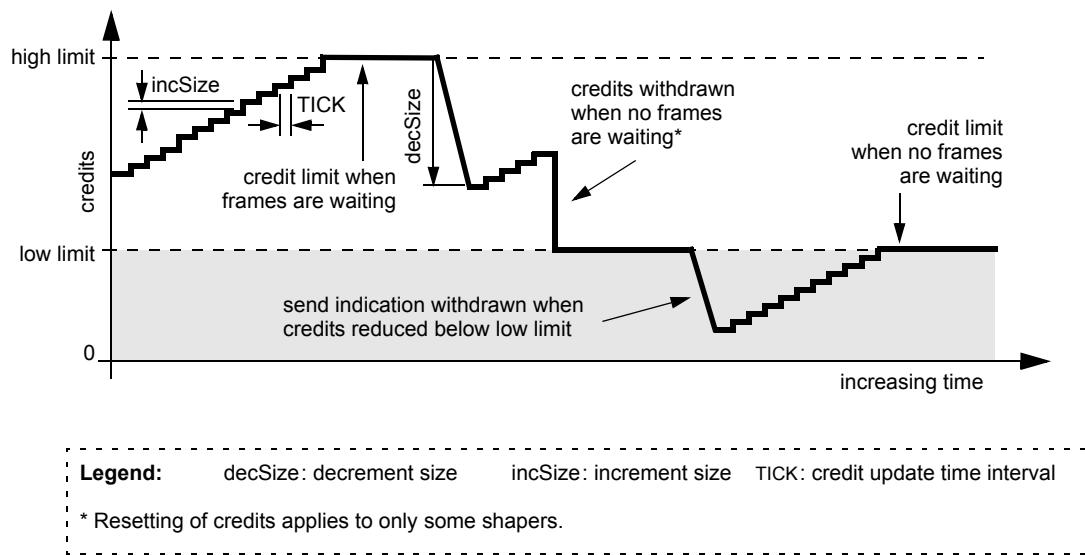


Figure 7.5—Credit adjustments over time

Crossing below the low-limit threshold shall generate a rate-limiting indication (the removal of a send indication), so that offered traffic can stop before reaching zero credits, where excessive transmissions are rejected. To bound the burst traffic after inactivity intervals, when no frames are ready for transmission, credits are reduced to the low limit (if currently higher than the low limit) and can accumulate to no more than the low limit. For most shapers described in this clause, the low limit is set to *mtuSize* in order to allow the transmission of a full-sized frame without reducing the credits below zero.

The high-limit threshold limits the positive credits, to avoid overflow. When frames are ready for transmission (and are being blocked by transit traffic), credits can accumulate to no more than the high limit. The high-limit value shall be at least *mtuSize*. If set to exactly *mtuSize*, the worst-case small-frame burst is minimal (no larger than one of the largest frames).

Most of the shapers consist of a token bucket. The token bucket has a default maximum depth of at least *mtuSize* bytes. The credits in the token bucket are incremented by the increment size at every time credit

update time interval. The maximum number of credits that can accumulate in the token bucket depends on the shaper. When a frame is waiting for access to the ring at the head of a queue, it is granted ring access only if the number of credits in the token bucket is greater than or equal to the low limit.

The number of credits in a token bucket is decremented by the size of each frame while it is being transmitted. Credits shall be reduced as each byte is sent, or at least as frequently as every 256 bytes, and always at the end of each frame transmission. It should never be possible for the number of credits to become negative after subtracting the number of bytes in a transmitted frame.

The shaper algorithms are conceptual, with no implied implementation. Shapers shall have accuracy of at least 5% of the expected credits.

7.5.2 Add queue flow control

The MAC does not maintain queues for shaping the client's add traffic, but it provides per-ringlet, per-class flow control indications to the client. Transmission of client traffic is enabled with the send indications listed in Table 7.6 and disabled otherwise. The MAC does not accept requests from the client when the appropriate send indication is not present. The details of this interaction are implementation specific. Flow control indications are required to be sent to the client only when the values of *sendA*, *sendB*, and *sendC* change (see Table 6.7).

Table 7.6—Send indication summary

Description	Send indication	Subclause
idle transmissions	<i>sendI</i>	7.5.3
MAC control allocated transmissions	<i>sendM</i>	7.5.4
classA allocated transmissions	<i>sendA</i>	7.5.5
classB allocated transmissions	<i>sendB</i>	7.5.6
fairness eligible transmissions	<i>sendC</i>	7.5.7.5
sustain downstream subclassA0 transmissions	<i>sendD</i>	7.5.8

In addition to the above, all classes of traffic are effectively rate limited by the transmission selection algorithm (see 7.7.7 or 7.7.8).

Some of the transmission paths are affected by only one of these shapers; others are influenced by multiple shapers.

Optional idle frames from the MAC rate synchronization add queue are shaped by the idle shaper (see 7.5.3).

Frames from the MAC control add queue are shaped by the MAC control shaper (see 7.5.4). Control frames are also shaped by the shaper for the service class chosen, described below. As control traffic has precedence over client traffic, the class-based shapers effectively throttle only the client's traffic.

All classA add traffic is shaped by the classA shaper (see 7.5.5) to avoid having the client exceed its classA allocated rates. The classA shaper is logically partitioned into the subclassA0 subshaper and the subclassA1 subshaper.

All classB add traffic is shaped by the classB shaper (see 7.5.6) and/or by the fairness eligible shaper, to constrain the client within its classB-CIR and classB-EIR rates.

All classC add traffic is shaped by the fairness eligible shaper (see 7.5.7.5), to constrain the client within its weighted fair-share use of the unused and reclaimable bandwidth.

All subclassA1, classB, and classC add traffic and all classB and classC transit traffic from a secondary transit queue (STQ) are additionally shaped by the downstream shaper (see 7.5.8), to constrain the MAC to sustain the downstream allocated subclassA0 rate.

7.5.3 IdleShaper state machine

RPR stations and networks can be implemented with either synchronously or asynchronously timed PHYs. In a synchronous network, the transmit clock for each station is locked to a common timing source that may be recovered from the received data stream or provided through an external timing interface, and the transmit data rate from each station is exactly identical to the received data rate. However, in an asynchronous network, the transmit data rate at each station is determined by a local clock source, and the transmit data rate from each station varies slightly from the network data rate. In the case of a station transmitting at a slower data rate than the preceding station, this could cause primary transit queue overflow.

The idle shaper describes an optional transmit rate synchronization function that eliminates the possibility of primary transit queue overflow by inserting a variable number of small idle frames in the transmitted data stream. The transmit rate synchronization function is part of the MAC datapath. Although designed for operation with asynchronously timed stations, this function can be included for all implementations, including those intended for synchronous network operation.

The transmit rate synchronization function supports PHYs with data rate clock tolerances of up to ± 100 PPM.

The plesiochronous clock synchronization method that forms the basis of the transmit rate synchronization function is further described in Bates [B4].

The idle shaper limits the MAC-supplied idle traffic to its allocated limits, based on the parameters listed in Table 7.7. Idle frames are inserted into the transmit datapath at a fixed rate equivalent to 0.05% of the *lineRate*. The rate of inserted idle frames is controlled by the level of the primary transit queue (PTQ). As the queue fills, the free space becomes less than the *idleThreshold* value. At this threshold, the idle frame rate reduces to 0.025% of the *lineRate*, which increases the transmit data rate, which eventually reduces the queue depth.

The idle shaper may optionally be implemented. If the idle shaper is not implemented, then no idle frames are generated.

The following subclauses describe parameters used within the context of this state machine.

7.5.3.1 IdleShaper state machine definitions

IDLE_SIZE

See 7.2.4.

TICK

See 7.2.1.

7.5.3.2 IdleShaper state machine variables

addRateI

The calculated rate for MAC idle add traffic, in units of bytes per second.

creditI

The total current credits accumulated by the idle shaper, in units of bytes.

currentTime

See 7.2.2.

hiLimitI

The high threshold for the idle shaper credits.

Default: $loLimitI + IDLE_SIZE$.

idleThreshold

The threshold that indicates that a MAC is experiencing an adverse rate mismatch. When the available space in the PTQ is less than or equal to this value, the incoming link is considered adversely rate mismatched.

Range: $[mtuSize, sizePtq - mtuSize]$, in bytes

Default: *mtuSize*

lineRate

See 7.2.4.

loLimitI

The low threshold for the idle shaper credits.

Value: *IDLE_SIZE*.

mtuSize

See 7.2.2.

sendI

See 7.2.2.

sizePtq

See 7.2.2.

tickTime

A value representing the start time of the current tick processing interval.

7.5.3.3 IdleShaper state machine routines

Min(value1, value2)

See 7.2.3.

SpaceInPtq()

See 7.2.3.

7.5.3.4 IdleShaper state table

The IdleShaper state machine generates the *sendI* control signal, as specified in Table 7.7. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.7—IdleShaper state table

Current state		Row	Next state	
state	condition		action	state
START	currentTime – tickTime >= TICK	1	creditI = Min(hiLimitI, creditI+addRateI*(currentTime-tickTime)); tickTime = currentTime;	START
	—	2	sendI = (creditI >= loLimitI);	CALC
CALC	SpaceInPtq() > idleThreshold	3	addRateI = (lineRate/8)*500/1000000;	START
	—	4	addRateI = (lineRate/8)*250/1000000;	

Row 7.7-1: The *creditI* accumulations occur once every update interval. And the *tickTime* is updated at the start of each update interval. (*creditI* is decremented by transmitted idle frames in Table 7.34, Row 3 and Table 7.35, Row 3.)

Row 7.7-2: The *sendI* indication is updated.

Row 7.7-3: The idle transmission rate is a fixed rate equivalent to 0.05% of the *lineRate*, expressed in bits per second. The *lineRate* value is transformed to bytes per second for usage in the calculation.

Row 7.7-4: As the PTQ fills, the rate is reduced to 0.025% of the *lineRate*, expressed in bits per second. Reduced idles increases the non-idle transmit data rate, which eventually reduces the PTQ queue depth. The *lineRate* value is transformed to bytes per second for usage in the calculation.

7.5.4 MacControlShaper state machine

The control shaper is run in addition to the other shapers through which control frames pass (i.e., the subclassA0 shaper, the classB shaper, and the fairness eligible shaper). A control frame debits credits from both shapers through which it passes. Accordingly, implementations should account for this by setting the rates of the subclassA0 shaper, the classB shaper, and the fairness eligible shaper such that they include enough bandwidth for the anticipated control traffic.

The following subclauses (7.5.4.1–7.5.4.4) describe parameters used within the context of this state machine.

7.5.4.1 MacControlShaper state machine definitions

CONTROL_MAX

See 7.2.4.

TICK

See 7.2.1.

7.5.4.2 MacControlShaper state machine variables

addRateM

The allocated rate for MAC control add traffic, in units of bytes per second.

creditM

The total current credits accumulated by the MAC control shaper, in units of bytes.

currentTime

See 7.2.2.

hiLimitM

The high threshold for the MAC control shaper credits. The default value is *loLimitM* + **CONTROL_MAX**.

loLimitM

The low threshold for the idle shaper credits.

Value: CONTROL_MAX.

sendM

See 7.2.2.

tickTime

A value representing the start time of the current tick processing interval.

7.5.4.3 MacControlShaper state machine routines

Min(value1, value2)

See 7.2.3.

7.5.4.4 MacControlShaper state table

The MacControlShaper state machine generates the *sendM* control signal, as specified in Table 7.8. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.8—MacControlShaper state table

Current state		Row	Next state	
state	condition		action	state
START	currentTime – tickTime >= TICK	1	creditM = Min(hiLimitM, creditM+addRateM*(currentTime-tickTime)); tickTime = currentTime;	START
	—	2	sendM = (creditM >= loLimitM);	

Row 7.8-1: The *creditM* accumulations occur once every *tickTime* interval. And the *tickTime* is updated at the start of each update interval. (*creditM* is decremented by transmitted control frames in Table 7.31, Row 19.)

Row 7.8-2: The *sendM* indication is updated.

7.5.5 ClassAShaper state machine

The following subclauses (7.5.5.1–7.5.5.4) describe parameters used within the context of this state machine.

7.5.5.1 ClassAShaper state machine definitions

TICK

See 7.2.1.

7.5.5.2 ClassAShaper state machine variables

addRateA0

The allocated rate for subclassA0 client add traffic, in units of bytes per second.

addrateA1

The allocated rate for subclassA1 client add traffic, in units of bytes per second.

classAAccessDelayTimer

classAAccesDelayTimerExpired

classAAccesDelayTimerThreshold

See 7.2.2.

creditA0

The total current credits accumulated by the subclassA0 shaper, in units of bytes.

creditA1

The total current credits accumulated by the subclassA1 shaper, in units of bytes.

currentTime

See 7.2.2.

hiLimitA0

The high threshold for the subclassA0 subshaper credits.

Default: specified in 7.5.5.5.

hiLimitA1

The high threshold for the subclassA1 subshaper credits.

Default: specified in 7.5.5.5.

lastCreditA0

The value of *creditA0* at the end of updating the credit accumulation, saved for comparison with *creditA0* at the beginning of the next update interval.

lastCreditA1

The value of *creditA1* at the end of updating the credit accumulation, saved for comparison with *creditA1* at the beginning of the next update interval.

loLimitA0

The low threshold for the subclassA0 shaper credits.

Value: *mtuSize*.

loLimitA1

The low threshold for the subclassA1 shaper credits.

Value: *mtuSize*.

mtuSize

See 7.2.4.

optionClassAAccesDelayTimer

Available for stations wishing to monitor add access delay for classA frames.

TRUE—Indicates that classA access delay is to be monitored.

FALSE—(Otherwise.)

passA0

passA1

sendA

sendD

See 7.2.2.

startingCreditA0

The value of *creditA0* before updating the credit accumulation, saved for comparison with *lastcreditA0* later in the state machine.

startingCreditA1

The value of *creditA1* before updating the credit accumulation, saved for comparison with *lastcreditA1* later in the state machine.

tickTime

A value representing the start time of the current tick processing interval.

7.5.5.3 ClassAShaper state machine routines

ClassAEntryAvailable()

Indicates if a classA data frame is ready to be transmitted, as specified by Equation (7.12).

TRUE—A classA data frame to transmit is available from the client.

FALSE—(Otherwise.)

(ExamineQueue(Q_TX_RS, FT_DATA, SC_CLASS_A) != NULL) (7.12)

$\text{Min}(\text{value1}, \text{value2})$

See 7.2.3.

7.5.5.4 ClassAShaper state table

The ClassAShaper state machine generates the *sendA* control signal, as specified in Table 7.9. Optional states are shaded in gray. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.9—ClassAShaper state table

Current state		Row	Next state	
state	condition		action	state
START	currentTime - tickTime >= TICK	1	startingCreditA0 = creditA0; startingCreditA1 = creditA1; creditA0 = Min(hiLimitA0, creditA0 + addRateA0 * (currentTime - tickTime)); creditA1 = Min(hiLimitA1, creditA1 + addRateA1 * (currentTime - tickTime));	TICK
	—	2	passA0 = creditA0 >= loLimitA0; passA1 = creditA1 >= loLimitA1; sendA = passA0 (passA1 && sendD);	START
TICK	!ClassAEntryAvailable() && startingCreditA0 == lastCreditA0 && startingCreditA1 == lastCreditA1	3	creditA0 = Min(loLimitA0, creditA0); creditA1 = Min(loLimitA1, creditA1); classAAccessDelayTimer = 0;	SAVE
	ClassAEntryAvailable() && startingCreditA0 == lastCreditA0 && startingCreditA1 == lastCreditA1	4	classAAccessDelayTimer += currentTime - tickTime;	TIMER
	—	5	classAAccessDelayTimer = 0;	SAVE
TIMER	optionClassAAccesDelayTimer && classAAccesDelayTimer >= classAAccesDelayTimerThreshold	6	classAAccesDelayTimerExpired = TRUE;	SAVE
	—	7	classAAccesDelayTimerExpired = FALSE;	
SAVE	—	8	lastCreditA0 = creditA0; lastCreditA1 = creditA1; tickTime = currentTime;	START

Row 7.9-1: The credit amount is saved for the comparison in the TICK state. The *creditA0* and *creditA1* accumulations are incremented every update interval. (*creditA0* is decremented by subclassA0 add traffic in Table 7.31, Row 5. *creditA1* is decremented by subclassA1 add traffic in Table 7.31, Row 6.)

Row 7.9-2: The *sendA* indication is set according to whether credits are sufficient.

Row 7.9-3: The *creditA0* and *creditA1* accumulations are reset if no frames were waiting to be transmitted during the last update interval. (If no frames were transmitted, as determined by no decrease in the credits

during the last interval, and none are waiting now, then none were waiting to be transmitted at any time during the last update interval.) The accumulations are reset only if they are higher than the low limit; otherwise they are allowed to accumulate up to the low limit. Also, as no classA frames are waiting, the classA access delay timer is reset.

Row 7.9-4: A classA entry is available and no classA bytes have been transmitted during the last TICK time. The classA access delay timer is adjusted for the amount of time classA frames were waiting to be transmitted and were not able to be transmitted (assumed to be the entire TICK time).

Row 7.9-5: Otherwise, the *creditA0* and *creditA1* accumulations are allowed to continue to accumulate, and the classA access delay timer is reset.

Row 7.9-6: (optional) ClassA frames have waited too long to be transmitted.

Row 7.9-7: (optional) ClassA frames have not waited too long.

Row 7.9-8: The credit amounts are saved for the comparison at the next update interval. And the *tickTime* is updated after each update interval.

7.5.5.5 ClassA high limit credit values

The default values for *hiLimitA0* and *hiLimitA1* are set to preserve the delay and jitter guarantees of classA traffic. The default values are given by Equation (7.13) through Equation (7.15).

$$hiLimitA0 = mtuSize + addRateA0 * MAX_JITTER / 2 \quad (7.13)$$

$$hiLimitA1 = mtuSize + addRateA1 * MAX_JITTER / 2 \quad (7.14)$$

$$MAX_JITTER = (numStations * (mtuSize / lineRate)) / (1 - (addRateA0 / lineRate)) \quad (7.15)$$

NOTE 1—The rates in this formula are converted to bytes/second for this calculation.

NOTE 2—MAX_JITTER is determined by the possible coincidental transmission of frames by all stations in the ring, (*numStations* * (*mtuSize* / *lineRate*)), and the rate at which the station can recover, (1 - (*addRateA0* / *lineRate*)). For a local single transit queue implementation, any frames can affect MAX_JITTER. For a local dual transit queue implementation, any classA frames can affect MAX_JITTER.

NOTE 3—*hiLimitA0* and *hiLimitA1* are determined by the amount of credits that would be needed to buffer the amount of traffic that could be generated by MAX_JITTER, plus one MTU to account for respective *loLimitA0* and *loLimitA1* values.

NOTE 4—Neither *addRateA0* nor (*addRateA0* + *addRateA1*) should be allowed to approach *lineRate* because this causes delay and jitter bounds to approach infinity.

NOTE 5—Assuming neither *addRateA0* nor *addRateA1* approach *lineRate*, the highest values of *hiLimitA0* and *hiLimitA1* are not reached, and therefore the bandwidth and jitter guarantees are preserved.

7.5.6 ClassBShaper state machine

The following subclauses describe parameters used within the context of this state machine.

7.5.6.1 ClassBShaper state machine definitions

TICK

See 7.2.1.

7.5.6.2 ClassBShaper state machine variables

addRateB

The allocated rate for classB-CIR client add traffic, in units of bytes per second.

classBAccessDelayTimer

classBAccessDelayTimerExpired

classBAccessDelayTimerThreshold

See 7.2.2.

creditB

The total current credits accumulated by the classB shaper, in units of bytes.

currentTime

See 7.2.2.

hiLimitB

The high threshold for the classB shaper credits.

Default: specified by Equation (7.17).

lastCreditB

The value of *creditB* at the end of updating the credit accumulation, saved for comparison with *creditB* at the beginning of the next update interval.

loLimitB

The low threshold for the classB shaper credits.

Value: *mtuSize*.

mtuSize

See 7.2.4.

optionClassBAccessDelayTimer

Available for stations wishing to monitor add access delay for classB frames.

TRUE—Indicates that classB access delay is to be monitored.

FALSE—(Otherwise.)

sendB

sendD

See 7.2.2.

startingCreditB

The value of *creditB* before updating the credit accumulation, saved for comparison with *lastcreditB* later in the state machine.

tickTime

A value representing the start time of the current tick processing interval.

7.5.6.3 ClassBShaper state machine routines

ClassBEntryAvailable()

Indicates if a classB data frame is ready to be transmitted, as specified by Equation (7.16).

TRUE—A classB data frame to transmit is available from the client.

FALSE—(Otherwise.)

(ExamineQueue (Q_TX_RS, FT_DATA, SC_CLASS_B) != NULL) (7.16)

Min(value1, value2)

See 7.2.3.

7.5.6.4 ClassBShaper state table

The ClassBShaper state machine generates the *sendB* control signal, as specified in Table 7.10. Optional states are shaded in gray. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.10—ClassBShaper state table

Current state		Row	Next state	
state	condition		action	state
START	currentTime – tickTime >= TICK	1	startingCreditB = creditB; creditB = Min(hiLimitB, creditB + addRateB * (currentTime - tickTime));	TICK
	—	2	sendB = (creditB >= loLimitB) && sendD;	START
TICK	!ClassBEntryAvailable() && startingCreditB == lastCreditB	3	creditB = Min(loLimitB, creditB); classBAccessDelayTimer = 0;	SAVE
	ClassBEntryAvailable() && startingCreditB == lastCreditB	4	classBAccessDelayTimer += currentTime - tickTime;	TIMER
	—	5	classBAccessDelayTimer = 0;	SAVE
TIMER	optionClassBAccessDelayTimer && classBAccessDelayTimer >= classBAccessDelayTimerThreshold	6	classBAccessDelayTimerExpired = TRUE;	SAVE
	—	7	classBAccessDelayTimerExpired = FALSE;	
SAVE	—	8	lastCreditB = creditB; tickTime = currentTime;	START

Row 7.10-1: The credit amount is saved for the comparison in the TICK state. The *creditB* accumulation is incremented every update interval. (*creditB* is decremented by classB add traffic in Table 7.31, Row 10.)

Row 7.10-2: The *sendB* permission is granted in the presence of sufficient classB-CIR credits.

Row 7.10-3: The *creditB* accumulation is reset if no frames were waiting to be transmitted during the last update interval. (If no frames were transmitted, as determined by no decrease in the credits during the last interval, and none are waiting now, then none were waiting to be transmitted at any time during the last update interval.) The accumulation is reset only if it is higher than the low limit; otherwise it is allowed to accumulate up to the low limit. Also, as no classB frames are waiting, the classB access delay timer is reset.

Row 7.10-4: A classB entry is available and no classB bytes have been transmitted during the last TICK time. The classB access delay timer is adjusted for the amount of time classB frames were waiting to be transmitted and were not able to be transmitted (assumed to be the entire TICK time).

Row 7.10-5: Otherwise, the *creditB* accumulations are allowed to continue to accumulate, and the classB access delay timer is reset.

Row 7.10-6: (optional) ClassB frames have waited too long to be transmitted.

Row 7.10-7: (optional) ClassB frames have not waited too long.

Row 7.10-8: The credit amount is saved for the comparison at the next update interval. And the *tickTime* is updated after each update interval.

The default *hiLimitB* value is set to an estimate of the amount of data that can be sent in the time before a fairness adjustment is made. The default value is given by Equation (7.17).

$$hiLimitB = (2 * numStations * mtuSize + (FRTT * lineRate)) * (addRateB / lineRate) \quad (7.17)$$

NOTE 1—The rates in this formula are converted to bytes/second for this calculation.

NOTE 2—FRTT is expressed in seconds for this calculation.

NOTE 3—A calculation for FRTT is given in Equation (G.3).

NOTE 4—The first half of the formula is the amount of accumulated frames that can be received during the fairness response time. The second half of the equation is the fraction of that time that is desired to be used to add classB frames from the local station.

NOTE 5—*mtuSize* is chosen in Equation (7.17) because this is the value of *loLimitB*.

NOTE 6—*hiLimitB* could also be set the same as *hiLimitA1* with the exception of replacing “*addRateA1*” with “*addRateA1+addRateB*” because a dual queue implementation can utilize the STQ to reduce the jitter of classB.

7.5.7 Fairness eligible shaper state machines

The fairness eligible shaper consists of the following state machines:

- a) PreCongestionShaper state machine: adjusts credits for fairness eligible add traffic traveling short of and/or beyond the congestion point.
- b) PostCongestionShaper state machine: adjusts credits for fairness eligible add traffic traveling beyond the congestion point.
- c) SourceShaper state machine: provides an indication that can be used to regulate between fairness eligible add traffic and STQ transit traffic.
- d) FairnessEligibleIndication state machine: provides the *sendC* indication.

The PreCongestionShaper, PostCongestionShaper, and SourceShaper state machines are applicable only if using the SHAPER_BASED admission method (see 10.4.1) of the fairness algorithm. Their results are not used if using the RATE_BASED admission method of the fairness algorithm.

7.5.7.1 MAC congestion calculations

Single queue and dual queue implementations calculate and detect congestion in different manners. The exact conditions for determining if congestion exist are specified in Clause 10.

Single queue implementations measure congestion with access delay timers and with link utilization. There are separate access delay timers for classA, classB, and classC traffic, each maintained for each ringlet. The timer corresponding to the class of frame is started at the time the frame becomes the head-of-line frame in the MAC. The timer is reset when the MAC successfully transmits that frame. For classA frames, if the MAC is unable to transmit the frame before the timer expires, the MAC treats this as a reportable provisioning error. For classB and classC frames, if the MAC is unable to transmit the frame before the timer expires, the MAC treats this the same as having exceeded the *stqLowThreshold* condition as described below.

The value of the classA access delay timer should be based on the guaranteed jitter for classA traffic. The default value is given by Equation (7.18), where *numStations* and *mtuSize* are contained within the *ringInfo* collection of topology variables. The range is from 100 microseconds to 25.5 milliseconds, in 100-microsecond increments.

$$\text{Min}(\text{Round}100(2 * \text{numStations} * \text{mtuSize} * 8 / \text{lineRate}), 25.5) \quad (7.18)$$

The values of the classB and classC access delay timers should be based on the acceptable MAC end-to-end delay for classB and classC traffic. The default value is 1.00 milliseconds for classB and 10.00 milliseconds for classC. The range is from 100 microseconds to 25.5 milliseconds, in 100-microsecond increments.

Dual queue implementations measure congestion with STQ thresholds. There are both a low threshold, *stqLowThreshold*, and high threshold, *stqHighThreshold*, maintained for each ringlet. A threshold is indicated to have been exceeded whenever the number of bytes queued in the STQ is greater than the threshold value.

7.5.7.2 PreCongestionShaper state machine

The precongestion shaper indirectly shapes fairness eligible traffic traveling short of and/or beyond the congestion point. Its output, *passAddFe*, is used in the generation of *addRateOK* (see 10.4.1), which in turn is used in the generation of the *sendD* indication (see 7.5.7.5).

7.5.7.2.1 PreCongestionShaper state machine definitions

TICK

See 7.2.1.

7.5.7.2.2 PreCongestionShaper state machine variables

advertisingInterval

See 7.2.4.

ageCoef

See 7.2.4.

allowedRate

See 7.2.4.

classCAccessDelayTimer

classCAccessDelayTimerExpired

classCAccessDelayTimerThreshold

See 7.2.2.

creditC

The total current credits accumulated by the precongestion fairness eligible shaper.

currentTime

See 7.2.2.

hiLimitC

The high threshold for the fairness eligible shaper credits.

Default: *loLimitC* + *mtuSize*.

lastCreditC

The value of *creditC* at the end of updating the credit accumulation, saved for comparison with *creditC* at the beginning of the next update interval.

loLimitC

The low threshold for the fairness eligible shaper credits.

Value: *mtuSize*.

mtuSize

See 7.2.2.

optionClassCAccessDelayTimer

Available for stations wishing to monitor add access delay for classC frames.

TRUE—Indicates that classC access delay is to be monitored.

FALSE—(Otherwise.)

passAddFe

See 7.2.2.

tickTime

A value representing the start time of the current tick processing interval.

7.5.7.2.3 PreCongestionShaper state machine routines

ClassCEntryAvailable()

Indicates if a classC data frame is ready to be transmitted, as specified by Equation (7.19).

TRUE—A classC data frame to transmit is available from the client.

FALSE—(Otherwise.)

(ExamineQueue(Q_TX_RS, FT_DATA, SC_CLASS_C) != NULL) (7.19)

FeAddByteAvailable()

See 7.2.4.

Max(value1, value2)

See 7.2.3.

Min(value1, value2)

See 7.2.3.

7.5.7.2.4 PreCongestionShaper state table

The PreCongestionShaper state machine generates the *passAddFe* indication, as specified in Table 7.11. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.11—PreCongestionShaper state table

Current state		Row	Next state	
state	condition		action	state
START	(currentTime – tickTime) >= TICK	1	—	TICK
	creditC >= loLimitC	2	passAddFe = TRUE;	AVAIL
	—	3	passAddFe = FALSE;	
AVAIL	FeAddByteAvailable()	4	creditC = Max(0, creditC - 1);	START
	—	5	—	
TICK	classCEEntryAvailable() && creditC == lastCreditC	6	classCAccessDelayTimer += currentTime - tickTime;	TIMER
	—	7	classCAccessDelayTimer = 0;	SAVE
TIMER	optionClassCAccessDelayTimer && classCAccessDelayTimer >= classCAccessDelayTimerThreshold	8	classCAccessDelayTimerExpired = TRUE;	SAVE
	—	9	classCAccessDelayTimerExpired = FALSE;	
SAVE	—	10	creditC = Min(hiLimitC, creditC + (allowedRate * 1/(ageCoef * advertisingInterval) * (currentTime - tickTime))); tickTime = currentTime; lastCreditC = creditC;	START

Row 7.11-1: An update interval has elapsed since the last *creditC* increment.

Row 7.11-2: Sufficient credits sets the *passAddFe* indication.

Row 7.11-3: Otherwise, the *passAddFe* indication is cleared.

Row 7.11-4: The *creditC* reductions occur while a fairness eligible frame is being added.

Row 7.11-5: Otherwise, wait for the next update interval or for a fairness eligible frame to be added.

Row 7.11-6: The classC access delay timer is adjusted for the amount of time fairness eligible frames were waiting to be transmitted and were not able to be transmitted.

Row 7.11-7: Otherwise, the classC access delay timer is reset.

Row 7.11-8: (optional) Fairness eligible frames have waited too long to be transmitted.

Row 7.11-9: (optional) Fairness eligible frames have not waited too long.

Row 7.11-10: The credit amount is saved for the comparison at the next update interval. The *creditC* accumulation is incremented every update interval. And the *tickTime* is updated after each update interval. The *allowedRate* value is transformed to bytes per second for usage in this calculation.

7.5.7.3 PostCongestionShaper state machine

The postcongestion shaper indirectly shapes fairness eligible traffic traveling beyond the congestion point. Its output, *passAddFeCongested*, is used in the generation of *addRateCongestedOK* (see 10.4.1), which in turn is used in the generation of the *sendD* indication (see 7.5.7.5).

7.5.7.3.1 PostCongestionShaper state machine definitions

TICK

See 7.2.1.

7.5.7.3.2 PostCongestionShaper state machine variables

advertisingInterval

See 7.2.4.

ageCoef

See 7.2.4.

allowedRateCongested

See 7.2.4.

creditCC

The total current credits accumulated by the postcongestion shaper.

currentTime

See 7.2.2.

hiLimitCC

The high threshold for the postcongestion shaper credits.

Default: *loLimitCC* + *mtuSize*.

loLimitCC

The low threshold for the postcongestion shaper credits.

Value: *mtuSize*.

mtuSize

See 7.2.2.

passAddFeCongested

See 7.2.2.

tickTime

A value representing the start time of the current tick processing interval.

7.5.7.3.3 PostCongestionShaper state machine routines

FeAddByteAvailable()

See 7.2.4.

Max(value1, value2)

See 7.2.3.

Min(value1, value2)

See 7.2.3.

7.5.7.3.4 PostCongestionShaper state table

The PostCongestionShaper state machine generates the *passAddFeCongested* indication, as specified in Table 7.12. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.12—PostCongestionShaper state table

Current state		Row	Next state	
state	condition		action	state
START	(currentTime – tickTime) >= TICK	1	creditCC = Min(hiLimitCC, creditCC + allowedRateCongested * 1/(ageCoef * advertisingInterval) * (currentTime – tickTime)); tickTime = currentTime;	START
	FeAddByteAvailable() && frame.ttl > hopsToCongestion	2	creditCC = Max(0, creditCC - 1);	
	creditCC >= loLimitCC	3	passAddFeCongested = TRUE;	
	—	4	passAddFeCongested = FALSE;	

Row 7.12-1: The *creditCC* accumulations occur once every update interval, and the *tickTime* is updated. The *allowedRateCongested* value is transformed to bytes per second for usage in the calculation.

Row 7.12-2: The *creditCC* reductions occur when a fairness eligible frame traveling beyond the congestion point is added.

Row 7.12-3: Sufficient credits set the *passAddFeCongested* indication.

Row 7.12-4: Otherwise, the *passAddFeCongested* indication is cleared.

7.5.7.4 SourceShaper state machine

The source shaper indirectly balances added fairness eligible traffic and transited STQ traffic. Its output, *passAddFeOrStq*, is used in the generation of *addRateOK* and *addRateCongestedOK* (see 10.4.1), which in turn are used in the generation of the *sendD* indication (see 7.5.7.5).

7.5.7.4.1 SourceShaper state machine definitions

No definitions are used by this state machine.

7.5.7.4.2 SourceShaper state machine variables

creditS

The total current credits accumulated by the source shaper.

frame

The contents of an RPR frame.

hiLimitS

The high threshold for the source shaper credits.

Default: $loLimitS + mtuSize$.

loLimitS

The low threshold for the source shaper credits.

Value: $mtuSize$.

passAddFeOrStq

See 7.2.2.

stqDepth

See 7.2.2.

stqHighThreshold

See 7.2.4.

7.5.7.4.3 SourceShaper state machine routines

*FeAddByteAvailable()**FeFwByteAvailable()*

See 7.2.4.

Max(value1, value2)

See 7.2.3.

7.5.7.4.4 SourceShaper state table

The SourceShaper state machine generates the *passAddFeOrStq* control signal, as specified in Table 7.13. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.13—SourceShaper state table

Current state		Row	Next state	
state	condition		action	state
START	FeAddByteAvailable()	1	creditS = Max(0, creditS - 1);	START
	FeFwByteAvailable()	2	creditS = Max(0, creditS + 1);	
	stqDepth == 0	3	creditS = hilimitS;	
	—	4	passAddFeOrStq = (creditS >= lolimitS && (stqDepth < stqHighThreshold);	

Row 7.13-1: The *creditS* reductions occur when a classB-EIR or classC frame is added.

Row 7.13-2: The *creditS* increases occur when a classB-EIR or classC frame is transited.

Row 7.13-3: Maximum *creditS* is assumed whenever the STQ has been emptied.

Row 7.13-4: The *passAddFeOrStq* indication is constantly generated.

7.5.7.5 FairnessEligibleIndication state machine

This subclause describes the flow-control indication generated by the MAC to control fairness eligible traffic (i.e., classB-EIR and classC traffic) to its allowed limits. Fairness eligible add traffic is shaped by the outputs of the fairness algorithm, not by a token bucket method as used by the other shapers.

Fairness eligible add traffic is rate controlled, via the *sendC* indication, to two rates. It is always limited to the *allowedRate* value calculated in Clause 10. In addition, any fairness eligible traffic that is passing the congestion point is also limited to the *allowedRateCongested* value calculated in Clause 10.

The *sendC* value is different from the *sendA* and *sendB* values in that it is not a boolean value, but it is an integer value that provides a maximum hop count that fairness eligible frames can travel. This allows an implementation to selectively throttle fairness eligible traffic based on the hop count to its target. A *sendC* indication of 0 indicates that the *allowedRate* value for the station has been exceeded, and therefore, fairness eligible frames cannot be sent (because any destination would be more than 0 hops away). A *sendC* indication of greater than 0 and less than the ring size indicates that the *allowedRateCongested* value through the congestion has been exceeded, and therefore, fairness eligible frames can be sent as far as the congestion point (whose hop-count distance is indicated by the *sendC* value). A *sendC* indication of (at least) the ring size indicates that no fairness rate has been exceeded, and therefore, fairness eligible frames can be sent as far as desired.

The following subclauses (7.5.7.5.1–7.5.7.5.3) describe parameters used within the context of this state machine.

7.5.7.5.1 FairnessEligibleIndication state machine definitions

MAX_STATIONS

See 7.2.4.

7.5.7.5.2 FairnessEligibleIndication state machine variables

addRateOK

addRateCongestedOK

See 7.2.4.

sendC

sendD

See 7.2.2.

7.5.7.5.3 FairnessEligibleIndication state machine routines

No routines are used by this state machine.

7.5.7.5.4 FairnessEligibleIndication state table

The FairnessEligibleIndication state machine generates the *sendC* control signal, as specified in Table 7.14. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 7.14-1: The fairness algorithm and the downstream shaper allow the *sendC* indication to indicate that the entire ringlet is available for fairness eligible transmissions.

Row 7.14-2: The fairness algorithm and the downstream shaper allow the *sendC* indication to indicate that the ringlet up to the congestion point is available for fairness eligible transmissions.

Row 7.14-3: The *sendC* indication indicates that none of the ringlet is available for fairness eligible transmissions.

7.5.8 DownstreamShaper state machine

The following subclauses (7.5.8.1–7.5.8.3) describe parameters used within the context of this state machine.

Table 7.14—FairnessEligibleIndication state table

Current state		Row	Next state	
state	condition		action	state
START	addRateCongestedOK && sendD	1	sendC = MAX_STATIONS;	START
	addRateOK && sendD	2	sendC = hopsToCongestion;	
	—	3	sendC = 0;	

7.5.8.1 DownstreamShaper state machine definitions

TICK

See 7.2.1.

7.5.8.2 DownstreamShaper state machine variables*creditD*

The total current credits accumulated by the downstream shaper, in units of bytes.

currentTime

See 7.2.2.

hiLimitD

The high threshold for the downstream shaper credits.

Default: $loLimitD + mtuSize$.*loLimitD*

The low threshold for the downstream shaper credits.

Value: $mtuSize$.*mtuSize*

See 7.2.4.

rateD

The configured rate for unreserved downstream traffic, in units of bytes per second.

Default: $unreservedRate$, converted to bytes per second.*sendD*

See 7.2.2.

tickTime

A value representing the start time of the current tick processing interval.

7.5.8.3 DownstreamShaper state machine routines*Min(value1, value2)*

See 7.2.3.

7.5.8.4 DownstreamShaper state table

The downstream shaper monitors the add and transit traffic to ensure sufficient levels of sustainable subclassA0 traffic for downstream stations, as specified in Table 7.15. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.15—DownstreamShaper state table

Current state		Row	Next state	
state	condition		action	state
START	(currentTime-tickTime) >= TICK	1	creditD = Min(hiLimitD, creditD+rateD*(currentTime-tickTime)); tickTime = currentTime;	START
	—	2	sendD = (creditD>=loLimitD);	

Row 7.15-1: The *creditD* accumulations occur every TICK time interval. The accumulations increase by the unreserved bandwidth (the unreserved fraction times the link rate) multiplied by the update interval. And the *tickTime* is updated at the start of each update interval. (*creditD* is decremented by non-subclassA0 add and transit traffic in Table 7.31, Rows 6, 8, 10, 11, and 13; Table 7.34, Row 9; and Table 7.35, Rows 9 and 11.)

Row 7.15-2: Set *sendD* whenever sufficient *creditD* credits are available.

7.6 Receive operation

The receive operation is initiated upon receipt of a PHY_DATA.indication (see 8.2.2) or upon receipt of a wrapped frame from the opposite datapath (see 7.7.9).

The receive operation provides the following functionality, which is detailed in 7.6.1–7.6.4:

- a) Receipt of frames, accounting, and direction to appropriate locations (see 7.6.3).
- b) Detection and monitoring of frames arriving on the wrong ringlet (see 7.6.4).

7.6.1 Receive operation for strict data frames

Strict data frames are data frames with the *so* (strict order) field set to 1 (see 6.3 and 9.7.4). These frames are required to be delivered strictly in order (within any {*sa*, *da*, service_class} tuple) and without any duplications, in all circumstances. Strict data frames are therefore subject to additional rules during the receive operation.

7.6.1.1 Context containment for strict data frames

A context is an image of the ring (including the stations, spans, and link statuses), as viewed from the local station, and as stored in the topology and status database (see 11.5). Each frame is transmitted within an implicit context. Frames transmitted prior to a change in ring image can have an outdated context.

In order to avoid any duplication or misordering of frames with no tolerance of such, receiving stations use a context containment mechanism that removes strict data frames from the ring that were transmitted using an outdated context prior to the transmission of strict data frames using an updated context.

The context containment mechanism is triggered by the reception of a protection control frame that results in changes to the local topology and status database. See Clause 11 for a description of when protection control frames are dispatched and their impact on a station's topology and status database. The context containment mechanism involves discarding strict data frames that could be misordered or duplicated received.

Context is contained through the following actions:

- a) At the initial triggering of context containment (when *containmentActive* transitions to TRUE), all strict data frames in a secondary transit queue (STQ) shall be discarded, incrementing *containedFrames* for each discarded frame.
- b) At the initial triggering of context containment, all strict data frames in Q_TX_SS shall be discarded, without incrementing *containedFrames* for these frames (because they have not been transmitted).
- c) As long as context containment remains active, strict data frames are discarded before being allowed to be transited or transmitted (see Table 7.20, Row 34 and Table 7.29, Row 7 and Row 9, respectively).
- d) At any time, strict data frames that have inconsistent hop counts are optionally discarded in transit (see Table 7.20, Row 28) and are always discarded upon being received at their destination (see Table 7.24, Row 9).
- e) At any time, any data frames that are received on the wrong ringlet are discarded in transit by steering stations (see Table 7.20, Row 36) and by wrapping stations that do not expect to receive such frames (see Table 7.20, Row 38).
- f) At any time, any data frames that are received that have been wrapped and rewound are discarded at a following wrapping point (see Table 7.22, Row 4 and Table 7.22, Row 5).

7.6.1.2 *ttl* validation for strict data frames

In addition to the *ttl* scoping rules used for all frames, strict data frames are also discarded before sending to the client if the number of hops they have traveled is not consistent with the receiving station's topology and status database. This extra discard check ensures that no frame duplication occurs due to an intermediate station going into passthrough, or for any hazardous wrap/unwrap sequences. Both wrapping and steering systems perform this check on strict data frames being received for copying to the client. Any station can optionally implement the consistency check as part of the ReceiveCheck state machine (in order to save ring bandwidth by removing the possibly faulty frames as soon as the inconsistency is discovered).

7.6.1.3 Reception in steering systems for strict data frames

Steering systems dealing with strict data frames use the context containment mechanism when a station receives a protection switch event. Upon reception of such a protection control frame, receiving steering stations purge all strict data frames currently within the transit queues of both ringlets and continue to purge all received strict data frames until directed otherwise by the topology and protection entity. After the duration has expired, the station returns to normal operations and transits all frames as requested.

Consider the illustration in Figure 7.6. A link failure occurs between stations S1 and S2. Protection control frames are dispatched by the stations adjacent to the failure (i.e., stations S1 and S2). The context containment mechanism is triggered in stations S1 and S2 when the fault is detected and they have to update their local topology and status database. The context containment mechanism is triggered at the interior stations (i.e., stations S3, S4, S5, S6, and S7), when they receive a protection switch event (i.e., a protection control frame resulting in a change to the local topology and status database).

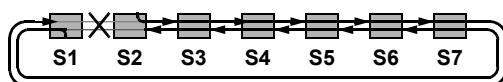


Figure 7.6—Protection event example

The end result of this operation is the removal of strict data frames on the ring that were dispatched using a context that is no longer current.

Reorder can occur during a protection switch, such as when a source station transmits frames using one context followed by a different context. The context containment mechanism ensures that there is no reorder of frames. Context containment removes frames dispatched using an old context before frames are dispatched using a new context to prevent any frame reorder.

7.6.1.4 Reception in wrapping systems for strict data frames

The act of unwrapping the ring (i.e., ring recovery) traps frames on the wrong ringlet. These frames are discarded from the secondary ringlet if no protection event is current. However, in the case of a second fault occurring immediately after the ring recovery, it is likely that not all frames are discarded, and wrapping these frames onto their primary ringlet would (if allowed) cause reordering. Therefore, following the act of unwrapping a ring, all stations delete all strict data frames in flight and all strict data frames in the transit queues of both ringlets whose *ri* does not match the ringlet traveled. This state persists until the topology and protection protocols determine that it is no longer necessary.

NOTE—If a new wrap occurs within this period, some additional frame loss occurs (of the newly wrapped frames) but no reorder occurs.

7.6.2 Reception in wrapping systems

Wrapping systems perform additional checks beyond those checks performed on steering systems. Because it is possible that a series of failures can leave the topology significantly different from when the frame was originally launched, the *ps* (passed source) field in the frame header and some eligibility rules for wrapping frames are used to prevent frame duplication.

The *ps* field is usually set to 0 on transmission of a frame. The exception to this rule is when transmission occurs at an edge station that is center wrapped. These frames bypass the receive state machines on the wrapped ringlet and therefore normally do not have the *ps* field adjusted when wrapping onto the opposing ringlet (see Table 7.23, Row 5). For these frames that require strict ordering, to ensure that context containment functions properly in this special case, these frames are considered to have “passed the source” on transmission, and the *ps* field is therefore set to 1 on transmit (see Table 7.29, Row 13).

A frame is wrapped from its primary ringlet to its opposing ringlet only when the *we* (wrap eligible) field is 1, the *ps* field is 0, and the *ri* (ringlet identifier) of the frame and ringlet traveled are the same. When the frame passes the source station on the secondary ringlet, the *ps* field is set to 1. A frame is wrapped from its opposing ringlet to its primary ringlet only when the *we* field is 1, the *ps* field is 1, and the *ri* of the frame and ringlet traveled are different.

NOTE—A frame could arrive out of order if a station was to change the *ri* associated with transmit frames of a particular conversation. This is because frames transmitted with a new *ri* may traverse a fewer number of stations, arriving at the intended receiver prior to arrival of the earlier frames. As strict order frames cannot be misordered, they must either be flushed or purged from the ring via context containment prior to the changing of selected *ri*. Wrapping stations (both center and edge) maintain ordering of strict frames during a protection switch by not changing the *ri* for strict order traffic following a switch. Strict traffic continues to follow the same path on the ring as the traffic preceding the switch.

The *ttl* is decremented on the original ringlet, not decremented on the opposing ringlet. Because edge stations that are center wrapped behave as though the receive is occurring on the primary ringlet, but actually occurring on the opposing ringlet, the *ttl* is decremented in these edge stations on the opposing ringlet, not on the primary ringlet.

These behaviors of the source station and the wrapped stations in a wrapping ring are illustrated in Figure 7.7, where step a) shows the initial clearing of the *ps* bit on transmission, step b) shows the setting of

the *ps* bit on transiting the source station on the secondary ringlet, and step c) shows the checking of the *ps* bit by the wrapped stations.

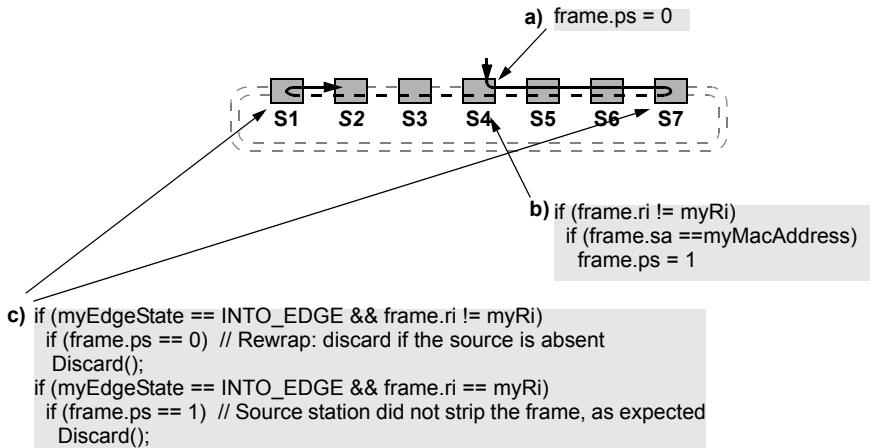


Figure 7.7—Use of *ps* bit in wrapping systems

7.6.3 Receive operation state machines

This subclause specifies the state machines for receiving frames from the physical layer or from a wrap point, and the actions taken after receipt of the frames. The receive operation is logically divided into a receive entity and a filter entity.

7.6.3.1 Receive entity

Each station has transmit and receive entities, as illustrated in Figure 7.8, which use rules designed to support local and bridged RPR traffic.

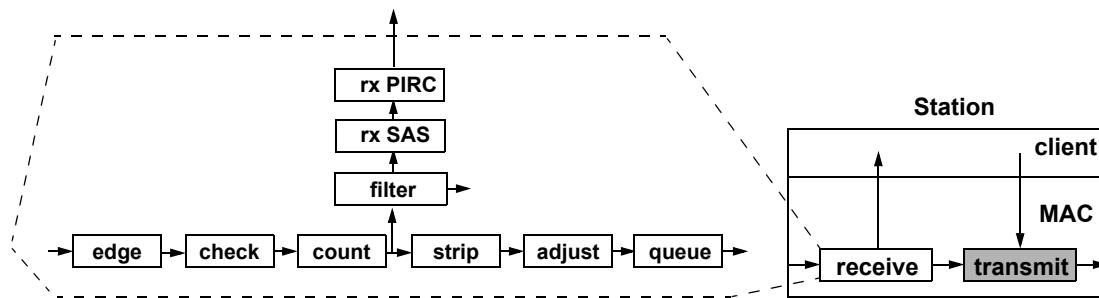


Figure 7.8—Receive subentities

The receive entity includes the following subentities, detailed in 7.6.3.3–7.6.3.10. Each received frame is processed by each state machine (unless discarded) in the order listed below:

- Edge (see 7.6.3.6): Frames received across an edge are stripped.
- Check (see 7.6.3.7): Errorred frames are discarded.
- Count (see 7.6.3.8): Traffic-flow statistics are updated.

- d) Strip (see 7.6.3.9): Selected frames are stripped or transited.
- e) Adjust (see 7.6.3.10): Frames headers are adjusted.
- f) Queue (see 7.6.3.10): Frames are queued into PTQ or STQ.
- g) Filter (see 7.6.3.3): Frames destined to the client are filtered.
- h) Rx SAS (see 7.6.3.5): Frames destined to the client are processed by SAS.
- i) Rx PIRC (see 15.5.1): Frames destined to the client are processed by PIRC.

7.6.3.2 Strip rules overview

In general, strip actions are enabled only when the *ri* (ringlet identifier) of the frame and *myRi* of the datapath are the same. The exception to this is for edge stations that are center wrapped, where some frames will be stripped from the opposing ring. (See Table 7.20, Row 10 and the definition of *SameSide()*.) When thus enabled, the strip rules affect the stripping of frames from the ring (by not transiting them), as summarized in Table 7.16. In the case of any ambiguity between this summary and the receive entity state machines, the state machines shall take precedence. These strip rules apply to data and control frames.

Table 7.16—Strip rules

Frame field conditions			Row	action
fi	da	sa		
—	—	sa == myMacAddress	1	—
fi == FI_NONE	da == myMacAddress	sa != myMacAddress	2	CopyToTransit();
	da != myMacAddress	sa != myMacAddress	3	
fi != FI_NONE	—	sa != myMacAddress	4	

Row 7.16-1: All frames are stripped at their sourced station.

Row 7.16-2: Non-flooded frames are stripped at their destination.

Row 7.16-3: Non-flooded frames that are not destined for or sourced from this station are not stripped.

Row 7.16-4: Flooded frames are not destination stripped.

NOTE—For flexibility and uniformity, the receive rules have minimal checks before passing frames to the receive filter (see 7.6.3.4). Some of the rejected frames (including others' non-flooded unicast frames) are discarded by the filter before being passed to the client. This strategy decouples the reception rules from the selection rules that specify which data frames are passed to the client.

7.6.3.3 Filter entity

Each filter entity is composed of select and count subentities, as illustrated in Figure 7.9, which use rules designed to support local and bridged RPR traffic.

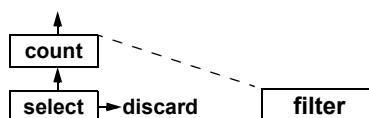


Figure 7.9—Filter subentities

The filter entity includes the following subentities, detailed in 7.6.3.12–7.6.3.13. Each received frame is processed by each state machine (unless discarded) in the order listed below:

- a) Select (see 7.6.3.12): Frames are selected for one of the following actions:
 - 1) Data: data frames are passed to the client.
 - 2) Control: control frames are passed to the MAC control entity.
 - 3) Discard: invalid data and control frames are discarded.
- b) Count (see 7.6.3.12 and 7.6.3.13): Selected frames are counted as follows:
 - 1) Data: data frames and bytes are counted per service class before being passed to the client.
 - 2) Control: control frames are counted per *controlType* before being passed to the client.

7.6.3.4 Filter rules overview

In general, filter actions copy frames to the client or to the MAC control sublayer only when the *ri* (ringlet identifier) of the frame and the *myRi* value of the datapath are the same. The exception to this is for edge stations that are center wrapped, where some frames will be copied from the opposing ring. (See Table 7.24, Row 5 and the definition of *SameSide()*.) The filter rules that determine which of the data frames are copied to the client are summarized in Table 7.17. In the case of any ambiguity between this summary and the filter entity state machines, the state machines shall take precedence.

Table 7.17—Client filter selection summary

Condition			Row	action
Frame parameter conditions		<i>myRxFilter</i>		
<i>da</i>	<i>sa</i>			
da == myMacAddress	sa != myMacAddress	—	1	CopyToClient();
Multicast(da)	sa != myMacAddress	—	2	
da != myMacAddress && !Multicast(da)	sa != myMacAddress	RX_FLOOD	3	
		RX_BASIC	4	
—	sa == myMacAddress	—	5	

Row 7.17-1: Self-destined, non-self-sourced unicast frames are copied to the client.

Row 7.17-2: Broadcast and multicast, non-self-sourced frames are copied to the client.

Row 7.17-3: When *myRxFilter*==RX_FLOOD, others' unicast frames are copied to the client.

Row 7.17-4: When *myRxFilter*==RX_BASIC, others' unicast frames are not copied to the client.

Row 7.17-5: Self-sourced frames are never copied to the client.

The filter rules that determine which of the control frames are copied to the MAC control sublayer are summarized in Table 7.18. In the case of any ambiguity between this summary and the filter entity state machines, the state machines shall take precedence.

Table 7.18—MAC control filter selection summary

da	Row	action
da == myMacAddress	1	CopyToControl();
Multicast(da)	2	
—	3	—

Row 7.18-1: Self-destined unicast control frames are copied to the MAC control sublayer.

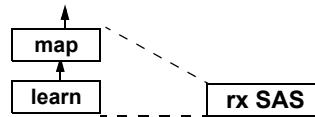
Row 7.18-2: Broadcast and multicast control frames are copied to the MAC control sublayer.

Row 7.18-3: All other control frames are not copied to the MAC control sublayer.

The primary differences between the two set of filter rules are that self-sourced frames are not sent to the client, but they are sent to the MAC control sublayer, and a client can elect to receive frames not addressed to it, but control frames are not sent to the MAC control sublayer unless addressed to the station.

7.6.3.5 Receive SAS entity

Each receive SAS entity is composed of learn and map subentities, as illustrated in Figure 7.10, which use rules designed to support local and bridged RPR traffic.

**Figure 7.10—Rx SAS subentities**

The receive SAS entity includes the following subentities, detailed in 14.5.4. Each received frame is processed by each state machine in the order listed below:

- a) Learn (see 14.5.4.1): Learns associations between station address and remote addresses, for SAS frames.
- b) Map (see 14.5.4.1): Optionally remaps extended addresses to base addresses.

7.6.3.6 ReceiveFromEdge state machine

The ReceiveFromEdge state machine implements the functions necessary for copying certain frames from a neighbor across an edge to the MAC control sublayer, and discarding all other frames received across an edge.

The following subclauses describe parameters used within the context of this state machine.

7.6.3.6.1 ReceiveFromEdge state machine definitions

CT_TOPO_PROT

See 7.2.4.

FT_FAIRNESS

See 7.2.4.

FROM_EDGE

See 7.2.2.

FT_CONTROL

See 7.2.4.

MAX_STATIONS

See 7.2.4.

NULL

See 7.2.1.

Q_RX_CHECK

Q_RX_PHY

Q_RX_TP

See 7.2.1.

SINGLE_CHOKE

See 7.2.4.

7.6.3.6.2 ReceiveFromEdge state machine variables*currentTime*

See 7.2.2.

frame

The contents of an RPR frame.

keepaliveTime

See 7.2.4.

myEdgeState

See 7.2.2.

size

The size, in bytes, of an RPR frame.

*toCtrlFrames**toCtrlTopoTPFrames*

See 7.2.5.

7.6.3.6.3 ReceiveFromEdge state machine routines*Crc32(frame)*

See 7.2.3.

Crc32Compact(frame)

See 7.2.3.

*Dequeue(queue)**Enqueue(queue,frame)*

See 7.2.3.

Parity(frame)

See 7.2.3.

SizeOf(frame)

See 7.2.3.

7.6.3.6.4 ReceiveFromEdge state table

The ReceiveFromEdge state machine specified in Table 7.19 implements the functions necessary for filtering frames received across an edge. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.19—ReceiveFromEdge state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_RX_PHY)) != NULL	1	size = SizeOf(frame);	CHECK
	—	2	—	START
CHECK	frame.ft == FT_CONTROL && frame.controlType == CT_TOPO_PROT && frame.ttl == MAX_STATIONS && frame.hec == Crc16(frame) && frame.fcs == Crc32(frame) && !SameSide(frame)	3	toCtrlFrames += 1; toCtrlTopoTPFrames += 1; Enqueue(Q_RX_TP, frame);	START
	myEdgeState != FROM_EDGE	4	Enqueue(Q_RX_CHECK, frame);	
	frame.ft == FT_CONTROL && frame.controlType == CT_TOPO_PROT && frame.ttl == MAX_STATIONS && frame.hec == Crc16(frame) && frame.fcs == Crc32(frame) && (!optionTransitLength (size >= CONTROL_MIN && size <= CONTROL_MAX))	5	toCtrlFrames += 1; toCtrlTopoTPFrames += 1; Enqueue(Q_RX_TP, frame);	
	frame.ft == FT_FAIRNESS && frame.ftType == SINGLE_CHOKE && frame.parity == Parity(frame) && frame.fcs == Crc32Compact(frame) && size == FAIRNESS_SIZE	6	keepaliveTime = currentTime;	
	—	7	—	

Row 7.19-1: Fetch a new input frame from the PHY.

Row 7.19-2: Refetch input frames while NULL frames are received.

Row 7.19-3: Non-errored topology and protection frames received from the neighbor on the wrong ringlet are passed to the MAC control sublayer for miscabling detection (see 11.6.8). (There is no need to check for some errors that do not directly effect the validity of the frames because the frames are being used for a limited purpose and are not being transited.) They are counted as received and are copied to the topology and protection entity of the MAC control sublayer.

Row 7.19-4: Frames are accepted for further processing when received across a functional span.

Row 7.19-5: Non-errored topology and protection frames received across an edge from the neighbor on the other side of the edge are passed to the MAC control sublayer for topology and protection processing. (There is no need to check for some errors that do not directly affect the validity of the frames because the frames are being used for a limited purpose and are not being transited.) They are counted as received and are copied to the topology and protection entity of the MAC control sublayer.

Row 7.19-6: Non-errored fairness frames received across an edge from the neighbor on the other side of the edge reset the protection keepalive timer (see 11.6.4). (There is no need to check for some errors that do not

directly affect the validity of the frames because the frames are being used for a limited purpose and are not being transited.) They are not counted as received and are not copied to the MAC control sublayer.

Row 7.19-7: Other beyond-the-edge frames are silently discarded.

7.6.3.7 ReceiveCheck state machine

The ReceiveCheck state machine implements the functions necessary for discarding invalid frames.

The following subclauses (7.6.3.7.1–7.6.3.7.3) describe parameters used within the context of this state machine.

7.6.3.7.1 ReceiveCheck state machine definitions

CONTROL_MIN

CONTROL_MAX

See 7.2.4.

DATA_MIN

See 7.2.4.

EXT_HDR_SIZE

See 7.2.4.

FAIRNESS_SIZE

See 7.2.4.

FT_CONTROL

FT_FAIRNESS

FT_IDLE

See 7.2.4.

IDLE_SIZE

See 7.2.4.

INTO_EDGE

See 7.2.2.

NULL

See 7.2.1.

PASSTHROUGH

See 7.2.2.

Q_RX_CHECK

Q_RX_COUNT

See 7.2.1.

STEERING

See 7.2.4.

STOMP

See 7.2.1.

7.6.3.7.2 ReceiveCheck state machine variables

badAddressFrames

Counter for frames received with an invalid *sa* (or *saCompact*) value.

badFcsFrames

See 7.2.2.

badFcsUser

See 7.2.4.

badHecFrames

Counter for frames received with *hec* value not matching the expected *hec* value.

badParityFrames

Counter for frames received with *parity* value not matching the expected *parity* value.

containedFrames

See 7.2.2.

containmentActive

See 7.2.4.

frame

The contents of an RPR frame.

mtuSize

See 7.2.4.

myEdgeState

See 7.2.2.

myRi

See 7.2.2.

optionControlFcsDeletion

Available for store-and-forward stations wishing to not propagate errored frames. Not applicable to cut-through stations.

TRUE—Indicates that control frames with invalid *fcs* values are discarded during transit.

FALSE—(Otherwise.)

optionDataFcsDeletion

Available for store-and-forward stations wishing to not propagate errored frames, when there are no stations on the ring with a desire to receive *fcs* errored data frame (indicated by *badFcsUser*). Not applicable to cut-through stations.

TRUE—Indicates that data frames with invalid *fcs* values are discarded during transit, if no other station wants to receive them.

FALSE—(Otherwise.)

optionTransitConsistency

Available for any stations wishing to not propagate contained frames.

TRUE—Indicates that consistency is checked during transit.

FALSE—(Otherwise.)

optionTransitLength

Available for store-and-forward stations wishing to not propagate errored frames. Not applicable to cut-through stations.

TRUE—Indicates that data and control frame lengths are checked during transit.

FALSE—(Otherwise.)

protConfig

See 7.2.4.

scffErrors

See 7.2.2.

size

The size, in bytes, of an RPR frame.

tooLongFrames

Counter for frames received with length exceeding the maximum allowed length.

tooShortFrames

Counter for frames received with length shorter than the minimum allowed length.

tossWrongRingletIDs

See 7.2.2.

ttlExpiredFrames

See 7.2.2.

wrongRingletSensed

See 7.2.2.

7.6.3.7.3 ReceiveCheck state machine routines

ConsistentHopCount(frame)

See 7.2.3.

*Compact(frameType)*True if *frameType* is one of the compact formats (i.e., FT_FAIRNESS or FT_IDLE).*Crc16(frame)**Crc32Compact(frame)**Crc32(frame)*

See 7.2.3.

*Dequeue(queue)**Enqueue(queue,frame)*

See 7.2.3.

Multicast(address)

See 7.2.3.

Parity(frame)

See 7.2.3.

SizeOf(frame)

See 7.2.3.

Stomp(frame)

See 7.2.3.

7.6.3.7.4 ReceiveCheck state table

The ReceiveCheck state machine specified in Table 7.20 implements the functions necessary for validating received frames.

Unlike other state tables in this standard, the rows are evaluated in top-to-bottom order only for rows other than within the following groups:

- a) Row 7.20-4 through Row 7.20-9
- b) Row 7.20-12 through Row 7.20-18
- c) Row 7.20-20 through Row 7.20-24

Within the aforementioned groups, rows are evaluated in whatever order within their respective group that is chosen by the implementation.

Additionally, the ordering of states LENGTH, CHECK, and TTL_FAIR, relative to each other, is an implementation decision.

Because of the above evaluation flexibility, different implementations can report different counts for frames with multiple errors, each reporting only one error for each errored frame. Furthermore, because not all implementations can check frame length in transit (e.g., cut-through implementations), frame length error counts can also be different between different implementations.

Optional rows are shaded in gray. The optional rows allow for discarding some additional invalid frames while in transit, instead of leaving them to be discarded on receipt at the destination station. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.20—ReceiveCheck state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_RX_CHECK)) != NULL	1	size = SizeOf(frame);	BYPASS
	—	2	—	START
BYPASS	frame.ft == FT_IDLE && myEdgeState == PASSTHROUGH	3	—	START
	frame.ft == FT_IDLE && size < IDLE_SIZE	4	tooShortFrames += 1;	
	frame.ft == FT_IDLE && size > IDLE_SIZE	5	tooLongFrames += 1;	
	frame.ft == FT_IDLE && frame.parity != Parity(frame)	6	badParityFrames += 1;	
	frame.ft == FT_IDLE && frame.fcs != Crc32Compact(frame)	7	badFcsFrames += 1;	
	frame.ft == FT_IDLE && Multicast(frame.saCompact)	8	badAddressFrames += 1;	
	frame.ft == FT_IDLE	9	—	
	myEdgeState == PASSTHROUGH	10	—	FINAL
	—	11	—	LENGTH
LENGTH	frame.ft == FT_FAIRNESS && size < FAIRNESS_SIZE	12	tooShortFrames += 1; scffErrors += 1;	START
	frame.ft == FT_FAIRNESS && size > FAIRNESS_SIZE	13	tooLongFrames += 1; scffErrors += 1;	
	optionTransitLength && frame.ft == FT_CONTROL && size < CONTROL_MIN	14	tooShortFrames += 1;	
	optionTransitLength && frame.ft == FT_CONTROL && size > CONTROL_MAX	15	tooLongFrames += 1;	
	optionTransitLength && frame.ft == FT_DATA && !frame.ef && size < DATA_MIN	16	tooShortFrames += 1;	
	optionTransitLength && frame.ft == FT_DATA && frame.ef && size < DATA_MIN + EXT_HDR_SIZE;	17	tooShortFrames += 1;	
	optionTransitLength && frame.ft == FT_DATA && size > mtuSize	18	tooLongFrames += 1;	
	—	19	—	CHECK

Table 7.20—ReceiveCheck state table (continued)

Current state		Row	Next state	
state	condition		action	state
CHECK	frame.ft == FT_FAIRNESS && frame.parity != Parity(frame)	20	badParityFrames += 1; scffErrors += 1;	START
	frame.ft == FT_FAIRNESS && frame.fcs != Crc32Compact(frame)	21	badFcsFrames += 1; scffErrors += 1;	
	frame.ft != FT_FAIRNESS && frame.hec != Crc16(frame)	22	badHecFrames += 1;	
	frame.ft == FT_FAIRNESS && Multicast(frame.saCompact)	23	badAddressFrames += 1; scffErrors += 1;	
	frame.ft != FT_FAIRNESS && Multicast(frame.sa)	24	badAddressFrames += 1;	
	—	25	—	TTL_FAIR
TTL_FAIR	frame.ttl == 0	26	ttlExpiredFrames += 1;	START
	—	27	—	OPTION
OPTION	optionTransitConsistency && frame.ft == FT_DATA && SameSide(frame) && frame.so && !ConsistentHopCount(frame)	28	containedFrames += 1;	START
	optionDataFcsDeletion && !badFcsUser && frame.ft == FT_DATA && frame.fcs == Stomp(frame)	29	—	
	optionDataFcsDeletion && !badFcsUser && frame.ft == FT_DATA && frame.fcs != Crc32(frame)	30	badFcsFrames += 1;	
	optionControlFcsDeletion && frame.ft == FT_CONTROL && frame.fcs == Stomp(frame)	31	—	
	optionControlFcsDeletion && frame.ft == FT_CONTROL && frame.fcs != Crc32(frame)	32	badFcsFrames += 1;	
	—	33	—	
	—	33	—	RI

Table 7.20—ReceiveCheck state table (continued)

Current state		Row	Next state	
state	condition		action	state
RI	containmentActive && ((protConfig == STEERING && frame.so == 1) (protConfig == WRAPPING && frame.ri != myRi))	34	wrongRingletSensed = 0; containedFrames += 1;	START
	protConfig == STEERING && frame.ri != myRi && frame.ft == FT_FAIRNESS	35	—	FINAL
	protConfig == STEERING && frame.ri != myRi	36	containedFrames += 1;	START
	protConfig == WRAPPING && frame.ri != myRi && frame.ft == FT_FAIRNESS	37	—	FINAL
	protConfig == WRAPPING && tossWrongRingletIDs && frame.ri != myRi	38	wrongRingletSensed = 1; containedFrames += 1;	START
	protConfig == WRAPPING && frame.ri != myRi	39	wrongRingletSensed = 1;	FCS
	frame.ft == FT_FAIRNESS	40	—	FINAL
	—	41	—	FCS
FCS	frame.fcs == Crc32(frame)	42	—	FINAL
	frame.fcs == Stomp(frame)	43	—	
	—	44	frame.fcs = Stomp(frame); badFcsFrames += 1;	
FINAL	—	45	Enqueue(Q_RX_COUNT, frame);	START

Row 7.20-1: Fetch an input frame and begin checking when a non-NUL frame is provided.**Row 7.20-2:** Refetch an input frame when only a NUL frame is available.**Row 7.20-3:** When in passthrough mode, skip checking of idle frames.**Row 7.20-4:** Idle frames that are too short are discarded.**Row 7.20-5:** Idle frames that are too long are discarded.**Row 7.20-6:** Idle frames with bad parity are discarded.**Row 7.20-7:** If an idle frame *fcs* does not match its expected value, discard the frame.**Row 7.20-8:** If an idle frame *sa* corresponds to a group address, discard the frame.**Row 7.20-9:** Idle frames are discarded (without error) as soon as being received.**Row 7.20-10:** When in passthrough mode, no transit frame checking is done.**Row 7.20-11:** Check next for frame length errors.**Row 7.20-12:** Fairness frames that are too short are discarded.**Row 7.20-13:** Fairness frames that are too long are discarded.**Row 7.20-14:** (optional) Control frames that are too short are discarded.**Row 7.20-15:** (optional) Control frames that are too long are discarded.

Row 7.20-16: (optional) Basic data frames that are too short are discarded.
Row 7.20-17: (optional) Extended data frames that are too short are discarded.
Row 7.20-18: (optional) Data frames that are too long are discarded.
Row 7.20-19: Check next for header check and frame check errors.

Row 7.20-20: Fairness frames with bad parity are discarded.
Row 7.20-21: If a fairness frame *fcs* does not match its expected value, discard the frame.
Row 7.20-22: If a data/control-frame *hec* does not match its expected value, discard the frame.
Row 7.20-23: If a fairness frame *sa* corresponds to a group address, discard the frame.
Row 7.20-24: If a data/control frame *sa* corresponds to a group address, discard the frame.
Row 7.20-25: Check next for *ttl* expiration and fairness frames.

Row 7.20-26: Expired frames are discarded.
Row 7.20-27: Check next for optional checks.

Row 7.20-28: (optional) Frames that are strict and inconsistent are discarded by a transit station.
Row 7.20-29: (optional) Data frames with a stomped *fcs* value, on a ring with no one interested in receiving frames with *fcs* errors, are silently discarded by a transit station.
Row 7.20-30: (optional) Data frames with a bad *fcs* value, on a ring with no one interested in receiving frames with *fcs* errors, are counted and discarded by a transit station.
Row 7.20-31: (optional) Control frames with a stomped *fcs* value are silently discarded by a transit station.
Row 7.20-32: (optional) Control frames with a bad *fcs* value are counted and discarded by a transit station.
Row 7.20-33: Check next for frames on the wrong ringlet.

Row 7.20-34: Possibly misordered or duplicated frames are discarded when *containmentActive* is set.
Row 7.20-35: Fairness frames received on the wrong ringlet are processed further.
Row 7.20-36: Frames received on the wrong ringlet are discarded by steering stations.
Row 7.20-37: Fairness frames received on the wrong ringlet are processed further.
Row 7.20-38: Wrong-ringlet frames are flagged and discarded, when *tossWrongRingletIDs* is set (see 7.6.4).
Row 7.20-39: Wrong-ringlet frames are flagged (see 7.6.4), but not discarded.
Row 7.20-40: Fairness frames are processed further.
Row 7.20-41: Otherwise, wrong-ringlet frames are processed normally. Check next for CRC errors.

Row 7.20-42: A frame with a good payload CRC is processed normally.
Row 7.20-43: A frame with a stomped payload CRC is processed normally.
Row 7.20-44: A frame with a bad payload CRC is stomped (see E.2.3), an error is logged, and the revised frame is then processed normally.

Row 7.20-45: Validated frames are next analyzed by performance monitor counters.

7.6.3.8 ReceiveCount state machine

The ReceiveCount state machine implements the functions necessary for updating flow-rate statistics.

The following subclauses (7.6.3.8.1–7.6.3.8.3) describe parameters used within the context of this state machine.

7.6.3.8.1 ReceiveCount state machine definitions

CLASS_A0
 CLASS_A1
 CLASS_B
 CLASS_C
 See 7.2.4.

NULL

See 7.2.1.

PASSTHROUGH

See 7.2.2.

Q_RX_COUNT

Q_RX_FILTER

Q_RX_STRIP

See 7.2.1.

7.6.3.8.2 ReceiveCount state machine variables

copyFrame

The contents of an RPR frame being copied from the transit path to the client or to the MAC control sublayer.

frame

The contents of an RPR frame.

myEdgeState

See 7.2.2.

rxMcastClassABytes

rxMcastClassAFrames

rxMcastClassBCirBytes

rxMcastClassBCirFrames

rxMcastClassBEirBytes

rxMcastClassBEirFrames

rxMcastClassCBytes

rxMcastClassCFrames

See 7.2.5.

rxUcastClassABytes

rxUcastClassAFrames

rxUcastClassBCirBytes

rxUcastClassBCirFrames

rxUcastClassBEirBytes

rxUcastClassBEirFrames

rxUcastClassCBytes

rxUcastClassCFrames

See 7.2.5.

size

The size, in bytes, of an RPR frame.

7.6.3.8.3 ReceiveCount state machine routines

Dequeue(queue)

See 7.2.3.

Duplicate(frame)

Returns a copy of *frame*.

Enqueue(queue,frame)

See 7.2.3.

Multicast(address)

See 7.2.3.

SizeOf(frame)

See 7.2.3.

7.6.3.8.4 ReceiveCount state table

The ReceiveCount state machine specified in Table 7.21 implements the functions necessary for counting received bytes and frames. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.21—ReceiveCount state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_RX_COUNT)) != NULL	1	size = SizeOf(frame);	UCAST
	—	2	—	START
UCAST	myEdgeState == PASSTHROUGH	3	—	FINAL
	frame.ft != FT_DATA	4	—	
	Multicast(frame.da)	5	—	
	frame.sc == CLASS_A0	6	rxUcastClassAFrames += 1; rxUcastClassABytes += size;	FINAL
	frame.sc == CLASS_A1	7	—	
	frame.sc == CLASS_B && frame.fe == 0	8	rxUcastClassBCirFrames += 1; rxUcastClassBCirBytes += size;	
	frame.sc == CLASS_B && frame.fe == 1	9	rxUcastClassBEirFrames += 1; rxUcastClassBEirBytes += size;	
	frame.sc == CLASS_C	10	rxUcastClassCFrames += 1; rxUcastClassCBytes += size;	
MCAST	frame.sc == CLASS_A0	11	rxMcastClassAFrames += 1; rxMcastClassABytes += size;	FINAL
	frame.sc == CLASS_A1	12	—	
	frame.sc == CLASS_B && frame.fe == 0	13	rxMcastClassBCirFrames += 1; rxMcastClassBCirBytes += size;	
	frame.sc == CLASS_B && frame.fe == 1	14	rxMcastClassBEirFrames += 1; rxMcastClassBEirBytes += size;	
	frame.sc == CLASS_C	15	rxMcastClassCFrames += 1; rxMcastClassCBytes += size;	
FINAL	—	16	copyFrame = Duplicate(frame); Enqueue(Q_RX_FILTER, copyFrame); Enqueue(Q_RX_STRIP, frame);	START

Row 7.21-1: Fetch the next input frame.

Row 7.21-2: Refetching an input frame whenever a NULL frame is received.

Row 7.21-3: Counters are not updated when operating in passthrough mode.

Row 7.21-4: Only data frames are counted.

Row 7.21-5: Multicast frames have distinct counters.

Row 7.21-6: Count the number of unicast classA frames and bytes, subclassA0 contribution.

Row 7.21-7: Count the number of unicast classA frames and bytes, subclassA1 contribution.

Row 7.21-8: Count the number of unicast classB-CIR frames and bytes.

Row 7.21-9: Count the number of unicast classB-EIR frames and bytes.

Row 7.21-10: Count the number of unicast classC frames and bytes received.

Row 7.21-11: Count the number of multicast classA frames and bytes, subclassA0 contribution.

Row 7.21-12: Count the number of multicast classA frames and bytes, subclassA1 contribution.

Row 7.21-13: Count the number of multicast classB-CIR frames and bytes received.

Row 7.21-14: Count the number of multicast classB-EIR frames and bytes received.

Row 7.21-15: Count the number of multicast classC frames and bytes received.

Row 7.21-16: Continue processing with the receive entity and filter entity state machines.

7.6.3.9 ReceiveStrip state machine

The ReceiveStrip state machine implements the functions necessary for stripping stale frames from the ringlet.

The following subclauses (7.6.3.9.1–7.6.3.9.3) describe parameters used within the context of this state machine.

7.6.3.9.1 ReceiveStrip state machine definitions

FI_NONE

FT_FAIRNESS

See 7.2.4.

INTO_EDGE

See 7.2.2.

MULTI_CHOKE

See 7.2.4.

NULL

See 7.2.1.

PASSTHROUGH

See 7.2.2.

Q_RX_ADJUST

Q_RX_STRIP

See 7.2.1.

SINGLE_CHOKE

See 7.2.4.

STEERING

See 7.2.4.

WRAPPING

See 7.2.4.

7.6.3.9.2 ReceiveStrip state machine variables

containedFrames

See 7.2.2.

currentTime

See 7.2.2.

frame

The contents of an RPR frame.

keepaliveTime

See 7.2.4.

myEdgeState

See 7.2.2.

myMacAddress

See 7.2.4.

protConfig

See 7.2.4.

selfSourcedFrames

A counter for the number of frames sourced by this station and then received by this station that were not expected to have been received by this station.

ttlExpiredFrames

See 7.2.2.

7.6.3.9.3 ReceiveStrip state machine routines

*Dequeue(queue)**Enqueue(queue,frame)*

See 7.2.3.

SameSide(frame)

See 7.2.3.

7.6.3.9.4 ReceiveStrip state table

The ReceiveStrip state machine specified in Table 7.22 implements the functions necessary for stripping received frames from the ring when they have transited as far as they should. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.22—ReceiveStrip state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_RX_STRIP)) != NULL	1	—	STRIP
	—	2	—	START

Table 7.22—ReceiveStrip state table (continued)

Current state		Row	Next state	
state	condition		action	state
STRIP	myEdgeState == PASSTHROUGH	3	—	THRU
	protConfig == WRAPPING && myEdgeState == INTO_EDGE && frame.ri != myRi && frame.sa != myMacAddress && frame.ft != FT_FAIRNESS && frame.ps == 0	4	containedFrames += 1;	START
	protConfig == WRAPPING && myEdgeState == INTO_EDGE && frame.ri == myRi && frame.sa != myMacAddress && frame.ft != FT_FAIRNESS && frame.ps == 1	5	—	START
	frame.ft == FT_FAIRNESS && frame.ffType == MULTI_CHOKE && frame.saCompact != myMacAddress	6	—	THRU
	SameSide(frame) && frame.ft == FT_FAIRNESS && frame.ffType == MULTI_CHOKE && frame.saCompact == myMacAddress	7	—	START
	frame.ft == RPR_FT_FAIRNESS && frame.ffType == RPR_SINGLE_CHOKE && frame.saCompact == myMacAddress && frame.ri == myRi	8	—	
	frame.ft == FT_FAIRNESS && frame.ffType == SINGLE_CHOKE	9	keepaliveTime = currentTime;	
	!SameSide(frame)	10	—	THRU
	frame.sa == myMacAddress && frame.ft != FT_DATA	11	—	START
	frame.sa == myMacAddress && frame.fi != FI_NONE	12	—	
	frame.sa == myMacAddress	13	selfSourcedFrames += 1;	
	frame.da == myMacAddress && frame.fi == FI_NONE	14	—	
	frame.ttl == 1 && frame.fi != FI_NONE	15	—	
	frame.ttl == 1 && frame.fi == FI_NONE	16	ttlExpiredFrames += 1;	
	—	17	—	THRU

Table 7.22—ReceiveStrip state table (continued)

Current state		Row	Next state	
state	condition		action	state
THRU	myEdgeState == INTO_EDGE && protConfig == STEERING	18	—	START
	myEdgeState == INTO_EDGE && frame.we == 0	19	—	
	—	20	Enqueue(Q_RX_ADJUST, frame);	

Row 7.22-1: Fetch a new input frame.

Row 7.22-2: Refetch frames while NULL frames are returned.

Row 7.22-3: All frames are passed while in passthrough mode.

Row 7.22-4: On wrap-capable rings, twice wrapped frames are stripped at the wrap point.

Row 7.22-5: On wrap-capable rings, rewrapped frames are stripped if the source is unconfirmed.

Row 7.22-6: Multi-choke fairness frames are not stripped before their sourcing station.

Row 7.22-7: Multi-choke fairness frames are stripped at their sourcing station.

Row 7.22-8: With edge wrapping, it can appear that a station receives a copy of its own fairness frame, which should not be counted as a received SCFF.

Row 7.22-9: Single-choke fairness frames are stripped by the source's downstream neighbor. Receiving a single-choke fairness frame resets the protection keepalive timer (see 11.6.4).

Row 7.22-10: Stripping is inhibited when returning on the opposing ringlet.

Row 7.22-11: Self-sourced non-data frames are stripped on their sending ringlet, without error.

Row 7.22-12: Self-sourced flooded frames are stripped on their sending ringlet, without error.

Row 7.22-13: All other self-sourced frames are stripped on their sending ringlet, as errors.

Row 7.22-14: Non-flooded frames addressed to this station are destination stripped from the ring.

Row 7.22-15: Expired flooded frames are quietly discarded, as through normal.

Row 7.22-16: Expired unicast frames are discarded, as through errors.

Row 7.22-17: Processing continues with the not-stripped frames.

Row 7.22-18: Steered frames heading into an edge are stripped.

Row 7.22-19: Non-wrappable frames heading into an edge (or wrap point) are stripped.

Row 7.22-20: Not-stripped frames are moved to the ReceiveAdjust state machine.

7.6.3.10 ReceiveAdjust state machine

The ReceiveAdjust state machine implements the functions necessary for updating control fields in frames that transit the station.

The following subclauses (7.6.3.10.1–7.6.3.10.3) describe parameters used within the context of this state machine.

7.6.3.10.1 ReceiveAdjust state machine definitions

NULL

See 7.2.1.

PASSTHROUGH

See 7.2.2.

Q_RX_ADJUST

Q_TX_PTQ

Q_TX_STQ

See 7.2.1.

WRAPPING

See 7.2.4.

7.6.3.10.2 ReceiveAdjust state machine variables

frame

The contents of an RPR frame.

myDualQueueStation

See 7.2.2.

myEdgeState

See 7.2.2.

myMacAddress

See 7.2.4.

myRi

See 7.2.2.

protConfig

See 7.2.4.

7.6.3.10.3 ReceiveAdjust state machine routines

Crc16(frame)

See 7.2.3.

Dequeue(queue)

Enqueue(queue,frame)

See 7.2.3.

SameSide(frame)

See 7.2.3.

7.6.3.10.4 ReceiveAdjust state table

The ReceiveAdjust state machine specified in Table 7.23 implements the functions necessary for adjusting header values in transiting frames. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.23—ReceiveAdjust state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_RX_ADJUST)) != NULL	1	—	ADJUST
	—	2	—	START
ADJUST	myEdgeState == PASSTHROUGH	3	—	PLACE
	frame.ft == FT_FAIRNESS && protConfig == WRAPPING && frame.ri != myRi && frame.sa == myMacAddress	4	—	
	protConfig == WRAPPING && frame.ri != myRi && frame.sa == myMacAddress	5	frame.ps = 1; frame.hec = Crc16(frame);	
	!SameSide(frame)	6	—	
	frame.ft == FT_FAIRNESS	7	frame.ttl -= 1; frame.parity = Parity(frame);	
	—	8	frame.ttl -= 1; frame.hec = Crc16(frame);	
PLACE	!myDualQueueStation	9	Enqueue(Q_TX_PTQ, frame);	START
	frame.sc == CLASS_A0	10		
	frame.sc == CLASS_A1	11		
	—	12	Enqueue(Q_TX_STQ, frame);	

Row 7.23-1: Fetch a new input frame.**Row 7.23-2:** Refetch input frames whenever a NULL frame is returned.**Row 7.23-3:** All frames are passed while in passthrough mode.**Row 7.23-4:** Wrapped fairness frames are not adjusted.**Row 7.23-5:** Wrapped frames have their *ps* bit set when passing their source station on the opposing ringlet and therefore have their *hec* value adjusted.**Row 7.23-6:** The *ttl* field is not updated for frames transiting on the opposing ringlet. (Table 7.20, Row 36 discards these frames for steering, so this is effectively checking only for wrapping stations transiting the frame on the opposing ringlet or center-wrapped stations transiting the frame into an edge. However, the check is done for all stations just for thoroughness.)**Row 7.23-7:** The *ttl* field in fairness frames transiting on the primary ringlet (or for frames being wrapped by a center wrap station) is updated, and the *parity* is then adjusted.**Row 7.23-8:** The *ttl* field is always updated for frames transiting on the primary ringlet (or for frames being wrapped by a center wrap station), and the *hec* is then adjusted.**Row 7.23-9:** In single queue designs, all transiting frames are placed into the primary transit queue.**Row 7.23-10:** Transiting subclassA0 frames are placed into the primary transit queue.**Row 7.23-11:** Transiting subclassA1 frames are placed into the primary transit queue.**Row 7.23-12:** Transiting classB and classC frames are placed into the secondary transit queue.

7.6.3.11 ReceiveFilter state machine

The ReceiveFilter state machine implements the functions necessary for copying frames from the ringlet to the client or to the MAC control sublayer.

The following subclauses (7.6.3.11.1–7.6.3.11.3) describe parameters used within the context of this state machine.

7.6.3.11.1 ReceiveFilter state machine definitions

CT_TOPO_PROT

See 7.2.4.

FI_NONE

See 7.2.4.

FROM_EDGE

See 7.2.2.

FT_CONTROL

FT_DATA

See 7.2.4.

MAX_STATIONS

See 7.2.4.

NULL

See 7.2.1.

OK_IN_USE

See 7.2.4.

Q_RX_F_C_CNT

Q_RX_F_D_CNT

Q_RX_FILTER

See 7.2.1.

RX_FLOOD

See 7.2.2.

7.6.3.11.2 ReceiveFilter state machine variables

badFcsFrames

See 7.2.2.

containedFrames

See 7.2.2.

copyBadFcs

A boolean variable determining if frames with bad *fcs* values are copied to the client.

TRUE—Causes all data frames destined to the client to be copied to the client, regardless of the validity of the *fcs* field.

FALSE—Causes the MAC to copy to the client only those data frames with valid *fcs* values.

frame

The contents of an RPR frame.

myEdgeState

See 7.2.2.

myMacAddress

See 7.2.4.

myRxFilter

See 7.2.4.

optionSecMacReceive

Available for stations wishing to receive frames sent to their secondary MAC addresses.

TRUE—Indicates that frames sent to a local secondary MAC address will be filtered from the receive path.

FALSE—(Otherwise.)

secMac

See 7.2.4.

7.6.3.11.3 ReceiveFilter state machine routines

ConsistentHopCount(frame)

See 7.2.3.

Crc32(frame)

See 7.2.3.

Dequeue(queue)

See 7.2.3.

Enqueue(queue,frame)

See 7.2.3.

Multicast(address)

See 7.2.3.

SameSide(frame)

See 7.2.3.

7.6.3.11.4 ReceiveFilter state table

The ReceiveFilter state machine specified in Table 7.24 implements the functions necessary for filtering received frames to be copied to the client or the MAC control sublayer. Optional rows are shaded in gray. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.24—ReceiveFilter state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_RX_FILTER)) != NULL	1	—	FILTER
	—	2	—	START

Table 7.24—ReceiveFilter state table (continued)

Current state		Row	Next state	
state	condition		action	state
FILTER	myEdgeState == PASSTHROUGH	3	—	START
	frame.ft == FT_FAIRNESS	4	Enqueue(Q_RX_F_C_CNT, frame);	
	!SameSide(frame)	5	—	
	frame.ft == FT_DATA && !copyBadFcs && frame.fcs != Crc32(frame)	6	badFcsFrames +=1;	
	frame.ft == FT_DATA && frame.sa == myMacAddress	7	—	
	frame.ft == FT_CONTROL && frame.fcs != Crc32(frame)	8	badFcsFrames +=1;	
	frame.ft == FT_DATA && frame.so && !ConsistentHopCount(frame)	9	containedFrames +=1;	
	frame.da == myMacAddress	10	—	
	optionSecMacReceive && (secMac[0].state == OK_IN_USE && frame.da == secMac[0].address)	11	—	
	optionSecMacReceive && (secMac[1].state == OK_IN_USE && frame.da == secMac[1].address)	12	—	
COPY	Multicast(frame.da)	13	—	COPY
	frame.ft == FT_DATA && frame.fi != FI_NONE && myRxFilter == RX_FLOOD	14	—	
	—	15	—	
	frame.fcs != Crc32(frame)	16	badFcsFrames +=1; Enqueue(Q_RX_F_D_CNT, frame);	
	frame.ft == FT_DATA	17	Enqueue(Q_RX_F_D_CNT, frame);	
COPY	frame.ft == FT_CONTROL	18	Enqueue(Q_RX_F_C_CNT, frame);	START

Row 7.24-1: Fetch a new input frame.**Row 7.24-2:** Refetch input frames while NULL frames are received.**Row 7.24-3:** When in passthrough, no frames are copied to the MAC control sublayer or to the client.**Row 7.24-4:** All fairness frames are passed to the MAC control sublayer.**Row 7.24-5:** Wrapped frames are ignored until they are rewrapped.**Row 7.24-6:** Discard bad-payload data frames if the client does not want them.**Row 7.24-7:** Self-sourced data frames are discarded without error counting because they were counted in Table 7.22, Row 13.**Row 7.24-8:** Discard all bad-payload control frames.**Row 7.24-9:** Frames that are strict and inconsistent are discarded.

Row 7.24-10: Frames addressed to the local MAC address are passed to the MAC control sublayer or to the client.

Row 7.24-11: (optional) Frames addressed to secondary MAC address 0 are passed to the MAC control sublayer or to the client.

Row 7.24-12: (optional) Frames addressed to secondary MAC address 1 are passed to the MAC control sublayer or to the client.

Row 7.24-13: Broadcast and multicast frames are passed to the MAC control sublayer or to the client.

Row 7.24-14: Others' unicast frames are passed to the client when flood-reception is enabled.

Row 7.24-15: Others' unicast frames are not accepted when flooding is disabled.

Row 7.24-16: If the client wants bad-payload data frames, copy them to the client. (But still increment the error counter.)

Row 7.24-17: Client data frames are counted before being passed to the client.

Row 7.24-18: Control frames are counted before being passed to the MAC control sublayer.

7.6.3.12 ReceiveFilterDataCount state machine

The ReceiveFilterDataCount state machine implements the functions necessary for updating statistics for received and copied data frames.

The following subclauses (7.6.3.12.1–7.6.3.12.3) describe parameters used within the context of this state machine.

7.6.3.12.1 ReceiveFilterDataCount state machine definitions

CLASS_A0

CLASS_A1

CLASS_B

CLASS_C

See 7.2.4.

NULL

See 7.2.1.

Q_RX_F_D_CNT

See 7.2.1.

Q_RX_SAS

See 7.2.1.

7.6.3.12.2 ReceiveFilterDataCount state machine variables

frame

The contents of an RPR frame.

myEdgeState

See 7.2.2.

myRxFilter

See 7.2.4.

size

The size, in bytes, of an RPR frame.

toClientBcastFrames

toClientMcastFrames

See 7.2.5.

toClientMcastClassABytes
toClientMcastClassAFrames
toClientMcastClassBCirBytes
toClientMcastClassBCirFrames
toClientMcastClassBEirBytes
toClientMcastClassBEirFrames
toClientMcastClassCBytes
toClientMcastClassCFrames

See 7.2.5.

toClientUcastClassABytes
toClientUcastClassAOctets
toClientUcastClassBCirBytes
toClientUcastClassBCirFrames
toClientUcastClassBEirBytes
toClientUcastClassBEirFrames
toClientUcastClassCBytes
toClientUcastClassCFrames

See 7.2.5.

7.6.3.12.3 ReceiveFilterDataCount state machine routines

Broadcast(address)

See 7.2.3.

Dequeue(queue)

Enqueue(queue, frame)

See 7.2.3.

Multicast(address)

See 7.2.3.

SameSide(frame)

See 7.2.3.

SizeOf(frame)

See 7.2.3.

7.6.3.12.4 ReceiveFilterDataCount state table

The ReceiveFilterDataCount state machine specified in Table 7.25 implements the functions necessary for counting bytes and frames to be copied to the client. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.25—ReceiveFilterDataCount state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_RX_F_D_CNT)) != NULL	1	size = SizeOf(frame);	UCAST
	—	2	—	START

Table 7.25—ReceiveFilterDataCount state table (continued)

Current state		Row	Next state	
state	condition		action	state
UCAST	Broadcast(frame.da)	3	toClientBcastFrames += 1;	MCAST
	Multicast(frame.da)	4	toClientMcastFrames += 1;	
	frame.sc == CLASS_A0	5	toClientUcastClassAFrames += 1; toClientUcastClassABytes += size;	FINAL
	frame.sc == CLASS_A1	6		
	frame.sc == CLASS_B && frame.fe == 0	7	toClientUcastClassBCirFrames += 1; toClientUcastClassBCirBytes += size;	
	frame.sc == CLASS_B && frame.fe == 1	8	toClientUcastClassBEirFrames += 1; toClientUcastClassBEirBytes += size;	
	frame.sc == CLASS_C	9	toClientUcastClassCFrames += 1; toClientUcastClassCBytes += size;	
MCAST	frame.sc == CLASS_A0	10	toClientMcastClassAFrames += 1; toClientMcastClassABytes += size;	FINAL
	frame.sc == CLASS_A1	11		
	frame.sc == CLASS_B && frame.fe == 0	12	toClientMcastClassBCirFrames += 1; toClientMcastClassBCirBytes += size;	
	frame.sc == CLASS_B && frame.fe == 1	13	toClientMcastClassBEirFrames += 1; toClientMcastClassBEirBytes += size;	
	frame.sc == CLASS_C	14	toClientMcastClassCFrames += 1; toClientMcastClassCBytes += size;	
FINAL	—	15	Enqueue(Q_RX_SAS, frame);	START

Row 7.25-1: Fetch a new input frame.**Row 7.25-2:** Refetch an input frame, after each NULL value is returned.**Row 7.25-3:** Broadcast frames have distinct counters.**Row 7.25-4:** Multicast frames have distinct counters.**Row 7.25-5:** Count the number of unicast classA frames and bytes, subclassA0 contribution.**Row 7.25-6:** Count the number of unicast classA frames and bytes, subclassA1 contribution.**Row 7.25-7:** Count the number of unicast classB-CIR frames and bytes.**Row 7.25-8:** Count the number of unicast classB-EIR frames and bytes.**Row 7.25-9:** Count the number of unicast classC frames and bytes.**Row 7.25-10:** Count the number of multicast classA frames and bytes, subclassA0 contribution.**Row 7.25-11:** Count the number of multicast classA frames and bytes, subclassA1 contribution.**Row 7.25-12:** Count the number of multicast classB-CIR frames and bytes.**Row 7.25-13:** Count the number of multicast classB-EIR frames and bytes.**Row 7.25-14:** Count the number of multicast classC frames and bytes.**Row 7.25-15:** Pass the frames upward toward the client, through SAS.

7.6.3.13 ReceiveFilterControlCount state machine

The ReceiveFilterControlCount state machine implements the functions necessary for updating statistics for received and copied control frames.

The following subclauses (7.6.3.13.1–7.6.3.13.3) describe parameters used within the context of this state machine.

7.6.3.13.1 ReceiveFilterControlCount state machine definitions

CT_FDD
See 7.2.4.
CT_LRTT_REQ
CT_LRTT_RSP
See 7.2.4.
CT_OAM_ECHO_REQ
CT_OAM_ECHO_RSP
CT_OAM_FLUSH
CT_OAM_ORG
CT_OAM_PIRC_STATUS
CT_OAM_SAS_NOTIFY
CT_STATION_ATD
CT_TOPO_CHKSUM
CT_TOPO_PROT
See 7.2.4.
NULL
See 7.2.1.
Q_RX_F_C_CNT
See 7.2.1.
Q_RX_ATD
Q_RX_ECHO_REQ
Q_RX_ECHO_RSP
Q_RX_FAIR
Q_RX_FDD
Q_RX_FLUSH
Q_RX_LRTT_REQ
Q_RX_LRTT_RSP
Q_RX_ORG
Q_RX_SAS_NOTIFY
Q_RX_TC
Q_RX_TP
See 7.2.1.

7.6.3.13.2 ReceiveFilterControlCount state machine variables

badControlTypeFrames

Counter for frames destined to the MAC control sublayer with an invalid *controlType*.

frame

The contents of an RPR frame.

toCtrlFrames

See 7.2.5.

*toCtrlFairnessFrames**toCtrlFddFrames**toCtrlLrttReqFrames**toCtrlLrttRspFrames**toCtrlOamEchoReqFrames**toCtrlOamEchoRspFrames**toCtrlOamFlushFrames**toCtrlOamOrgFrames**toCtrlOamPircStatusFrame**toCtrlOamSasNotifyFrames**toCtrlTopoATDFrames**toCtrlTopoSumFrames**toCtrlTopoTPFrames*

See 7.2.5.

7.6.3.13.3 ReceiveFilterControlCount state machine routines

*Dequeue(queue)**Enqueue(queue, frame)*

See 7.2.3.

7.6.3.13.4 ReceiveFilterControlCount state table

The ReceiveFilterControlCount state machine specified in Table 7.26 implements the functions necessary for counting control frames to be copied to the MAC control sublayer. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.26—ReceiveFilterControlCount state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_RX_F_C_CNT)) != NULL	1	—	COUNT
	—	2	—	START

Table 7.26—ReceiveFilterControlCount state table (continued)

Current state		Row	Next state	
state	condition		action	state
COUNT	frame.ft == FT_FAIRNESS	3	toCtrlFairnessFrames += 1; Enqueue(Q_RX_FAIR, frame);	FINAL
	frame.controlType == CT_STATION_ATD	4	toCtrlTopoATDFrames += 1; Enqueue(Q_RX_ATD, frame);	
	frame.controlType == CT_TOPO_PROT	5	toCtrlTopoTPFrames += 1; Enqueue(Q_RX_TP, frame);	
	frame.controlType == CT_TOPO_CHKSUM	6	toCtrlTopoSumFrames += 1; Enqueue(Q_RX_TC, frame);	
	frame.controlType == CT_LRTT_REQ	7	toCtrlLrttReqFrames += 1; Enqueue(Q_RX_LRTT_REQ, frame);	
	frame.controlType == CT_LRTT_RSP	8	toCtrlLrttRspFrames += 1; Enqueue(Q_RX_LRTT_RSP, frame);	
	frame.controlType == CT_FDD	9	toCtrlFddFrames += 1; Enqueue(Q_RX_FDD, frame);	
	frame.controlType == CT_OAM_ECHO_REQ	10	toCtrlOamEchoReqFrames += 1; Enqueue(Q_RX_ECHO_REQ, frame);	
	frame.controlType == CT_OAM_ECHO_RSP	11	toCtrlOamEchoRspFrames += 1; Enqueue(Q_RX_ECHO_RSP, frame);	
	frame.controlType == CT_OAM_FLUSH	12	toCtrlOamFlushFrames += 1; Enqueue(Q_RX_FLUSH, frame);	
	frame.controlType == CT_OAM_ORG	13	toCtrlOamOrgFrames += 1; Enqueue(Q_RX_ORG, frame);	
	frame.controlType == CT_OAM_SAS_NOTIFY	14	toCtrlOamSasNotifyFrames += 1; Enqueue(Q_RX_SAS_NOTIFY, frame);	
	frame.controlType == CT_OAM_PIRC_STATUS	15	toCtrlOamPircStatusFrames += 1; Enqueue(Q_RX_PIRC_STATUS, frame);	
	—	16	badControlTypeFrames += 1;	START
FINAL	—	17	toCtrlFrames += 1;	START

Row 7.26-1: Fetch a new input frame.**Row 7.26-2:** Refetch an input frame, after each NULL value is returned.**Row 7.26-3:** Count the number of fairness frames.**Row 7.26-4:** Count the number of station attribute discovery frames.**Row 7.26-5:** Count the number of topology and protection frames.**Row 7.26-6:** Count the number of topology checksum frames.**Row 7.26-7:** Count the number of loop round-trip time request frames.**Row 7.26-8:** Count the number of loop round-trip time response frames.**Row 7.26-9:** Count the number of fairness differential delay frames.**Row 7.26-10:** Count the number of OAM echo request frames.**Row 7.26-11:** Count the number of OAM echo response frames.

Row 7.26-12: Count the number of OAM flush frames.

Row 7.26-13: Count the number of OAM organization-specific frames.

Row 7.26-14: Count the number of SAS notify frames.

Row 7.26-15: Count the number of PIRC status frames.

Row 7.26-16: Do not count any other frames.

Row 7.26-17: Count all the frames in aggregate and pass them to the MAC control sublayer.

7.6.4 WrongRinglet state machine

This subclause specifies the state machine for controlling the variable *tossWrongRingletIDs* used in the ReceiveCheck state machine (see Table-Row 7.20-38). This variable causes the MAC to discard frames received with the wrong *ri* (ringlet identifier), as needed to support containment on wrapping systems.

Frames with the wrong *ri* are discarded as part of context containment during unwrap events. During normal operation, frames with the wrong *ri* can occur due only to a noncompliant implementation injecting the frame and are discarded by the ReceiveCheck state machine.

NOTE—The WrongRinglet state machine is not part of the receive entity (or filter entity) state machines. The WrongRinglet state machine runs in parallel with the receive entity state machines.

The following subclauses (7.6.4.1–7.6.4.3) describe parameters used within the context of this state machine.

7.6.4.1 WrongRinglet state machine definitions

DELAY_TO_CHECK

Time to delay wrong-ringlet discarding while waiting for a protection event.

DELAY_TO_RESTORE

Time to delay restore wrong-ringlet discard delays, after a missing protection event. Restoring the wrong-ringlet discard delay eliminates false discards when a following protection event occurs.

STEERING

See 7.2.4.

7.6.4.2 WrongRinglet state machine variables

checkTime

The time at which check for an edge.

currentTime

See 7.2.2.

protConfig

See 7.2.4.

topoType

See 7.2.4.

tossWrongRingletIDs

See 7.2.2.

wrongRingletSensed

See 7.2.2.

7.6.4.3 WrongRinglet state machine routines

No routines are used by this state machine.

7.6.4.4 WrongRinglet state table

The WrongRinglet state machine specified in Table 7.27 implements the functions necessary for determining when to discard frames received on the wrong ringlet. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.27—WrongRinglet state table

Current state		Row	Next state	
state	condition		action	state
START	protConfig == STEERING	1	—	START
	wrongRingletSensed != 0 && topoType == CLOSED_RING	2	checkTime = currentTime + DELAY_TO_CHECK; wrongRingletSensed = 0;	WRONG
	—	3	tossWrongRingletIDs = 0;	START
WRONG	currentTime > checkTime	4	checkTime = currentTime + DELAY_TO_RESTORE; tossWrongRingletIDs = 1;	TOSS
	topoType == OPEN_RING	5	tossWrongRingletIDs = 0;	START
	—	6	tossWrongRingletIDs = 0;	WRONG
TOSS	currentTime > checkTime	7	wrongRingletSensed = 0; tossWrongRingletIDs = 0;	START
	topoType == OPEN_RING	8		
	—	9	wrongRingletSensed = 0; tossWrongRingletIDs = 1;	TOSS

Row 7.27-1: Steering stations do not support this state machine.

Row 7.27-2: Actions begin after a data frame from the other ringletID is observed.

Row 7.27-3: While waiting for action, the *tossWrongringletIDs* is cleared and frames pass through.

Row 7.27-4: Wait for a small time delay before discarding frames.

Row 7.27-5: Abort the timeout when a protection edge is detected.

Row 7.27-6: Continue passing wrong-ringlet frames until the first (~10 milliseconds) timeout is reached.

Row 7.27-7: Abort the timeout if the worst-case containment would have completed.

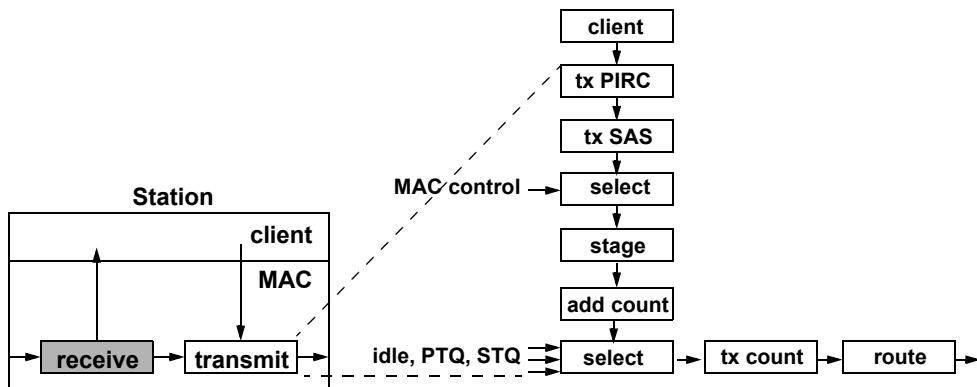
Row 7.27-8: Abort the timeout when a protection edge is detected.

Row 7.27-9: Continue discarding wrong-ringlet frames until the second (~1 second) timeout is reached.

7.7 Transmit operation

The transmit operation is initiated upon receipt of a MA_DATA.request (see 6.4.1), a MA_CONTROL.request (see 6.4.3), or any receipt of a frame from the control sublayer.

Each station has transmit, receive, and filter entities, as illustrated in Figure 7.11, which use rules designed to support local and bridged RPR traffic

**Figure 7.11—Transmit entities**

The transmit entity includes the following subentities, which are detailed in 7.7.1–7.7.9:

- protected inter-ring connection sublayer (see 15.5.2)
- spatially aware sublayer (see 14.5.3)
- ringlet selection and frame field initialization (see 7.7.1 and 7.7.2.1)
- stage queue selection (see 7.7.4)
- add traffic statistics (see 7.7.5 and 7.7.6)
- transmit selection (see 7.7.7 or 7.7.8)
- transmit traffic statistics (see 7.7.9).

7.7.1 Ringlet selection

This subclause describes how a ringlet is chosen. In most cases, a single ringlet, either ringlet0 or ringlet1, is chosen. In some cases, both ringlets can be chosen; in which case, ringlet selection is also responsible for replicating the frame onto both ringlets.

Ringlet selection is performed for client add traffic and for OAM add frames (see 12.1.3).

Ringlet selection performs the following actions:

- Chooses the appropriate ringlet(s) on which to send frames.
- Possibly redirects frames based on protection settings for the frames and for the station.
- Possibly changes the destination address placed in the frame based on optional evaluation of secondary destination address settings.
- Determines if frames need to be flooded.
- Sets the *sa*, *da*, *saExtended*, *daExtended*, *ef*, *ttl*, *ttlBase*, *ri*, *we*, *fi*, *ps*, and *so* fields of frames.

The primary action for ringlet selection is to choose the appropriate ringlet for client add frames and OAM add frames. The choice is based on the *ringlet_id* parameter, the *mac_protection* parameter, and the topology and status database. The result is the setting of the *ri* field of the frame(s). The client can exercise complete control, no control, or partial control over the path(s) that a frame takes.

For values of the destination_address parameter that are not contained in the topology and status database, ringlet selection floods the frame to all stations on the ring. Based on the protection state of the ring and on the flooding preference configuration, the flooding can be done on only one ringlet or on both ringlets. Flooding on one ringlet is indicated by a value in the *fi* (flooding indication) field of FI_UNIDIR. Flooding on both ringlets is indicated by a value in the *fi* field of FI_BIDIR and causes replication of the frames for transmission on both ringlets.

NOTE—Frames accepted and processed by ringlet selection go into per-ringlet logical queue entities of 0 to infinite bytes. Short queue length could cause head of line blocking.

When protection is active, ringlet selection optionally restears add frames for steering rings or for local attachment failure in wrapping rings using a center wrap scheme, or optionally replicates broadcast and multicast frames for transmission on both ringlets. The *we* (wrap eligible) field is set based on the optional mac_protection parameter (see 6.4.1) and the value of *protConfig* (see 11.2.5).

NOTE—The above mechanism allows the coexistence of steered and wrapped frames from the same client, on wrapping rings.

Additionally, once the ringlet choice has been made, the *ttl* (time to live) and *ttlBase* (time to live base) fields are set as described in 7.7.2.1.

7.7.1.1 Relationship to other clauses

Ringlet selection makes use of the topology and status database (see 11.5). The following is the information in the database that is used by ringlet selection:

- a) The topology of the ring. The ring topology is used for determining the distance to another station from the local station and for deciding to flood a frame if the location can not be determined.
- b) Availability of other stations. The availability of the path to a destination station is used for modifying the ringlet choice for protected frames on steering rings.
- c) Protection mechanism. The protection mechanism of the ring is used to determine whether to use steering or wrapping as the protection mechanism for the station.

7.7.1.2 Client control of ringlet selection

Ringlet selection is invoked for each frame generated by the MA_DATA.request primitive and each OAM frame generated by the MAC control sublayer. The selection is controlled via the optional ringlet_id parameter and the optional mac_protection parameter, both of which are set by the MA_DATA.request primitive supplied by the client (see 6.4.1), or by the OAM entity of the MAC control sublayer.

If unspecified, the value for ringlet_id shall default to RI_DEFAULT.

A client is expected to request to send a frame only when allowed by the send indication for the service class requested and for the ringlet requested. For a ringlet choice of RI_DEFAULT, or a ringlet choice that can be overridden due to steering protection, the client cannot know in advance which ringlet is to be chosen and therefore should use the send indications from both ringlets. This can be represented as follows:

$$\text{clientSendX} = \text{Min}(\text{ringlet0sendX}, \text{ringlet1sendX}) \quad (7.20)$$

NOTE—For the purposes of Equation (7.20), FALSE is assumed to have an integer value of 0, and TRUE is assumed to have an integer value of 1. For *sendA* and *sendB*, this operation is equivalent to a logical AND operation.

See 7.5 for a description of the send indications.

NOTE—A client using RI_DEFAULT, either by choice or by default, can experience head of line blocking while waiting for the opposite ringlet from which its frame is to be sent to become free.

If unspecified, the value for mac_protection shall default to TRUE.

7.7.1.2.1 RI_0, MAC protected

A unicast frame with a ringlet_id value of RI_0 and a mac_protection value of TRUE is sent on ringlet0, if it is a wrapping ring. If the destination address is not reachable on ringlet0 for a steering ring, and is reachable on ringlet1, the frame is sent on ringlet1. If a destination address is not reachable on ringlet0 for a center wrapped ring, the frame is transmitted on ringlet1 via the ringlet0 datapath.

For broadcast and multicast frames being sent on wrapping rings, the ringlet decision is the same as for unicast frames. For steering rings, broadcast and multicast frames can be sent in either or both directions, as necessary to reach all of the destination addresses.

For wrapping rings, the *we* (wrap eligible) field is set. For steering rings, the *we* field has no meaning and is not set.

7.7.1.2.2 RI_0, not MAC protected

A frame with a ringlet_id value of RI_0 and a mac_protection value of FALSE is sent on ringlet0, regardless of state of ringlet0 ringlet.

For unicast addresses, whether the frame arrives at the destination depends upon whether any links with a non-IDLE protection state exist before the destination.

For broadcast, multicast, and flooded frames, the frame can make it to all of the destination addresses, some of the destination addresses, or none of the destination addresses, depending upon whether any links with a non-IDLE protection state exist before the (final) destination of the frame.

For all rings, the *we* field is not set.

7.7.1.2.3 RI_1, MAC protected

A unicast frame with a ringlet_id value of RI_1 and a mac_protection value of TRUE is sent on ringlet1, if it is a wrapping ring. If the destination address is not reachable on ringlet1 for a steering ring, and is reachable on ringlet0, the frame is sent on ringlet0. If a destination address is not reachable on ringlet1 for a center wrapped ring, the frame is transmitted on ringlet0 via the ringlet1 datapath.

For broadcast and multicast frames being sent on wrapping rings, the ringlet decision is the same as for unicast frames. For steering rings, broadcast and multicast frames can be sent in either or both directions, as necessary to reach all of the destination addresses.

For wrapping rings, the *we* field is set. For steering rings, the *we* field has no meaning and is not set.

7.7.1.2.4 RI_1, not MAC protected

A frame with a ringlet_id value of RI_1 and a mac_protection value of FALSE is sent on ringlet1, regardless of state of ringlet1 ringlet.

For unicast addresses, whether the frame arrives at the destination depends upon whether any links with a non-IDLE protection state exist before the destination.

For broadcast, multicast, and flooded frames, the frame can make it to all of the destination addresses, some of the destination addresses, or none of the destination addresses, depending upon whether any links with a non-IDLE protection state exist before the (final) destination of the frame.

For both wrapping and steering rings, the *we* field is not set.

7.7.1.2.5 RI_DEFAULT, MAC protected

A frame with a ringlet_id value of RI_DEFAULT and a mac_protection value of TRUE is sent on either or both ringlets, as chosen by the MAC.

The algorithm used to choose ringlets is implementation specific. The only constraint on the algorithm is that it chooses the same direction for all frames with the same {sa, da, service_class} tuple, until protection or topology changes. (See also 7.6.1.)

For wrapping rings, the *we* field is set. For steering rings, the *we* field has no meaning and is not set.

7.7.1.2.6 RI_DEFAULT, not MAC protected

A frame with a ringlet_id value of RI_DEFAULT and a mac_protection value of FALSE is sent on either or both ringlets, as chosen by the MAC, with no allowance for unicast, broadcast, or multicast addresses beyond any links with a non-IDLE protection state.

For unicast addresses, whether the frame arrives at the destination depends upon whether any links with a non-IDLE protection state exist before the destination.

For broadcast, multicast, and flooded frames, the frame can make it to all of the destination addresses, some of the destination addresses, or none of the destination addresses, depending upon whether any links with a non-IDLE protection state exist before the (final) destination of the frame.

The algorithm used to choose ringlets is implementation specific. The only constraint on the algorithm is that it chooses the same direction for all frames with the same {sa, da, service_class} tuple, until protection or topology changes. (See also 7.6.1.)

For both wrapping and steering rings, the *we* field is not set.

7.7.1.3 Flooding determination

Ringlet selection uses the parameters of the MA_DATA.request, the default flooding preference given in *myFloodingForm*, and the information contained in the topology and protection database to determine whether and how to flood a frame. The default flooding preference is not changeable, except at the time when topology or protection changes, in order to avoid reordering of frames.

Two flooding alternatives for data frames are provided. They are as follows:

unidirectional:

A frame forwarding transfer involving sending a flooding frame on only one ringlet, to all stations on that ringlet.¹⁸ The *sa* found in the frame header is the local source MAC address. The *fi* field is set to FI_UNIDIR for this flooding alternative.

¹⁸A potentially more efficient form of unidirectional flooding can also be supported. This is a unidirectional transmission that terminates at the source station's upstream neighbor. The *sa* found in the frame header is the local source address. The *ttl* and *ttlBase* fields in the frame header are set to *numStations*-1, upon transmission by the source station. The *fi* field is set to FI_UNIDIR.

bidirectional:

A frame forwarding transfer involving sending two flooding frames, one on each ringlet, where each frame is directed to distinct adjacent stations. The scoping of the flooded frames is primarily governed by the *ttl* within the frame header. The *fi* field is set to FI_BIDIR for this flooding alternative.

7.7.1.4 Flooding rules

The transmit rules affecting the selection of *fi* (flooding indication) field values and the use of basic or extended format are summarized in Table 7.28. This table is intended to summarize rules that are embodied within the RingletSelection state machine specified in 7.7.1.6. In the case of any ambiguity between this summary and the RingletSelection state machine specified in Table 7.29, the state machine shall take precedence.

Table 7.28—Flooding rules

Client parameters		Row	Header parameters	Payload parameters	
destination_address	source_address		<i>fi</i>	daExtended	saExtended
—	$\neq myMacAddress$	1	$\neq FI_NONE$	destination_address	source_address
local	$= myMacAddress$	2	$= FI_NONE$	-na-	-na-
—	$= myMacAddress$	3	$\neq FI_NONE$		

NOTE 1—*frame.da* is always set to the destination_address parameter.
 NOTE 2—*frame.sa* is always set to the source_address parameter.

Row 7.28-1: Flood remote source frames using the extended frame format. Including the local station's MAC address as the source address allows the frame to be reliably source stripped, rather than accidentally learned, when unexpectedly passing by the source station.

Row 7.28-2: Do not flood locally sourced frames with a destination_address known to be local to the ring. Use the basic frame format.

Row 7.28-3: Flood other locally sourced frames using the basic frame format.

NOTE—All stations see a flooded frame, but sophisticated sources can restrict flood scoping as follows:

- a) Multicast: Scoping can be reduced to those stations within the multicast group.
- b) Unknown unicast: Scoping can be reduced to those bridge-capable stations.

7.7.1.5 Secondary MAC address substitution

Ringlet selection can support destination MAC address substitution. This is enabled by stations advertising that they wish to receive traffic being sent to one or two extra MAC addresses (also known as secondary MAC addresses) in addition to their station MAC addresses (also known as primary MAC addresses) (see 11.6.13). When given a frame transmit request with a destination address with a value of another station's secondary MAC address, the sending station can substitute the associated primary MAC address for the provided secondary MAC address. This allows these frames to be sent without flooding and to get the benefit of destination stripping. Substitution of secondary MAC addresses is transparent to all portions of the MAC (including the remainder of ringlet selection) other than the optional substitution itself, which is detailed in Table 7.29, Row 20. The advertisement of secondary MAC addresses, action taken upon receipt of such an advertisement, and address substitution of secondary MAC addresses are all optional.

7.7.1.6 RingletSelection state machine

This subclause specifies the state machine for selecting the ringlet and protection values for client frames being added.

The following subclauses (7.7.1.6.1–7.7.1.6.3) describe parameters used within the context of this state machine.

7.7.1.6.1 RingletSelection state machine definitions

CENTER_WRAP

See 7.2.2.

FI_BIDIR

FI_NONE

FI_UNIDIR

See 7.2.4.

FLOOD_BIDIR

FLOOD_NONE

FLOOD_UNIDIR

See 7.2.4.

FT_CONTROL

See 7.2.4.

NULL

See 7.2.1.

OPEN_RING

See 7.2.4.

Q_TX_RS

Q_TX_SS

See 7.2.1.

RI_0

RI_1

See 7.2.4.

RINGLET_0

RINGLET_1

See 7.2.4.

STEERING

WRAPPING

See 7.2.4.

7.7.1.6.2 RingletSelection state machine variables

containedFrames

See 7.2.2.

containmentActive

See 7.2.4.

destination_address

See 7.2.4.

flooding_form

See 7.2.4.

frame

The RPR data frame (see 9.2) constructed from the MA_DATA.request() primitive (see 6.4.1).

frame0

A copy of *frame*, to send on ringlet0.

frame1

A copy of *frame*, to send on ringlet1.

*mac_protection**mark_fe*

See 7.2.4.

myFloodingForm

The preferred flooding form to use for frames that need to be flooded and for which no flooding_form value was provided.

FLOOD_BIDIR—The default flooding form is bidirectional.

FLOOD_UNIDIR—The default flooding form in unidirectional.

myMacAddress

See 7.2.4.

myProtectMethod

See 7.2.2.

optionSecMacSubstitute

Available for stations wishing to substitute primary MAC addresses for secondary MAC addresses.

TRUE—Primary MAC addresses will be substituted for secondary MAC addresses.

FALSE—(Otherwise.)

protConfig

See 7.2.4.

*ringlet_id**source_address**source_address_extended*

See 7.2.4.

strict_order

See 7.2.4.

topoEntry

See 7.2.4.

7.7.1.6.3 RingletSelection state machine routines

CheckEnqueueRi(ringlet, queue, frame)

If frame.ttl is greater than 0, then call *EnqueueRi(ringlet, queue, frame)*.

*Crc16(frame)**Crc32(frame)*

See 7.2.3.

DefaultRinglet()

See 7.2.3

Dequeue(queue)

See 7.2.3.

EdgeAllowedFrame(frame)

See 7.2.3.

EdgeOfRing()

To indicate if this station is at an edge of the ring as determined by the boolean expression (ringlet0.myEdgeState == INTO_EDGE || ringlet1.myEdgeState == INTO_EDGE).

TRUE—This station is at an edge of the ring.

FALSE—(Otherwise.)

EnqueueRi(ringlet, queue, frame)

See 7.2.3.

Extend(frame)

Modifies *frame* from a basic data frame to an extended data frame.

InitialTtl(frame)

Returns the initial value for the *ttl* field of *frame* given the values of *da* and *ri* within the constraints detailed in 7.7.2.1, including optionally searching the SDB static group address table for a matching static entry.

IsEdge(ringlet)

Indicates whether the specified ringlet is transmitting into an edge.

MyEdgeRinglet()

See 7.2.3.

Other()

See 7.2.3.

PriAddress(secondaryMacAddress)

Returns the primary address associated with the supplied *secondaryMacAddress*, if such an association exists and is in use (see 11.6.13).

macAddress—The associated primary address.

NULL—No associated primary address.

Provided(parameter)

See 7.2.3.

Reachable(address, ringlet)

See 7.2.3.

Replicate(frame)

See 7.2.3.

EnqueueRi(ringlet, queue, frame)

See 7.2.3.

Unicast(address)

See 7.2.3.

7.7.1.6.4 RingletSelection state table

The RingletSelection state machine specified in Table 7.29 implements the functions necessary for client frame admission and the associated frame header processing. Optional rows are shaded in gray. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.29—RingletSelection state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_TX_RS)) == NULL	1	—	START
	ringlet_id == RI_0 ringlet_id == RI_1	2	—	PASSED
	—	3	ringlet_id = DefaultRinglet();	
PASSED	—	4	frame.ps = 0;	STRICT

Table 7.29—RingletSelection state table (continued)

Current state		Row	Next state	
state	condition		action	state
STRICT	frame.ft == FT_CONTROL && frame.controlType == CT_FDD	5	frame.da = destination_address; frame.sa = myMacAddress; frame.ri = ringlet_id;	TRANSMIT
	frame.ft == FT_CONTROL	6	frame.da = destination_address; frame.sa = myMacAddress; frame.ri = ringlet_id; frame.we = 0; frame.ef = 0; frame.fi = FI_NONE; frame.so = 0;	
	containmentActive && (transmissionControl == STRICT)	7	containedFrames += 1;	START
	transmissionControl == STRICT	8	frame.so = 1;	PROTECT
	containmentActive && (!Provided(strict_order) strict_order)	9	containedFrames += 1;	START
	!Provided(strict_order) strict_order	10	frame.so = 1;	PROTECT
	—	11	frame.so = 0;	
PROTECT	!mac_protection	12	frame.ri = ringlet_id; frame.we = 0;	EXTEND
	frame.so && protConfig == WRAPPING && EdgeOfRing() && myProtectMethod == CENTER_WRAP	13	frame.ri = ringlet_id; frame.we = 1; frame.ps = (frame.ri == MyEdgeRinglet());	
	protConfig == WRAPPING && EdgeOfRing() && myProtectMethod == CENTER_WRAP	14	frame.ri = Other(MyEdgeRinglet()); frame.we = 1;	
	protConfig == WRAPPING	15	frame.ri = ringlet_id; frame.we = 1;	
	—	16	frame.ri = ringlet_id; frame.we = 0;	

Table 7.29—RingletSelection state table (continued)

Current state		Row	Next state	
state	condition		action	state
EXTEND	Provided(source_address_extended)	17	Extend(frame); frame.sa = myMacAddress; frame.da = destination_address; frame.saExtended = source_address_extended; frame.daExtended = destination_address_extended; frame.ef = 1;	DA
	Provided(source_address) && source_address != myMacAddress	18	Extend(frame); frame.sa = myMacAddress; frame.da = destination_address; frame.saExtended = source_address; frame.daExtended = destination_address; frame.ef = 1;	
	—	19	frame.sa = myMacAddress; frame.da = destination_address; frame.ef = 0;	
DA	optionSecMacSubstitute && PriAddress(destination_address) != NULL	20	frame.da = PriAddress (destination_address);	FLOOD
	—	21	—	
FLOOD	!unicast(frame.da)	22	—	FORM
	Provided(flooding_form) && flooding_form == FLOOD_NONE	23	frame.fi = FI_NONE;	TRANSMIT
	Provided(flooding_form) && flooding_form != FLOOD_NONE	24	—	FORM
	Provided(source_address) && source_address != myMacAddress	25	—	
	Reachable(frame.da, ringlet_id)	26	frame.fi = FI_NONE;	TRANSMIT
	frame.we == 1 && Reachable(frame.da, Other(ringlet_id))	27	frame.fi = FI_NONE;	
	mac_protection && Reachable(frame.da, Other(ringlet_id))	28	frame.ri = Other(ringlet_id); frame.fi = FI_NONE;	
	!mac_protection && Reachable(frame.da, Other(ringlet_id))	29	frame.fi = FI_NONE;	
	—	30	—	FORM
FORM	(Provided(flooding_form) && flooding_form == FLOOD_BIDIR) (mac_protection && protConfig == STEERING && topoType == OPEN_RING) (!Provided(flooding_form) && myFloodingForm == FLOOD_BIDIR)	31	frame.fi = FI_BIDIR;	BIEDGE
	—	32	frame.fi = FI_UNIDIR;	TRANSMIT

Table 7.29—RingletSelection state table (continued)

Current state		Row	Next state	
state	condition		action	state
BIEDGE	IsEdge(RINGLET_0) && IsEdge(RINGLET_1)	33	—	START
	IsEdge(RINGLET_0)	34	frame.ri = RINGLET_1;	TRANSMIT
	IsEdge(RINGLET_1)	35	frame.ri = RINGLET_0;	
	—	36	Replicate(frame); frame0.ri = RINGLET_0; frame0.fi = FI_BIDIR; frame0.ttl = InitialTtl(frame0); frame0.ttlBase = frame0.ttl; frame0.fcs = Crc32(frame0); frame1.ri = RINGLET_1; frame1.fi = FI_BIDIR; frame1.ttl = InitialTtl(frame1); frame1.ttlBase = frame1.ttl; frame1.fcs = Crc32(frame1); CheckEnqueueRi(RINGLET_0, Q_TX_SS, frame0); CheckEnqueueRi((RINGLET_1, Q_TX_SS, frame1);	START
TRANSMIT	IsEdge(frame.ri) && (frame.we == 0 protConfig == STEERING) && !EdgeAllowedFrame(frame)	37	—	START
	protConfig == WRAPPING && EdgeOfRing() && myProtectMethod == CENTER_WRAP	38	frame.ttl = InitialTtl(frame); frame.ttlBase = frame.ttl; frame.fcs = Crc32(frame); CheckEnqueueRi(MyEdgeRinglet(), Q_TX_SS, frame);	
	—	39	frame.ttl = InitialTtl(frame); frame.ttlBase = frame.ttl; frame.fcs = Crc32(frame); CheckEnqueueRi(frame.ri, Q_TX_SS, frame);	

Row 7.29-1: No request to transmit a frame. Wait for such a request (generated from the MA_DATA.request primitive, indirectly from the MA_CONTROL.request primitive, or from the transmission of an echo response frame).

Row 7.29-2: The request included a choice of a specific ringlet.

Row 7.29-3: The request included a choice of the default ringlet or did not include a choice. Assign a first choice for the ringlet.

Row 7.29-4: The newly transmitted frame has not yet passed its source on the opposing ringlet.

Row 7.29-5: FDD control frames have their *frame.we* value set by Table 10.14, and they pass the remaining checks.

Row 7.29-6: Control frames bypass the remaining checks.

Row 7.29-7: Strict frames are discarded when *containmentActive* is set.

Row 7.29-8: The MAC is configured to transmit all of its frames with strict ordering required.

Row 7.29-9: Strict frames are discarded when *containmentActive* is set.

Row 7.29-10: The client has requested strict ordering for this frame, or had it assigned by default.

Row 7.29-11: The client has requested relaxed ordering for this frame.

Row 7.29-12: The client has requested no MAC protection, so the frame is not wrap eligible. Prepare the frame for transmit on the requested ringlet.

Row 7.29-13: This is a strict data frame being transmitted onto a wrapping ring by a center-wrapped station. Prepare the frame for transmit on the requested ringlet, with no adjustment for center wrap. (The ringlet is not allowed to be changed for strict data frames.)

Row 7.29-14: This is a wrapping ring, and the station is wrapped with a center wrap. Prepare the frame for transmit on the wrapped ringlet. (This allows a center wrap station to transmit a frame onto the ring with the opposite ringlet identifier of what was requested because the frame will immediately be wrapped to the opposite ringlet anyway.)

Row 7.29-15: This is a wrapping ring. Prepare the frame for transmit on the requested ringlet.

Row 7.29-16: This is a steering ring. Prepare the frame for transmit on the requested ringlet.

Row 7.29-17: The frame explicitly originates from beyond this station. Use the extended data frame format.

Row 7.29-18: The frame implicitly originates from beyond this station. Use the extended data frame format.

Row 7.29-19: The frame originates from this station. Retain the basic data frame format.

Row 7.29-20: (optional) The destination_address is a secondary address of one of the stations on the ring. Use the primary address of the addressed station for the *da* field value. (If the frame is an extended data frame format, *daExtended* is left set to the secondary address.)

Row 7.29-21: (optional) The destination_address is not a secondary address of one of the stations on the ring. Retain the existing *da* field value.

Row 7.29-22: The destination address is a unicast address, and therefore, the frame is flooded.

Row 7.29-23: The client has requested the frame not be flooded, regardless of the MAC's decision on the need for flooding.

Row 7.29-24: The client has requested the frame to be flooded, regardless of the MAC's decision on the need for flooding.

Row 7.29-25: The frame is apparently bridged, and therefore, the frame is flooded.

Row 7.29-26: The destination address is a unicast address, and the topology and status database indicates that it is reachable on the requested ringlet. Prepare the frame for transmit on the requested ringlet.

Row 7.29-27: The destination address is a unicast address, the topology and status database indicates that it is reachable on the opposing ringlet to the requested ringlet, and it can be wrapped onto the opposing ringlet to the requested ringlet. Prepare the frame for transmit on the requested ringlet.

Row 7.29-28: The destination address is a unicast address, the topology and status database indicates that it is reachable on the opposing ringlet to the requested ringlet, and the frame is protected. Prepare the frame for transmit on the opposing ringlet to the requested ringlet. (Sending a strict frame onto the other ringlet will not cause reordering because whatever caused the destination address to be unreachable on the original ringlet caused context containment, and strict frames were deleted during context containment in Table 7.29, Row 7 and Table 7.29, Row 9.)

Row 7.29-29: The destination address is a unicast address, the topology and status database indicates that it is reachable on the opposing ringlet to the requested ringlet, and the frame is not protected. Prepare the frame for transmit on the requested ringlet.

Row 7.29-30: The destination address is a unicast address not known to be reachable on ringlet0 or on ringlet1, or the destination address is a multicast address; therefore, the frame must be flooded.

Row 7.29-31: The default flooding form is bidirectional, or bidirectional flooding is necessary (and allowed) to make sure all stations get a chance to see the frame. Set the *fi* field to FI_BIDIR.

Row 7.29-32: The frame is not desired, needed, or allowed to be bidirectionally flooded. Set the *fi* field to FI_UNIDIR.

Row 7.29-33: Both ringlets are transmitting into edges. There is nowhere to transmit the frame.

Row 7.29-34: Ringlet0 is transmitting into edge. Flood the frame only on ringlet1.

Row 7.29-35: Ringlet1 is transmitting into edge. Flood the frame only on ringlet0.

Row 7.29-36: Replicate the frame and prepare one copy of the frame for transmission on ringlet0 and the other copy of the frame for transmission on ringlet1. Move the frames to the selection queues of each ringlet.

Row 7.29-37: The frame is heading into the edge, and is not allowed to transmit across an edge.

Row 7.29-38: For center-wrapped stations, transmit the frame via the selection queue of the ringlet that is heading into the edge.

Row 7.29-39: Move the frame to the selection queue of the selected ringlet.

NOTE—The MAC does not transmit frames when the appropriate send indication is not present. The details of this interaction with the client are implementation specific.

7.7.2 Determination of cleave point

A frame can be replicated and sent on both ringlets in the case of a bidirectionally flooded frame or in the case of a broadcast or multicast frame sent on a ring experiencing a protection event. In both cases, the replicated frames are not allowed to possibly overlap their delivery and therefore cannot be sent beyond a point on the ring common to both frames, the cleave point. The potential overlap is prevented by the setting of the *ttl* values in both frames to the number of hops to the cleave point, as determined by the number of hops to the station just short of the cleave point.

In the case of a ring that is currently experiencing a protection event, the cleave point for replicated frames is the point at which the protection event exists, regardless of the value of *stdCleave* (see 6.2.8). If more than one protection event exists, a separate cleave point is chosen for each ringlet, with the cleave point being the first span known to be an edge in the transmit direction of the ringlet.

For non-SAS stations, the determination of the cleave point for replicated frames is implementation specific when the ring is not currently experiencing a protection event. The cleave point should not change unless the topology changes (either topology type or contents of topology), to avoid the possibility of reordering frames apart from the cleave point selection.

NOTE—In order to avoid frame replication, the implementation can choose a cleave point that is immediately adjacent the station.

An example of a cleave point is shown in Figure 7.12. In this example, the cleave point is between station S1 and station S2. Station S4, therefore, sets the *ttl* for the frame traveling on ringlet0 to 4 and the *ttl* for the frame traveling on ringlet1 to 2.

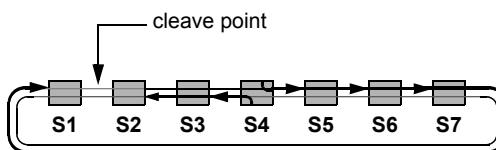


Figure 7.12—Cleave point example

7.7.2.1 Cleave point determination for SAS

To facilitate interoperability of SAS implementations, a station may use a standardized SAS cleave point calculation. This calculation is controlled by the value of *stdCleave* (see 6.2.8).

For rings with an odd number of stations, the cleave point is selected such that an equal number of stations appear to the east and west of the sourcing station, as illustrated in Figure 7.13.

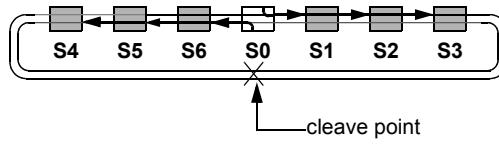
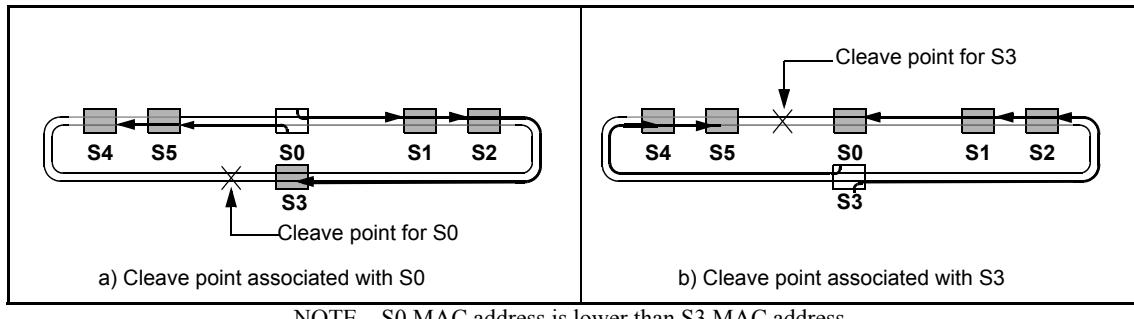


Figure 7.13—Odd number of stations cleave point example

For rings with an even number of stations, cleave points are selected such that diametrically opposite stations send flooded traffic to each other over the same path, as illustrated in Figure 7.14. The station with the lower MAC address sets its cleave point to the east of its opposite station, and the station with the higher MAC address sets its cleave point to the west of its opposite station.



NOTE—S0 MAC address is lower than S3 MAC address.

Figure 7.14—Even number of stations cleave point example

7.7.3 Setting of *ttl* and *ttlBase*

This subclause provides the constraints upon the initial value to which the *ttl* field may be set and the resultant *ttlBase* value. The following subclauses (7.7.3.1–7.7.3.3) describe parameters used within this context.

7.7.3.1 Definitions for use with setting of *ttl* and *ttlBase*

CT_LRTT_REQ
 CT_LRTT_RSP
 See 7.2.4.
 CT_FDD
 CT_OAM_ECHO_REQ
 CT_OAM_ECHO_RSP
 CT_OAM_FLUSH
 CT_OAM_ORG
 CT_OAM_PIRC_STATUS
 CT_OAM_SAS_NOTIFY
 CT_STATION_ATD
 CT_TOPO_CHKSUM
 CT_TOPO_PROT
 See 7.2.4.
 FI_BIDIR

FI_NONE
FI_UNIDIR
 See 7.2.4.
FT_CONTROL
FT_DATA
FT_FAIRNESS
FT_IDLE
 See 7.2.4.
MAX_STATIONS
 See 7.2.4.
MULTI_CHOKE
SINGLE_CHOKE
 See 7.2.4.

7.7.3.2 Variables for use with setting of *ttl* and *ttlBase*

cleavePoint

The location of the cleave point, relative to the ringlet being used for transmission. Specified as the MAC address of the station immediately before the edge.

frame

The contents of an RPR frame.

numStations

See 7.2.4.

optionSasMcastScope

See 7.2.4.

7.7.3.3 Routines for use with setting of *ttl* and *ttlBase*

NumberOfHops(address)

Returns the number of hops from this station to the station identified by the provided address, relative to the ringlet being used for transmission. If the provided address is a group address, this returns the number of hops from this station to the station in the group that is known to be furthest from this station, on the ringlet being used for transmission. If the implementation does not know which stations are in the group, it returns *numStations* - 1.

SasMcastScope(frame)

Returns TRUE, if the *optionSasMcastScope* variable is set to TRUE and the *frame.da* is a group address and the *SdbMcastLookup()* routine returns a value other than NULL. Returns FALSE otherwise.

SdbMcastLookup(frame)

If the frame is extended, then the extended destination address of the frame (and VLAN identifier (VID) if *optionSasVlanAware* is TRUE) is used as the index to the SDB static multicast lookup table. If the frame is not extended, then the destination address of the frame (and VID if *optionSasVlanAware* is TRUE) is used as the index to the SDB static multicast lookup table. If an entry is found (either destination address only, or destination address and VID), the *sasTxMcastScopedFrames* counter is incremented and a valid hop count associated with *frame.ri* is returned. A hop count is valid only if $(sdbStaticMcastHopsRinglet0 + sdbStaticMcastHopsRinglet1) < numStations$. Otherwise, if an entry is not found or a valid hop count cannot be returned, a value of NULL is returned.

Unicast(macAddress)

See 7.2.3.

7.7.3.4 Initial *ttl* field values

Every frame transmitted from the local client or local MAC control sublayer shall have its *ttl* field set according to the constraint rules specified in Table 7.30, where the rows are evaluated in top-to-bottom order. When a given constraint is a range of values, any value within the range is allowed.

Table 7.30—Initial *ttl* field values

ft field value	Additional relevant values	Row	Constraints on initial <i>ttl</i> field value
FT_IDLE	—	1	$\text{ttl} == 1$
FT_CONTROL	frame.controlType == CT_STATION_ATD	2	$\text{ttl} == \text{MAX_STATIONS}$
	frame.controlType == CT_TOPO_CHKSUM	3	$\text{ttl} == 1$
	frame.controlType == CT_TOPO_PROT	4	$\text{ttl} == \text{MAX_STATIONS}$
	frame.controlType == CT_LRTT_REQ	5	$\text{ttl} \geq \text{NumberOfHops}(\text{frame.da}) \&\& \text{ttl} \leq \text{MAX_STATIONS}$
	frame.controlType == CT_LRTT_RSP	6	$\text{ttl} \geq \text{NumberOfHops}(\text{frame.da}) \&\& \text{ttl} \leq \text{MAX_STATIONS}$
	frame.controlType == CT_FDD	7	$\text{ttl} \geq \text{NumberOfHops}(\text{frame.da}) \&\& \text{ttl} \leq \text{MAX_STATIONS}$
	frame.controlType == CT_OAM_ECHO_REQ	8	$\text{ttl} \geq \text{NumberOfHops}(\text{frame.da}) \&\& \text{ttl} \leq \text{MAX_STATIONS}$
	frame.controlType == CT_OAM_ECHO_RSP	9	$\text{ttl} \geq \text{NumberOfHops}(\text{frame.da}) \&\& \text{ttl} \leq \text{MAX_STATIONS}$
	frame.controlType == CT_OAM_FLUSH	10	$\text{ttl} \geq \text{numStations} - 1 \&\& \text{ttl} \leq \text{MAX_STATIONS}$
	frame.controlType == CT_OAM_ORG	11	$\text{ttl} \geq \text{NumberOfHops}(\text{frame.da}) \&\& \text{ttl} \leq \text{MAX_STATIONS}$
	frame.controlType == CT_OAM_SAS_NOTIFY	12	$\text{ttl} \geq \text{numStations} - 1 \&\& \text{ttl} \leq \text{MAX_STATIONS}$
	frame.controlType == CT_OAM_PIRC_STATUS	13	$\text{ttl} \geq \text{numStations} - 1 \&\& \text{ttl} \leq \text{MAX_STATIONS}$
FT_FAIRNESS	frame.ffType == SINGLE_CHOKE	14	$\text{ttl} == \text{MAX_STATIONS}$
	frame.ffType == MULTI_CHOKE	15	$\text{ttl} == \text{MAX_STATIONS}$
FT_DATA	SasMcastScope(frame)	16	$\text{ttl} == \text{SdbMcastLookup}(\text{frame})$
	!SasMcastScope(frame) && (frame.fi == FI_BIDIR)	17	$\text{ttl} == \text{NumberOfHops}(\text{cleavePoint})$
	!SasMcastScope(frame) && (frame.fi == FI_UNIDIR)	18	$\text{ttl} \geq \text{numStations} - 1 \&\& \text{ttl} \leq \text{MAX_STATIONS}$
	!SasMcastScope(frame) && (frame.fi == FI_NONE)	19	$\text{ttl} \geq \text{NumberOfHops}(\text{frame.da}) \&\& \text{ttl} \leq \text{MAX_STATIONS}$

Row 7.30-1: Idle frames always travel one hop.

Row 7.30-2: Station attribute discovery frames are sent without (accurate) knowledge of the size of the ring and are capable of traveling the largest possible ring.

Row 7.30-3: Topology checksum frames always travel one hop.

Row 7.30-4: Topology and protection frames are sent without (accurate) knowledge of the size of the ring and are capable of traveling the largest possible ring.

Row 7.30-5: Loop round-trip time request frames are capable of traveling as far as their intended destination.

Row 7.30-6: Loop round-trip time response frames are capable of traveling as far as their intended destination.

Row 7.30-7: Fairness differential delay frames are capable of traveling as far as their intended destination.

Row 7.30-8: Echo request frames are capable of traveling as far as their intended destination.

Row 7.30-9: Echo response frames are capable of traveling as far as their intended destination.

Row 7.30-10: Flush frames are capable of traveling the known size of the ring.

Row 7.30-11: Organization-specific frames are capable of traveling as far as their intended destination.

Row 7.30-12: SAS notify frames are capable of traveling the known size of the ring.

Row 7.30-13: PIRC status frames are capable of traveling the known size of the ring.

Row 7.30-14: Single-choke fairness frames always travel one hop, but start with a *ttl* of MAX_STATIONS whenever the *saCompact* value is reset to the local station. Otherwise, the *ttl* is decremented from the last received *ttl* when the frame is regenerated by the local station.

Row 7.30-15: Multi-choke fairness frames are sent without (accurate) knowledge of the size of the ring and are capable of traveling the largest possible ring.

Row 7.30-16: If the SAS multicast scoping feature is active, and a static multicast entry is found in the *SdbMcastLookup* table, then *ttl* is set to the result of the table lookup. If the entry found specifies 0 hops (i.e., don't send the frame), then the frame will be discarded.

Row 7.30-17: Bidirectionally flooded frames travel exactly as far as the cleave point, such that the following condition is satisfied: $frame0.ttl + frame1.ttl == numStations - 1$.

Row 7.30-18: Unidirectionally flooded frames are capable of traveling the known size of the ring.

Row 7.30-19: Frames that are not flooded travel at least as far as their intended destination.

NOTE—Setting *ttl* to a value larger than needed can interfere with bandwidth reclamation, which impedes full ring bandwidth utilization.

7.7.3.5 Initial *ttlBase* field value

Every frame transmitted from the local client or local MAC control sublayer shall have its *ttlBase* field set to the value of its *ttl* field.

7.7.4 StageQueueSelection state machine

A single-entry stage queue is provided for client and control transmissions without access delay. The stage queue can hold at most one frame at any time. As soon as it empties, the next highest priority frame is selected to be the next add frame to be transmitted.

The stage queue pulls frames from a logical selection queue that allows examination of the frames and selective frame dequeuing. This queue does not provide true FIFO behavior. (Alternatively, it can be thought of a grouping of FIFO queues, with one for each {class of service, frame type} tuple.) The size of the selection queue is dependent upon the implementation and is intended to be just large enough to handle the round-trip time of providing a send indication to the client and receiving back the resultant frame transmission action.

NOTE—This standard does not define any minimum or maximum size for the selection queue. Any such logical or physical structure is recommended to be sufficiently large to allow for full rate transmissions, including the latencies inherent in signaling flow control information across the MAC-to-client interface.

An entire frame is not required to be available if the stage queue is implemented as cut through or has sufficiently small latencies that it can receive the remainder of the frame at full rate. The amount that needs to be available, an entry, is that amount that meets the MAC's needs for subsequent full-rate transmission. The entry contains at least the header.

A frame in the stage queue is considered to have been transmitted, from the MAC client's point of view. This includes any measurements of delay or jitter. Acceptance into the stage queue is the equivalent of acceptance for transmit. Selection of a frame for acceptance into the stage queue, and therefore for transmission, is governed by the per class shapers, the downstream shaper, and the fairness algorithm (as specified in Clause 10).

Acceptance of control frames and client data frames for transmission is done in strict precedence order, limited only by any rate control send indications. The rate controls have the effect of limiting the strict precedence of transmit decisions such that each service class gets its fair share of transmissions.

The following subclauses (7.7.4.1–7.7.4.3) describe parameters used within the context of this state machine.

7.7.4.1 StageQueueSelection state machine definitions

CLASS_A0
CLASS_A1
CLASS_B
CLASS_C
 See 7.2.4.
DATA_MIN
 See 7.2.4.
FT_DATA
 See 7.2.4.
NONE
 See 7.2.1.
NULL
 See 7.2.1.
Q_TX_SS
Q_TX_C_COUNT
Q_TX_D_COUNT
 See 7.2.1.
READY
 See 7.2.4.
SC_CLASS_A
SC_CLASS_B
SC_CLASS_C
 See 7.2.4.

7.7.4.2 StageQueueSelection state machine variables

creditA0
creditA1
creditB

creditD

See 7.2.2.

frame

The contents of an RPR frame.

frameType

The type of frame to limit being chosen.

mark_fe

See 7.2.4.

mtuSize

See 7.2.4.

*passA0**passA1*

See 7.2.2.

*sendA**sendB**sendC**sendD**sendM*

See 7.2.2.

7.7.4.3 StageQueueSelection state machine routines

EntryInQueue(queue)

See 7.2.3.

ExamineQueue(queue, frameType, serviceClass)

See 7.2.3.

ReplaceCredits(frame)

Replaces the credits that had been subtracted from creditA0, creditA1, creditB, or creditC, and for creditD, as applicable for the given frame.

*SelectQueue(queue, frameType, serviceClass)*Looks inside the specified queue within the datapath identified by *myRi* for the next available frame of the specified frame type and the specified service class. If the frame type is specified as NONE, no frame type constraint is applied, but FT_CONTROL is selected in preference to FT_DATA, if both exist. If such a frame is available, it is returned, and removed from the queue.*frame*—The next available frame meeting the specified criteria.

NULL—No frame available meeting the specified criteria.

SizeOfFrame

See 7.2.3.

7.7.4.4 StageQueueSelection state table

The StageQueueSelection state machine specified in Table 7.31 implements the functions necessary for selecting a control frame or a client data frame for admission to the stage queue and the associated frame header processing. Credits are decremented as a frame is processed (see 7.5.1). In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.31—StageQueueSelection state table

Current state		Row	Next state	
state	condition		action	state
START	EntryInQueue(Q_TX_STAGE)	1	—	START
	—	2	—	INIT
INIT	sendM	3	frameType = NONE;	CLASSA
	—	4	frameType = FT_DATA;	
CLASSA	(frame = ExamineQueue(Q_TX_SS, frameType, SC_CLASS_A)) != NULL && passA0	5	frame = SelectQueue(Q_TX_SS, frameType, SC_CLASS_A) frame.fe = 0; frame.sc = CLASS_A0; creditA0 -= SizeOf(frame);	LENGTH
	(frame = ExamineQueue(Q_TX_SS, FT_DATA, SC_CLASS_A)) != NULL && passA1 && sendD	6	frame = SelectQueue(Q_TX_SS, FT_DATA, SC_CLASS_A) frame.fe = 0; frame.sc = CLASS_A1; creditA1 -= SizeOf(frame); creditD -= SizeOf(frame);	
	—	7	—	CLASSB
CLASSB	(frame = ExamineQueue(Q_TX_SS, frameType, SC_CLASS_B)) != NULL && sendC >= frame.ttl && sendD && mark_fe	8	frame = SelectQueue(Q_TX_SS, frameType, SC_CLASS_B) frame.fe = 1; frame.sc = CLASS_B; creditD -= SizeOf(frame);	LENGTH
	(frame = ExamineQueue(Q_TX_SS, frameType, SC_CLASS_B)) != NULL && mark_fe	9	—	CLASSC
	(frame = ExamineQueue(Q_TX_SS, frameType, SC_CLASS_B)) != NULL && sendB && sendD	10	frame = SelectQueue(Q_TX_SS, frameType, SC_CLASS_B) frame.fe = 0; frame.sc = CLASS_B; creditB -= SizeOf(frame); creditD -= SizeOf(frame);	LENGTH
	(frame = ExamineQueue(Q_TX_SS, frameType, SC_CLASS_B)) != NULL && sendC >= frame.ttl && sendD	11	frame = SelectQueue(Q_TX_SS, frameType, SC_CLASS_B) frame.fe = 1; frame.sc = CLASS_B; creditD -= SizeOf(frame);	
	—	12	—	CLASSC
CLASSC	(frame = ExamineQueue(Q_TX_SS, frameType, SC_CLASS_C)) != NULL && sendC >= frame.ttl && sendD	13	frame = SelectQueue(Q_TX_SS, frameType, SC_CLASS_C) frame.fe = 1; frame.sc = CLASS_C; creditD -= SizeOf(frame);	LENGTH
	—	14	—	START

Table 7.31—StageQueueSelection state table (continued)

Current state		Row	Next state	
state	condition		action	state
LENGTH	frame.ft == FT_CONTROL && SizeOf(frame) > CONTROL_MAX	15	ReplaceCredits(frame);	START
	frame.ft == FT_DATA && SizeOf(frame) > mtuSize	16		
	—	17	frame.hec = Crc16(frame);	COUNT
COUNT	frame.ft == FT_DATA	18	Enqueue(Q_TX_D_COUNT, frame);	START
	—	19	creditM -= SizeOf(frame); Enqueue(Q_TX_C_COUNT, frame);	

Row 7.31-1: The stage queue already holds an entry. Wait for the stage queue to become available.

Row 7.31-2: The stage queue is available to accept an entry.

Row 7.31-3: Initialize the frame type constraint to no constraint between FT_CONTROL and FT_DATA.

Row 7.31-4: Constrain the frame type that can be selected to FT_DATA.

Row 7.31-5: ClassA frames within the bounds of the subclassA0 shaper are marked subclassA0 and decrement credits.

Row 7.31-6: ClassA data frames within the bounds of the subclassA1 shaper are marked subclassA1 and decrement credits. (Control frames do not use subclassA1. Control frames are sent at a very low rate and at a fairly consistent rate, so there is no gain in using subclassA1.)

Row 7.31-7: No classA frames to be transmitted at this time.

Row 7.31-8: ClassB frames within the bounds of the fairness eligible shaper and requested to be marked fairness eligible have the *fe* field set and decrement credits.

Row 7.31-9: ClassB frames requested to be marked fairness eligible and with no fairness eligible shaper credits are not accepted for transmission.

Row 7.31-10: ClassB frames within the bounds of the classB shaper have *fe* cleared and decrement credits.

Row 7.31-11: ClassB frames within the bounds of the fairness eligible shaper have *fe* set and decrement credits.

Row 7.31-12: No classB frames to be transmitted at this time.

Row 7.31-13: ClassC frames within the bounds of the fairness eligible shaper have *fe* set and decrement credits.

Row 7.31-14: No classC frames to be transmitted at this time. No frame is added to the stage queue.

Row 7.31-15: Control frames that are too long are not allowed to be transmitted.

Row 7.31-16: Data frames that are too long are not allowed to be transmitted.

Row 7.31-17: The frame is not too long. Finish filling in the header.

Row 7.31-18: The frame transmission statistics are updated for data frames being added to the stage queue.

Row 7.31-19: The frame transmission statistics are updated for control frames being added to the stage queue.

7.7.5 DataAddCount state machine

The DataAddCount state machine supports the accounting necessary for updating data frame add statistics.

The following subclauses (7.7.5.1–7.7.5.3) describe parameters used within the context of this state machine.

7.7.5.1 DataAddCount state machine definitions

CLASS_A0

CLASS_A1

CLASS_B

CLASS_C

See 7.2.4.

NULL

See 7.2.1.

Q_TX_S_COUNT

Q_TX_STAGE

See 7.2.1.

7.7.5.2 DataAddCount state machine variables

frame

The contents of an RPR frame.

fromClientBcastFrames

fromClientMcastFrames

See 7.2.5.

fromClientMcastClassABytes

fromClientMcastClassAFrames

fromClientMcastClassBCirBytes

fromClientMcastClassBCirFrames

fromClientMcastClassBEirBytes

fromClientMcastClassBEirFrames

fromClientMcastClassCBytes

fromClientMcastClassCFrames

See 7.2.5.

fromClientUcastClassABytes

fromClientUcastClassAFrames

fromClientUcastClassBCirBytes

fromClientUcastClassBCirFrames

fromClientUcastClassBEirBytes

fromClientUcastClassBEirFrames

fromClientUcastClassCBytes

fromClientUcastClassCFrames

See 7.2.5.

size

The size, in bytes, of an RPR frame.

7.7.5.3 DataAddCount state machine routines

Broadcast(address)

See 7.2.3.

Dequeue(queue)

Enqueue(queue, frame)

See 7.2.3.

Multicast(address)

See 7.2.3.

SizeOf(frame)

See 7.2.3.

7.7.5.4 DataAddCount state table

The DataAddCount state machine counts added data frames and bytes, as specified in Table 7.32. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.32—DataAddCount state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_TX_D_COUNT)) != NULL	1	size = SizeOf(frame);	UCAST
	—	2	—	START
UCAST	frame.ft != FT_DATA	3	—	STAGE
	Broadcast(frame.da)	4	fromClientBcastFrames += 1;	MCAST
	Multicast(frame.da)	5	fromClientMcastFrames += 1;	
	frame.sc == CLASS_A0	6	fromClientUcastClassAFrames += 1; fromClientUcastClassABytes += size;	STAGE
	frame.sc == CLASS_A1	7	—	
	frame.sc == CLASS_B && frame.fe == 0	8	fromClientUcastClassBCirFrames += 1; fromClientUcastClassBCirBytes += size;	
	frame.sc == CLASS_B && frame.fe == 1	9	fromClientUcastClassBEirFrames += 1; fromClientUcastClassBEirBytes += size;	
	frame.sc == CLASS_C	10	fromClientUcastClassCFrames += 1; fromClientUcastClassCBytes += size;	
MCAST	frame.sc == CLASS_A0	11	fromClientMcastClassAFrames += 1; fromClientMcastClassABytes += size;	STAGE
	frame.sc == CLASS_A1	12	—	
	frame.sc == CLASS_B && frame.fe == 0	13	fromClientMcastClassBCirFrames += 1; fromClientMcastClassBCirBytes += size;	
	frame.sc == CLASS_B && frame.fe == 1	14	fromClientMcastClassBEirFrames += 1; fromClientMcastClassBEirBytes += size;	
	frame.sc == CLASS_C	15	fromClientMcastClassCFrames += 1; fromClientMcastClassCBytes += size;	
STAGE	—	16	Enqueue(Q_TX_STAGE, frame);	START

Row 7.32-1: Fetch the next add frame.

Row 7.32-2: Wait for an add frame.

Row 7.32-3: Non-client frames are not counted.

Row 7.32-4: Broadcast frames have distinct counters.

Row 7.32-5: Multicast frames have distinct counters.

Row 7.32-6: Count the number of unicast classA frames and bytes, subclassA0 contribution.

Row 7.32-7: Count the number of unicast classA frames and bytes, subclassA1 contribution.

Row 7.32-8: Count the number of unicast classB-CIR frames and bytes.

Row 7.32-9: Count the number of unicast classB-EIR frames and bytes.

Row 7.32-10: Count the number of unicast classC frames and bytes.

Row 7.32-11: Count the number of multicast classA frames and bytes, subclassA0 contribution.

Row 7.32-12: Count the number of multicast classA frames and bytes, subclassA0 contribution.

Row 7.32-13: Count the number of multicast classB-CIR frames and bytes.

Row 7.32-14: Count the number of multicast classB-EIR frames and bytes.

Row 7.32-15: Count the number of multicast classC frames and bytes.

Row 7.32-16: Add the frame to the tail of the stage queue.

7.7.6 ControlAddCount state machine

The ControlAddCount state machine supports the accounting necessary for updating control frame add statistics.

The following subclauses (7.7.6.1–7.7.6.3) describe parameters used within the context of this state machine.

7.7.6.1 ControlAddCount state machine definitions

CT_OAM_ECHO_REQ
CT_OAM_ECHO_RSP
CT_OAM_FLUSH
CT_OAM_ORG
CT_OAM_PIRC_STATUS
CT_OAM_SAS_NOTIFY
CT_STATION_ATD
CT_TOPO_CHKSUM
CT_TOPO_PROT
 See 7.2.4.
NULL
 See 7.2.1.
Q_TX_C_COUNT
Q_TX_STAGE
 See 7.2.1.

7.7.6.2 ControlAddCount state machine variables

frame
The contents of an RPR frame.
fromCtrlFrames
See 7.2.5.
fromCtrlFddFrames
fromCtrlLrttReqFrames
fromCtrlLrttRspFrames
fromCtrlOamEchoReqFrames

*fromCtrlOamEchoRspFrames
fromCtrlOamFlushFrames
fromCtrlOamOrgFrames
fromCtrlOamPircStatusFrames
fromCtrlOamSasNotifyFrames
fromCtrlTopoATDFrames
fromCtrlTopoSumFrames
fromCtrlTopoTPFrames*

See 7.2.5.

7.7.6.3 ControlAddCount state machine routines

*Dequeue(queue)
Enqueue(queue, frame)*

See 7.2.3.

7.7.6.4 ControlAddCount state table

The ControlAddCount state machine counts added control frames, as specified in Table 7.33. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.33—ControlAddCount state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_TX_C_COUNT)) != NULL	1	—	COUNT
	—	2	—	START
COUNT	frame.controlType == CT_STATION_ATD	3	fromCtrlTopoATDFrames += 1;	STAGE
	frame.controlType == CT_TOPO_PROT	4	fromCtrlTopoTPFrames += 1;	
	frame.controlType == CT_TOPO_CHKSUM	5	fromCtrlTopoSumFrames += 1;	
	frame.controlType == CT_LRTT_REQ	6	fromCtrlLrttReqFrames += 1;	
	frame.controlType == CT_LRTT_RSP	7	fromCtrlLrttRspFrames += 1;	
	frame.controlType == CT_FDD	8	fromCtrlFddFrames += 1;	
	frame.controlType == CT_OAM_ECHO_REQ	9	fromCtrlOamEchoReqFrames += 1;	
	frame.controlType == CT_OAM_ECHO_RSP	10	fromCtrlOamEchoRspFrames += 1;	
	frame.controlType == CT_OAM_FLUSH	11	fromCtrlOamFlushFrames += 1;	
	frame.controlType == CT_OAM_ORG	12	fromCtrlOamOrgFrames += 1;	
	frame.controlType == CT_OAM_SAS_NOTIFY	13	fromCtrlOamSasNotifyFrames += 1;	
	frame.controlType == CT_OAM_PIRC_STATUS	14	fromCtrlOamPircStatusFrames += 1;	
	—	15	—	

Table 7.33—ControlAddCount state table (continued)

Current state		Row	Next state	
state	condition		action	state
STAGE	—	16	fromCtrlFrames += 1; Enqueue(Q_TX_STAGE, frame);	START

Row 7.33-1: Fetch the next add frame.

Row 7.33-2: Wait for an add frame.

Row 7.33-3: Count the number of station attribute discovery frames.

Row 7.33-4: Count the number of topology and protection frames.

Row 7.33-5: Count the number of topology checksum frames.

Row 7.33-6: Count the number of loop round-trip time request frames.

Row 7.33-7: Count the number of loop round-trip time response frames.

Row 7.33-8: Count the number of fairness differential delay frames.

Row 7.33-9: Count the number of OAM echo request frames.

Row 7.33-10: Count the number of OAM echo response frames.

Row 7.33-11: Count the number of OAM flush frames.

Row 7.33-12: Count the number of OAM organization-specific frames.

Row 7.33-13: Count the number of SAS notify frames.

Row 7.33-14: Count the number of PIRC status frames.

Row 7.33-15: Do not count any other frames.

Row 7.33-16: Count all frames and add them to the tail of the stage queue.

7.7.7 Single queue MAC design

A single-queue MAC uses one queue, the primary transit queue (PTQ), for all transit traffic. This standard does not define sizing for the PTQ.

7.7.7.1 Single queue MAC datapaths

To be able to detect when to transmit and receive frames from the ring, a single queue MAC makes use of only one transit queue per ringlet, as illustrated in Figure 7.15. The PTQ has the behavior of a small FIFO.

Figure 7.15 is interpreted as explained for Figure 7.3, with the exception that all classA traffic is rate-limited by only the subclassA0 shaper because single queue (PTQ only) implementations cannot support subclassA1 add traffic.

A FIFO ordering shall be maintained when entries pass through the PTQ.

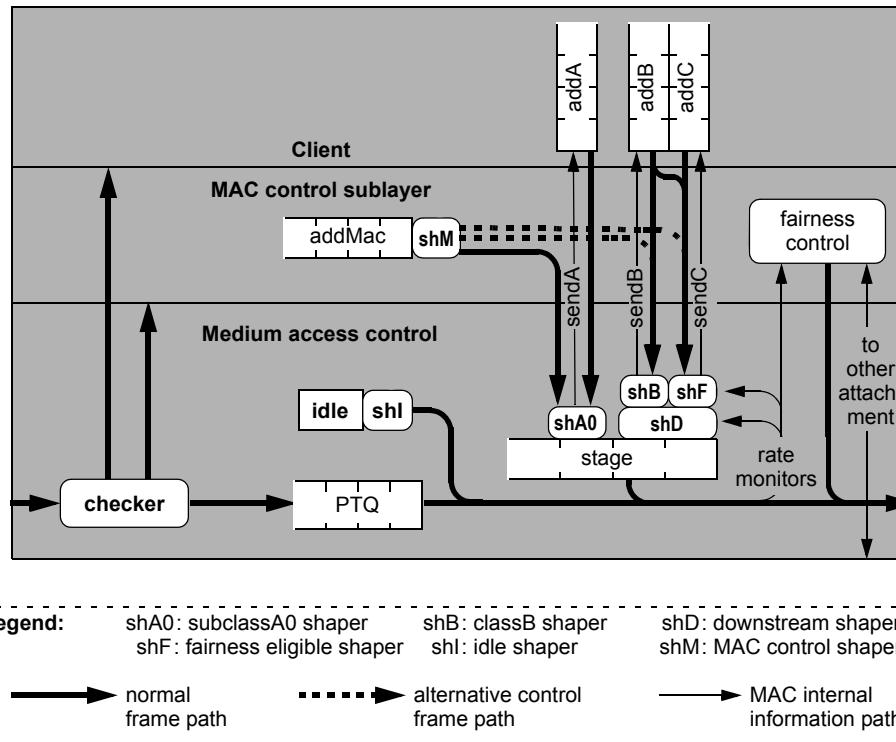


Figure 7.15—Single queue MAC datapath

7.7.7.2 SingleQueueTransmit state machine

The end result of the SingleQueueTransmit state machine is that a frame is transmitted via an invocation of `PHY_DATA.request`. When the MAC is required to provide the length specifier to the transmit PHY, where and how that length is computed is implementation dependent.

NOTE—For PHY layers that require length information prepended to the frame, the calculation of that length can entail an additional store-and-forward delay implemented inside or outside the MAC.

The following subclauses (7.7.7.2.1–7.7.7.2.3) describe parameters used within the context of this state machine.

7.7.7.2.1 SingleQueueTransmit state machine definitions

CLASS_A0

See 7.2.4.

NULL

See 7.2.1.

Q_TX_IDLE

Q_TX_MAC

Q_TX_PHY

Q_TX_PTQ

Q_TX_STAGE

See 7.2.1.

READY

See 7.2.4.

7.7.7.2.2 SingleQueueTransmit state machine variables*creditI**creditD*

See 7.2.2.

frame

The contents of an RPR frame.

fromCtrlFairnessFrames

See 7.2.5.

fromCtrlFrames

See 7.2.5.

7.7.7.2.3 SingleQueueTransmit state machine routines*Dequeue(queue)**Enqueue(queue, frame)*

See 7.2.3.

PHY_READY.indication()

See 7.2.4.

SizeOf(frame)

See 7.2.3.

7.7.7.2.4 SingleQueueTransmit state table

The SingleQueueTransmit state machine specified in Table 7.34 implements the functions necessary for selecting which frame to transmit in a single queue MAC. The intent is to always empty the primary transit queue (PTQ) before client frame transmissions. Credits are decremented as a frame is processed (see 7.5.1). In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.34—SingleQueueTransmit state table

Current state		Row	Next state	
state	condition		action	state
START	PHY_READY.indication() == READY	1	—	IDLE
	—	2	—	START
IDLE	(frame = Dequeue(Q_TX_IDLE)) != NULL	3	creditI -= SizeOf(frame); Enqueue(Q_TX_COUNT, frame);	START
	—	4	—	FAIRNESS
FAIRNESS	(frame = Dequeue(Q_TX_FAIR)) != NULL	5	fromCtrlFrames += 1; fromCtrlFairnessFrames += 1; Enqueue(Q_TX_COUNT, frame);	START
	—	6	—	PTQ
PTQ	(frame = Dequeue(Q_TX_PTQ)) != NULL	7	—	COUNT
	—	8	—	STAGE

Table 7.34—SingleQueueTransmit state table (continued)

Current state		Row	Next state	
state	condition		action	state
COUNT	frame.sc != CLASS_A0	9	creditD -= SizeOf(frame); Enqueue(Q_TX_COUNT, frame);	START
	—	10	Enqueue(Q_TX_COUNT, frame);	
STAGE	(frame = Dequeue(Q_TX_STAGE)) != NULL	11	Enqueue(Q_TX_COUNT, frame);	START
	—	12	—	

Row 7.34-1: The physical layer has indicated that it is ready.

Row 7.34-2: Wait for the physical layer to be ready.

NOTE—The exact point within transmit processing at which the check of the PHY_READY indication is done is implementation specific.

Row 7.34-3: MAC idle transmissions have precedence over all other transmissions.

Row 7.34-4: No idle frame ready. Check for fairness frames.

Row 7.34-5: MAC fairness frame transmissions have precedence over all control and data transmissions.

Row 7.34-6: No fairness frame ready. Check the PTQ.

Row 7.34-7: The PTQ is selected when an entry (at least a complete header for cut-through, or a complete frame for store-and-forward) is available in the queue, with precedence over client and control transmissions.

Row 7.34-8: No frame is in the PTQ. Check the stage queue.

Row 7.34-9: Transit frames using unreserved bandwidth subtract from the downstream shaper credits.

Row 7.34-10: Transit frames using reserved bandwidth do not subtract from any shaper credits.

Row 7.34-11: The stage queue is selected when a (data frame or control frame) entry is available in the queue.

Row 7.34-12: No frame is selected when no frame is available.

NOTE—The stage queue is conceptual only, and at no time is selection of frames into the stage queue relevant except when transmission would occur immediately. Therefore, no head-of-line blocking of the stage queue occurs.

7.7.8 Dual queue MAC design

A dual queue MAC uses two transit queues, the primary transit queue (PTQ) for classA traffic, and the secondary transit queue (STQ) for classB and classC traffic. The size of the both transit queues is left to the implementations. The size of the secondary transit queue determines its flow-control threshold values. The dual queue design is described in 7.7.8.1 and 7.7.8.2.

7.7.8.1 Dual queue MAC datapaths

A dual queue MAC makes use of two transit queues, as illustrated in Figure 7.16.

Figure 7.16 is interpreted as explained for Figure 7.3, with the use of both the PTQ and the STQ in the transit path.

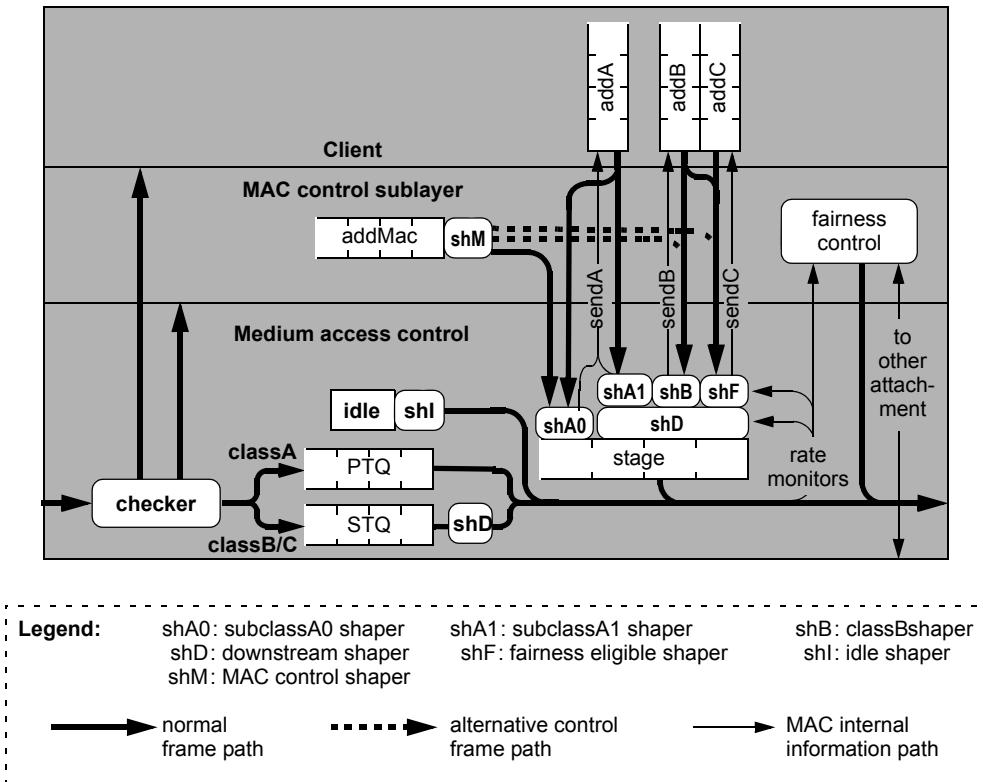


Figure 7.16—Dual queue MAC datapath

The following externally visible behaviors shall be supported:

- PTQ ordering. A FIFO ordering shall be maintained when entries pass through the PTQ.
- STQ ordering. A FIFO ordering shall be maintained when entries pass through the STQ.
- Cross ordering. An entry from the STQ shall not be output before a previously received PTQ entry.

7.7.8.2 DualQueueTransmit state machine

The transmit behavior of a dual queue MAC is described by its transmit-selection protocols (specified in this subclause) and shaping functions. The transmit-selection protocol and shaping functions are independent for *sendA*, *sendB*, and *sendC*. But coupling of *sendM* and the associated service class specific send indications is required to ensure conformance of control frame transmissions.

The end result of the DualQueueTransmit state machine is that a frame is transmitted via an invocation of `PHY_DATA.request`. When the MAC is required to provide the length specifier to the transmit PHY, where and how that length is computed is implementation dependent.

NOTE—For PHY layers that require length information prepended to the frame, the calculation of that length can entail an additional store-and-forward delay implemented inside or outside the MAC.

The following subclauses (7.7.8.2.1–7.7.8.2.3) describe parameters used within the context of this state machine.

7.7.8.2.1 DualQueueTransmit state machine definitions

CLASS_A0
 See 7.2.4.
 NULL
 See 7.2.1.
 Q_TX_IDLE
 Q_TX_MAC
 Q_TX_PHY
 Q_TX_PTQ
 Q_TX_STQ
 Q_TX_STAGE
 See 7.2.1.
 READY
 See 7.2.4.

7.7.8.2.2 DualQueueTransmit state machine variables

creditI
creditD
 See 7.2.2.
frame
 The contents of an RPR frame.
fromCtrlFairnessFrames
 See 7.2.5.
fromCtrlFrames
 See 7.2.5.
mtuSize
 See 7.2.4.
sendD
 See 7.2.2.
stqDepth
 See 7.2.2.
stqFullThreshold
 See 7.2.2.

7.7.8.2.3 DualQueueTransmit state machine routines

Dequeue(queue)
Enqueue(queue, frame)
 See 7.2.3.
PHY_READY.indication()
 See 7.2.4.
SizeOfframe
 See 7.2.3.

7.7.8.2.4 DualQueueTransmit state table

The DualQueueTransmit state machine specified in Table 7.35 implements the functions necessary for selecting which frame to transmit in a dual queue MAC. The intent is to always empty the primary transit queue (PTQ) before client frame transmissions, but to allow the secondary transit queue (STQ) to fill somewhat while adding client frame transmissions. Credits are decremented as a frame is processed (see 7.5.1). In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.35—DualQueueTransmit state table

Current state		Row	Next state	
state	condition		action	state
START	PHY_READY.indication() == READY	1	—	IDLE
	—	2	—	START
IDLE	(frame = Dequeue(Q_TX_IDLE)) != NULL	3	creditI -= SizeOf(frame); Enqueue(Q_TX_COUNT, frame);	START
	—	4	—	FAIRNESS
FAIRNESS	(frame = Dequeue(Q_TX_FAIR)) != NULL	5	fromCtrlFrames += 1; fromCtrlFairnessFrames += 1; Enqueue(Q_TX_COUNT, frame);	START
	—	6	—	PTQ
PTQ	(frame = Dequeue(Q_TX_PTQ)) != NULL	7	—	COUNT
	—	8	—	FULL
COUNT	frame.sc != CLASS_A0	9	creditD -= SizeOf(frame); Enqueue(Q_TX_COUNT, frame);	START
	—	10	Enqueue(Q_TX_COUNT, frame);	
FULL	stqDepth >= stqFullThreshold	11	frame = Dequeue(Q_TX_STQ); creditD -= SizeOf(frame); Enqueue(Q_TX_COUNT, frame);	START
	—	12	—	STAGE
STAGE	(frame = Dequeue(Q_TX_STAGE)) != NULL	13	Enqueue(Q_TX_COUNT, frame);	START
	—	14	—	STQ
STQ	sendD && (frame = Dequeue(Q_TX_STQ)) != NULL	15	creditD -= SizeOf(frame); Enqueue(Q_TX_COUNT, frame);	START
	—	16	—	

Row 7.35-1: The physical layer has indicated that it is ready.

Row 7.35-2: Wait for the physical layer to be ready.

NOTE—The exact point within transmit processing at which the check of the PHY_READY.indication is done is implementation specific.

Row 7.35-3: MAC idle transmissions have precedence over all other transmissions.

Row 7.35-4: No idle frame ready. Check for fairness frames.

Row 7.35-5: MAC fairness frame transmissions have precedence over all control and data transmissions.

Row 7.35-6: No fairness frame ready. Check the PTQ.

Row 7.35-7: The primary transit queue is selected when an entry (a complete header for cut-through, or a complete frame for store-and-forward) is available in the queue, with precedence over client transmissions.

Row 7.35-8: In the absence of primary transit queue frames, other transmission sources are checked.

Row 7.35-9: Transit frames using unreserved bandwidth subtract from the downstream shaper credits.

Row 7.35-10: Transit frames using reserved bandwidth do not subtract from any shaper credits.

Row 7.35-11: The secondary transit queue is selected when it gets too close to being full, to avoid overflows.

Row 7.35-12: In the absence of secondary transit queue overflow threats, other transmission sources are checked.

Row 7.35-13: The stage queue (data frame or control frame) entry preempts STQ transmissions, because provision checks were done previously.

Row 7.35-14: In the absence of stage queue frames, other transmission sources are checked.

Row 7.35-15: The secondary transit queue is selected when there are downstream credits and an entry is available.

Row 7.35-16: No frame is selected when no frame is available.

NOTE—The stage queue is conceptual only, and at no time is selection of frames into the stage queue relevant except when transmission would occur immediately. Therefore no head-of-line blocking of the stage queue occurs.

7.7.9 TransmitCount state machine

The TransmitCount state machine implements the functions necessary for counting transmitted bytes and frames.

The following subclauses (7.7.9.1–7.7.9.3) describe parameters used within the context of this state machine.

7.7.9.1 TransmitCount state machine definitions

- CLASS_A0
- CLASS_A1
- CLASS_B
- CLASS_C
 - See 7.2.4.
- FT_DATA
 - See 7.2.4.
- NULL
 - See 7.2.1.
- PASSTHROUGH
 - See 7.2.2.
- Q_TX_COUNT
- Q_TX_ROUTE
 - See 7.2.1.

7.7.9.2 TransmitCount state machine variables

frame

The contents of an RPR frame.

size

The size, in bytes, of an RPR frame.

txMcastClassABytes
txMcastClassAFrames
txMcastClassBCirBytes
txMcastClassBCirFrames
txMcastClassBEirBytes
txMcastClassBEirFrames
txMcastClassCBytes
txMcastClassCFrames
 See 7.2.5.

txUcastClassABytes
txUcastClassAFrames
txUcastClassBCirBytes
txUcastClassBCirFrames
txUcastClassBEirBytes
txUcastClassBEirFrames
txUcastClassCBytes
txUcastClassCFrames
 See 7.2.5.

7.7.9.3 TransmitCount state machine routines

Dequeue(queue)
 See 7.2.3.

Multicast(address)
 See 7.2.3.

Enqueue(queue,frame)
 See 7.2.3.

SizeOf(frame)
 See 7.2.3.

7.7.9.4 TransmitCount state table

The TransmitCount state machine specified in Table 7.36 implements the functions necessary for updating flow-rate MIB-attribute statistics. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.36—TransmitCount state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_TX_COUNT)) != NULL	1	size = SizeOf(frame);	UCAST
	—	2	—	START

Table 7.36—TransmitCount state table (continued)

Current state		Row	Next state	
state	condition		action	state
UCAST	myEdgeState == PASSTHROUGH	3	—	FINAL
	frame.ft != FT_DATA	4	—	
	Multicast(frame.da)	5	—	
	frame.sc == CLASS_A0	6	txUcastClassAFrames += 1; txUcastClassABytes += size;	FINAL
	frame.sc == CLASS_A1	7	txUcastClassBCirFrames += 1; txUcastClassBCirBytes += size;	
	frame.sc == CLASS_B && frame.fe == 0	8	txUcastClassBEirFrames += 1; txUcastClassBEirBytes += size;	
	frame.sc == CLASS_B && frame.fe == 1	9	txUcastClassBEirFrames += 1; txUcastClassBEirBytes += size;	
	frame.sc == CLASS_C	10	txUcastClassCFrames += 1; txUcastClassCBytes += size;	
MCAST	frame.sc == CLASS_A0	11	txMcastClassAFrames += 1; txMcastClassABytes += size;	FINAL
	frame.sc == CLASS_A1	12	txMcastClassABytes += size;	
	frame.sc == CLASS_B && frame.fe == 0	13	txMcastClassBCirFrames += 1; txMcastClassBCirBytes += size;	
	frame.sc == CLASS_B && frame.fe == 1	14	txMcastClassBEirFrames += 1; txMcastClassBEirBytes += size;	
	frame.sc == CLASS_C	15	txMcastClassCBytes += size;	
FINAL	—	16	Enqueue(Q_TX_ROUTE, frame);	START

Row 7.36-1: Fetch the next output frame.**Row 7.36-2:** Retry fetching an output frame.**Row 7.36-3:** Counters are not updated when operating in passthrough mode.**Row 7.36-4:** Only data frames are counted.**Row 7.36-5:** Multicast frames have distinct counters.**Row 7.36-6:** Count the number of unicast classA frames and bytes, subclassA0 contribution.**Row 7.36-7:** Count the number of unicast classA frames and bytes, subclassA1 contribution.**Row 7.36-8:** Count the number of unicast classB-CIR frames and bytes.**Row 7.36-9:** Count the number of unicast classB-EIR frames and bytes.**Row 7.36-10:** Count the number of unicast classC frames and bytes transmitted.**Row 7.36-11:** Count the number of multicast classA frames and bytes, subclassA0 contribution.**Row 7.36-12:** Count the number of multicast classA frames and bytes, subclassA1 contribution.**Row 7.36-13:** Count the number of multicast classB-CIR frames and bytes transmitted.**Row 7.36-14:** Count the number of multicast classB-EIR frames and bytes transmitted.**Row 7.36-15:** Count the number of multicast classC frames and bytes transmitted.**Row 7.36-16:** Pass the frame to the transmit routing queue.

7.7.10 TransmitRoute state machine

The TransmitRoute state machine provides the final transmission step that hands the frame to the PHY or wraps it to the other ringlet's receive path.

The following subclauses (7.7.10.1–7.7.10.3) describe parameters used within the context of this state machine.

7.7.10.1 TransmitRoute state machine definitions

CENTER_WRAP

See 7.2.2.

EDGE_WRAP

See 7.2.2.

FT_FAIRNESS

See 7.2.4.

INTO_EDGE

See 7.2.2.

NULL

See 7.2.1.

Q_RX_CHECK

Q_TX_ROUTE

Q_TX_PHY

See 7.2.1.

SINGLE_CHOKE

See 7.2.4.

STEERING

See 7.2.4.

7.7.10.2 TransmitRoute state machine variables

frame

The contents of an RPR frame.

myEdgeState

See 7.2.2.

myProtectMethod

See 7.2.2.

protConfig

See 7.2.4.

7.7.10.3 TransmitRoute state machine routines

Dequeue(queue)

See 7.2.3.

EdgeAllowedFrame(frame)

See 7.2.3.

Replicate(frame)

See 7.2.3.

EnqueueRi(ringlet, queue, frame)

See 7.2.3.

7.7.10.4 TransmitRoute state table

The TransmitRoute state machine specified in Table 7.37 implements the functions necessary for transmitting the frame based on the protection state of span into which this datapath is transmitting. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 7.37—TransmitRoute state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_TX_ROUTE)) == NULL	1	—	START
	myEdgeState == INTO_EDGE && (frame.we == 0 protConfig == STEERING) && !EdgeAllowedFrame(frame)	2	—	
	myEdgeState == INTO_EDGE && myProtectMethod == CENTER_WRAP && !EdgeAllowedFrame(frame)	3	EnqueueRi(Other(myRi), Q_TX_PHY, frame);	
	myEdgeState == INTO_EDGE && myProtectMethod == EDGE_WRAP && frame.ft == FT_FAIRNESS && frame.ffType == SINGLE_CHOKE	4	Replicate(frame); EnqueueRi(myRi, Q_TX_PHY, frame0); EnqueueRi(Other(myRi), Q_RX_CHECK, frame1);	
	myEdgeState == INTO_EDGE && myProtectMethod == EDGE_WRAP && !EdgeAllowedFrame(frame)	5	EnqueueRi(Other(myRi), Q_RX_CHECK, frame);	
	—	6	EnqueueRi(myRi, Q_TX_PHY, frame);	

Row 7.37-1: Keep trying to fetch a next output frame until successful.

Row 7.37-2: Do not send frames into an edge that are not going to be wrapped and are not allowed to be transmitted across an edge.

Row 7.37-3: Center wrap the frame by passing it to the reconciliation sublayer for the PHY on which the opposing datapath is transmitting.

Row 7.37-4: A single-choke fairness frame being transmitted by this station needs to be both edge wrapped (to the ReceiveCheck state machine on the opposing datapath) and passed to the PHY on which this datapath is transmitting.

Row 7.37-5: Edge wrap the frame (not including a topology and protection frame being transmitted by this station) by passing it to the ReceiveCheck state machine on the opposing datapath.

Row 7.37-6: Pass the frame to the reconciliation sublayer for the PHY on which this datapath is transmitting.

7.8 Protocol Implementation Conformance Statement (PICS) proforma for Clause 7¹⁹

7.8.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 7, Medium access control datapath, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

7.8.2 Identification

7.8.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

7.8.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, Medium access control datapath
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

¹⁹*Copyright release for PICS proforms:* Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

7.8.3 PICS tables for Clause 7

7.8.3.1 State machines

Item	Feature	Subclause	Value/Comment	Status	Support
SM1	IdleShaper	7.5.3	IdleShaper state machine is supported	O	Yes [] No []
SM2	MacControlShaper	7.5.4	MacControlShaper state machine is supported	M	Yes []
SM3	ClassAShaper	7.5.5	ClassAShaper state machine is supported	M	Yes []
SM4	ClassBShaper	7.5.6	ClassBShaper state machine is supported	M	Yes []
SM5	fairness eligible shaper	7.5.7	fairness eligible shaper state machines are supported	M	Yes []
SM6	DownstreamShaper	7.5.8	DownstreamShaper state machine is supported	M	Yes []
SM7	ReceiveFromEdge	7.6.3.6	ReceiveFromEdge state machine is supported	M	Yes []
SM8	ReceiveCheck	7.6.3.7	ReceiveCheck state machine is supported	M	Yes []
SM9	ReceiveCount	7.6.3.8	ReceiveCount state machine is supported	M	Yes []
SM10	ReceiveStrip	7.6.3.9	ReceiveStrip state machine is supported	M	Yes []
SM11	ReceiveAdjust	7.6.3.10	ReceiveAdjust state machine is supported	M	Yes []
SM12	ReceiveFilter	7.6.3.11	ReceiveFilter state machine is supported	M	Yes []
SM13	ReceiveFilterDataCount	7.6.3.12	ReceiveFilterDataCount state machine is supported	M	Yes []
SM14	ReceiveFilterControlCount	7.6.3.13	ReceiveFilterControlCount state machine is supported	M	Yes []
SM15	WrongRinglet	7.6.4	WrongRinglet state machine is supported	PM2:M	Yes []
SM16	RingletSelection	7.7.1	RingletSelection state machine is supported	M	Yes []
SM17	StageQueueSelection	7.7.4	StageQueueSelection state machine is supported	M	Yes []
SM18	DataAddCount	7.7.5	DataAddCount state machine is supported	M	Yes []

Item	Feature	Subclause	Value/Comment	Status	Support
SM19	ControlAddCount	7.7.6	ControlAddCount state machine is supported	M	Yes []
SM20a*	SingleQueueTransmit	7.7.7	SingleQueueTransmit state machine is supported	O.1	Yes [] No []
SM20b*	DualQueueTransmit	7.7.8	DualQueueTransmit state machine is supported	O.1	Yes [] No []
SM21	TransmitCount	7.7.9	TransmitCount state machine is supported	M	Yes []
SM22	TransmitRoute	7.7.10	TransmitRoute state machine is supported	M	Yes []

7.8.3.2 Datapaths

Item	Feature	Subclause	Value/Comment	Status	Support
DP1	Transit delay	7.4	Transit path delay is less than the maximum of 10 microseconds or 4 MTUs	M	Yes []

7.8.3.3 Datapath protection mechanisms

Item	Feature	Subclause	Value/Comment	Status	Support
PM1	steering	7.4.4	Station can be configured for steering protection	M	Yes []
PM2*	wrapping	7.4.4	Station can be configured for wrapping protection	O	Yes [] No[]
PM2a	center wrapping	7.4.4	Center wrapping implemented	PM2:O.2	Yes [] No[]
PM2b	edge wrapping	7.4.4	Edge wrapping implemented	PM2:O.2	Yes [] No[]
PM3	passthrough	7.4.3	Passthrough is supported and obeys specified behaviors	O	Yes [] No[]

7.8.3.4 Wrapping

Item	Feature	Subclause	Value/Comment	Status	Support
W1	wrapped station	7.4.4	To wrap, the local station shall be wrapped	PM2:M	Yes []
W2	frame fields for wrapping	7.6.2	To wrap, the frame shall have fields set as described	PM2:M	Yes []
W3	frame fields for unwrapping	7.6.2	To unwrap, the frame shall have fields set as described	PM2:M	Yes []

7.8.3.5 Shapers

Item	Feature	Subclause	Value/Comment	Status	Support
SH1	Crossing below low limit	7.5	Crossing below the low limit causes the send indication to be withdrawn	M	Yes []
SH2	high limit value	7.5	The high limit value is at least 1 MTU	M	Yes []
SH3	decrement frequency	7.5.1	Shapers are decremented at least every 256 bytes and at the end of each frame	M	Yes []
SH4	accuracy	7.5.1	Shaper implementations yield credits within 5% of conceptual calculations	M	Yes []

7.8.3.6 Context containment

Item	Feature	Subclause	Value/Comment	Status	Support
CC1	Steering purge	7.6.1	Steering configured systems discard strict frames after a context switch	M	Yes []
CC2	Queue purge	7.6.1.1	Strict frames are discarded from the STQ and Q_TX_SS upon triggering of context containment	M	Yes []
CC3	Wrapping purge	7.6.1, 7.6.1.4	Wrapping configured systems discard strict frames after unwrapping	PM2:M	Yes []
CC4	Wrapping field requirements	7.6.2	Fields set correctly to wrap	PM2:M	Yes []
CC5	Unwrapping field requirements	7.6.2	Fields set correctly to unwrap	PM2:M	Yes []

7.8.3.7 *ttl* setting

Item	Feature	Subclause	Value/Comment	Status	Support
TTL1	<i>ttl</i> setting	7.7.2.1	The <i>ttl</i> value of a frame is set within the specified constraints	M	Yes []
TTL2	<i>ttlBase</i> setting	7.7.2.1	The <i>ttlBase</i> value of a frame is set to the <i>ttl</i> value	M	Yes []
TTL3	<i>stdCleave</i> setting	7.7.2.1	Standard cleave point setting is supported	O	Yes [] No []

7.8.3.8 Frame ordering

Item	Feature	Subclause	Value/Comment	Status	Support
FO1	single queue	7.7.7.1	FIFO ordering is maintained on the PTQ	SM20a: M	Yes []
FO2	dual queue PTQ	7.7.8.1	FIFO ordering is maintained on the PTQ	SM20b: M	Yes []
FO3	dual queue STQ	7.7.8.1	FIFO ordering is maintained on the STQ	SM20b: M	Yes []
FO4	dual queue cross ordering	7.7.8.1	STQ entries do not pass PTQ entries	SM20b: M	Yes []

8. MAC physical interface

8.1 Overview

This clause specifies the physical layer interfaces (PHYS) for use with the MAC procedures defined by this standard. It describes two families of optional reconciliation sublayers for use with PacketPHYs (a family of packet-oriented physical layers defined in this standard) and SONET/SDH PHYs. The reconciliation sublayers map the MAC physical layer service primitives to standard electrical interfaces used by these PHYs. To ensure interoperability, an interface that is compliant to this standard shall implement at least one of the PHYs defined by this clause.

NOTE—Systems implementing this standard may include interfaces with other PHYs. This is outside of the scope of this standard.

PacketPHYs are 1 Gb/s and 10 Gb/s PHYs, similar to those defined by IEEE Std 802.3-2008. Reconciliation sublayers defined in this clause enable RPR MACs to use PacketPHYs. These reconciliation sublayers do not allow PacketPHYs to operate with Ethernet MACs.

The SONET/SDH reconciliation sublayers provide interfaces to adaptation sublayers that specify either frame-mapped generic framing procedure (GFP), byte-synchronous high-level data link control (HDLC)-like framing, or link access procedure-SDH (LAPS) framing for SONET/SDH networks and PHYs operating at 155 Mb/s to 10 Gb/s or higher.

Implementations other than those described in this standard or the use of PHYs or interfaces not described in this standard may provide interoperable implementations but are beyond the scope of this standard.

8.1.1 Objectives

The objectives of the MAC physical interface are as follows:

- a) Support physical layer independence.
- b) Support the operation of RPR with data rates ranging from 155 Mb/s to 10 Gb/s.
- c) Provide full-duplex media between adjacent RPR stations on the ring.
- d) Allow the RPR MAC to operate over both synchronous and asynchronous physical layers.
- e) Provide PHY interfaces that have no adverse effect on the RPR protection mechanism.

8.1.2 Relationship to other standards

Figure 8.1 shows the relationship among the RPR MAC, reconciliation sublayers, adaptation sublayers, and physical layers.

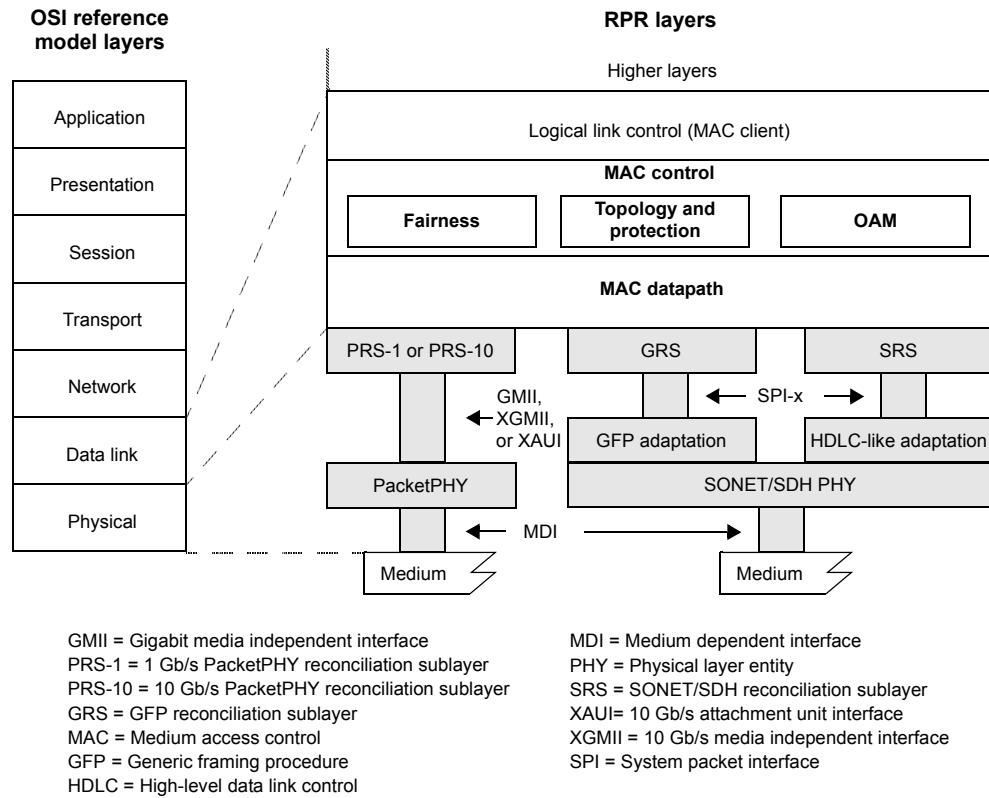


Figure 8.1—RPR RS and PHY relationship to the ISO/IEC OSI reference model

8.2 MAC physical layer service interface

The MAC physical layer service interface allows the MAC to transfer information to and from the reconciliation sublayer and PHYs through logical service primitives. The following primitives are defined:

- **PHY_DATA.request**
- **PHY_DATA.indication**
- **PHY_LINK_STATUS.indication**
- **PHY_READY.indication**

8.2.1 **PHY_DATA.request**

8.2.1.1 Function

This primitive defines the transfer of data from the MAC to the reconciliation sublayer.

8.2.1.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
PHY_DATA.request
(
    OUTPUT_FRAME,
    LENGTH
)
```

The parameters of the PHY_DATA.request are described below:

OUTPUT_FRAME

An RPR frame transferred from the MAC to the reconciliation sublayer.

LENGTH

Used by some reconciliation sublayers to indicate the length of the OUTPUT_FRAME parameter.

(non-null)—Indicates a valid length for the OUTPUT_FRAME parameter.

(null)—No valid length information is supplied.

8.2.1.3 When generated

This primitive is generated by the MAC sublayer to request the transmission of a frame on the physical medium.

8.2.1.4 Effect of receipt

The effect of receipt of this primitive and a detailed mapping of this primitive to specific electrical interfaces are described in Annex B and Annex C.

8.2.2 PHY_DATA.indication

8.2.2.1 Function

This primitive defines the transfer of data from the reconciliation sublayer to the MAC.

8.2.2.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
PHY_DATA.indication
(
    INPUT_FRAME,
    STATUS,
    LENGTH
)
```

The parameters of the PHY_DATA.indication are described below:

INPUT_FRAME

An RPR frame transferred from the reconciliation sublayer to the MAC.

STATUS

An indication of whether the frame was received in error.

OK—No error signaled during the received frame transmission.

ERROR—(Otherwise).

(null)—A null parameter indicates not used.

LENGTH

Indicate the INPUT_FRAME parameter length

(non-null)—Indicates a valid length for the INPUT_FRAME parameter.

(null)—No valid length information is supplied.

8.2.2.3 When generated

The definition of when this primitive is generated and a detailed mapping of specific electrical interfaces to this primitive are described in Annex B and Annex C.

8.2.2.4 Effect of receipt

The effect of receipt of this primitive by the MAC sublayer is specified in 7.6.3.

8.2.3 PHY_LINK_STATUS.indication

8.2.3.1 Function

This primitive conveys the status of the physical link as detected by the PHY to the MAC.

8.2.3.2 Semantics of the service primitive

The semantics of the primitives are as follows:

PHY_LINK_STATUS.indication

(

 LINK_STATUS

)

The parameters of the PHY_LINK_STATUS.indication are described below.

LINK_STATUS

A parameter that indicates the status of the link as detected by the PHY:

NO_DEFECT—No link degradation or fault condition has been sensed.

SIGNAL_FAIL—A link fault condition has been detected.

SIGNAL_DEGRADE—A link degradation condition has been detected.

DEGRADE_FAIL—Both the degrade and fail conditions have been detected.

Not all reconciliation sublayers support SIGNAL_DEGRADE. SONET/SDH reconciliation sublayers support SIGNAL_DEGRADE, and PacketPHY reconciliation sublayers may not support SIGNAL_DEGRADE.

8.2.3.3 When generated

The PHY_LINK_STATUS.indication service primitive shall be generated by the reconciliation sublayer whenever the value of the LINK_STATUS parameter changes.

Additional requirements for this primitive and a detailed mapping of specific electrical interfaces to this primitive are described in Annex B and Annex C.

8.2.3.4 Effect of receipt

The effect of receipt of this primitive by the MAC sublayer is specified in Clause 11.

8.2.4 Mapping of PHY_READY.indication

8.2.4.1 Function

This primitive indicates whether the reconciliation sublayer is ready to accept a new MAC frame.

8.2.4.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
PHY_READY.indication
(
    READY_STATUS
)
```

The parameters of the PHY_READY.indication are described below.

READY_STATUS

An indication of whether the reconciliation sublayer is able to accept a frame from the MAC.

READY—The reconciliation sublayer is able to accept a frame.

NOT_READY—The reconciliation sublayer is unable to accept a frame

8.2.4.3 When generated

The definition of when this primitive is generated and a detailed mapping of specific electrical interfaces to this primitive are described in Annex B and Annex C.

8.2.4.4 Effect of receipt

The effect of receipt of this primitive by the MAC sublayer is specified in 7.7.7.2 and 7.7.8.2.

8.3 PacketPHY physical layer interfaces and PHYs

8.3.1 PacketPHY reconciliation sublayers

Two reconciliation sublayers are defined to provide interfaces to PacketPHYS. The first is the 1 Gb/s PacketPHY reconciliation sublayer (PRS-1) that provides a standard interface for use with 1 Gb/s PacketPHYS. The second is the 10 Gb/s PacketPHY reconciliation sublayer (PRS-10) that provides a standard interface for use with 10 Gb/s PacketPHYS.

8.3.1.1 1 Gb/s PacketPHY reconciliation sublayer (PRS-1)

The 1 Gb/s PacketPHY reconciliation sublayer (PRS-1) maps the MAC physical layer service primitives to and from the gigabit media independent interface (GMII) specified in Clause 35 of IEEE Std 802.3-2008. The GMII is an optional interface, but it is used as a basis for the description of the PRS-1. An implementation that does not provide the GMII shall behave functionally as if the interface was implemented. The requirements for the PRS-1 are specified in Annex B.

8.3.1.2 10 Gb/s PacketPHY reconciliation sublayer (PRS-10)

The 10 Gb/s PacketPHY reconciliation sublayer (PRS-10) maps the MAC physical layer service primitives to and from the 10 Gb media independent interface (XGMII) specified in Clause 46 of IEEE Std 802.3-2008. Alternatively, an XGMII extender sublayer (XGXS) may be used to provide a 10 Gb attachment unit interface (XAUI) as defined in Clause 47 of IEEE Std 802.3-2008. The XGMII and XAUI are optional interfaces. The XGMII is used as a basis for the description of the PRS-10. An implementation that does not provide the XGMII or XAUI shall behave functionally as if the interfaces were implemented. The requirements for the PRS-10 are specified in Annex B.

8.3.2 PacketPHYS

8.3.2.1 1 Gb/s PacketPHYS

The 1Gb/s PacketPHYS consist of a physical coding sublayer (PCS), physical medium attachment sublayer (PMA), and physical medium dependent sublayer (PMD). The requirements for the PCS and PMA are described in IEEE Std 802.3-2008, Clause 36. The requirements for several different PMDs are described in IEEE Std 802.3-2008, Clause 38 and Clause 39. The following exceptions and changes apply to the specifications for the PCS, PMA, and PMD:

- a) Repeaters are not supported.
- b) The minimum frame size is 16 bytes.
- c) The maximum frame size is 9216 bytes.
- d) Autonegotiation is not used by this standard.
- e) Because autonegotiation is not used, the PRS-1 assumes full-duplex operation, flow control is disabled, and remote fault is not generated by the transmitter and if present is ignored by the receiver.

8.3.2.2 10 Gb/s PacketPHYS

The 10 Gb/s PacketPHYS consist of a physical coding sublayer (PCS), physical medium attachment sublayer (PMA), and physical medium dependent sublayer (PMD). The PacketPHYS may optionally include a WAN interface sublayer (WIS), and 10 Gb attachment unit interfaces (XAUI), and XGMII extender sublayers (XGXS). The requirements for the PCS are described in IEEE Std 802.3-2008, Clause 48 and Clause 49 for different types of PMDs. The requirements for the PMA are described in IEEE Std 802.3-2008, Clause 48 and Clause 51. The requirements for several different PMDs are described in IEEE Std 802.3-2008, Clause 52 and Clause 53. The optional WIS is described in IEEE-Std 802.3-2008, Clause 50, and the XGXS and XAUI are described in Clause 47. The following exceptions and changes apply to these requirements:

- a) The minimum frame size is 16 bytes.
- b) The maximum frame size is 9216 bytes.
- c) Flow control is disabled, and the remote fault condition is not generated and ignored if received.

8.4 SONET/SDH physical layer interfaces and PHYs

SONET/SDH physical layer interfaces and PHYs consist of the following elements:

- a) A reconciliation sublayer that maps the logical service primitives at the MAC physical layer service interface to and from standard electrical interfaces.
- b) A frame-mapped generic framing procedure (GFP), byte-synchronous HDLC-like framing, or LAPS framing adaptation sublayer.
- c) A SONET/SDH layer.

Two types of reconciliation sublayers are defined. One is the SONET/SDH reconciliation sublayer (SRS) that can be used with any adaptation sublayer. The other is the GFP reconciliation sublayer (GRS) that is intended for use only with a GFP adaptation sublayer. The two reconciliation sublayers are identical, except that the GRS conveys frame length information to the GFP adaptation sublayer to optionally eliminate the need to calculate the frame length parameter.

Both types of reconciliation sublayers are specified with 8-bit SPI-3, 32-bit SPI-3, SPI-4.1, and SPI-4.2 interfaces for various operating speeds, as defined by the corresponding OIF implementation agreements. These interfaces are optional, but they are used as a basis for specifying the SRS and GRS. An implementation that does not provide these interfaces shall behave functionally as if they were implemented.

The GFP, HDLC, and LAPS adaptation layers are built over the SONET/SDH path layer and not directly over a SONET/SDH medium. Because of the multiplexing and virtual concatenation capabilities of SONET/SDH, as well as the effects of frame encapsulation, there is not a one-to-one relationship between the bit rate at the RPR MAC sublayer and the speed of the physical media.

The interface between the SONET/SDH layer and the adaptation layer (either the GFP or the SONET/SDH) is the standard interface between the SONET/SDH path sublayer and any upper layer as defined in ANSI T1.105 (SONET) and in ITU-T G.707 and G.783 (SDH); it is not defined in this standard.

The SONET/SDH C2 signal labels specified in ANSI T1.105 (SONET) and ITU-T G.707 (SDH) to be used for GFP, HDLC-like framing, and LAPS framing sublayers are described in Table 8.1.

Table 8.1—C2 signal labels for GFP and HDLC-like framing

Adaptation sublayer type	C2 signal label
GFP framing	$1B_{16}$
HDLC-like framing	16_{16}
LAPS framing	18_{16}

The functionality performed in the SONET/SDH layer is beyond the scope of this standard; it is described in ANSI T1.105 (SONET) and ITU-T G.707 and G.783 (SDH). The higher layer paths, all the possible contiguous concatenated paths, and all the possible virtual concatenated paths are supported by this standard.

The SONET/SDH reconciliation sublayer for east and west at an RPR station shall both operate at the same rate.

The SONET/SDH layer may include protection mechanisms in addition to those provided by the RPR MAC. Interoperation with these protection mechanisms is described in Clause 11.

8.4.1 SONET/SDH reconciliation sublayers

8.4.1.1 SONET/SDH reconciliation sublayer (SRS)

The SONET/SDH reconciliation sublayer (SRS) maps the logical primitives at the MAC physical layer service interface to and from the following electrical interfaces:

- a) SPI Level 3 (SPI-3) with 8-bit transmit and receive data paths operating at 155 Mb/s to 622 Mb/s.
- b) SPI Level 3 (SPI-3) with 32-bit transmit and receive data paths operating at 155 Mb/s to 2.5 Gb/s.
- c) SPI Level 4 Phase 1 (SPI-4.1) operating at 622 Mb/s to 10 Gb/s.
- d) SPI Level 4 Phase 2 (SPI-4.2) operating at 622 Mb/s to 10 Gb/s.

Four versions of the SRS using these four interfaces are specified in Annex C.

8.4.1.2 GFP reconciliation sublayer (GRS)

The GFP reconciliation sublayer (GRS) maps the logical primitives at the MAC physical layer service interface to and from the following electrical interfaces:

- a) SPI Level 3 (SPI-3) with 8-bit transmit and receive data paths operating at 155 Mb/s to 622 Mb/s.
- b) SPI Level 3 (SPI-3) with 32-bit transmit and receive data paths operating at 155 Mb/s to 2.5 Gb/s.
- c) SPI Level 4 Phase 1 (SPI-4.1) operating at 622 Mb/s to 10 Gb/s.
- d) SPI Level 4 Phase 2 (SPI-4.2) operating at 622 Mb/s to 10 Gb/s.

Four versions of the GRS using the four electrical interfaces are specified in Annex C.

8.4.2 SONET/SDH adaptation sublayers

Three different adaptation sublayers for SONET/SDH interfaces are defined that differ from each other in the way RPR frames are mapped into the SONET/SDH path payload. The first adaptation sublayer uses frame-mapped GFP framing. The second adaptation sublayer uses byte-synchronous HDLC-like framing. The third adaptation sublayer uses LAPS. These three different adaptation layers do not interoperate with each other.

8.4.2.1 Frame-mapped GFP adaptation sublayer

Generic framing procedure (GFP) defined by ITU-T G.7041 is a standard method of mapping and delineating variable-length, byte-aligned payloads into byte-synchronous payload envelopes. Frame-mapped GFP defines a frame format for protocol data units (PDUs) transferred between GFP initiation and termination points. Frame-mapped GFP specifies a type of GFP mapping in which a client frame is received and mapped in its entirety into one GFP frame.

Frame-mapped GFP framing for RPR shall be performed in accordance with ITU-T G.7041 using a null extension header as defined by the extension header identifier (EXI), no GFP FCS field, and with a user payload identifier (UPI) corresponding to an RPR payload (listed in Amendment 2 of G.7041), as described in Table 8.2.

Table 8.2—GFP EXI and UPI values for RPR

GFP parameter	Name	Value
Extension header identifier (EXI)	EXI	0000 ₂
User payload identifier (UPI)	UPI	0000 1010 ₂

The GFP reconciliation sublayer (GRS) and the MAC sublayer provide an optional capability to propagate the PDU length value required by frame-mapped GFP, in order to optimize the forwarding of PDUs with minimal delay.

8.4.2.2 Byte-synchronous HDLC-like framing adaptation sublayer

RPR frames may be mapped into the SONET/SDH layer using byte-synchronous HDLC-like framing. This framing method is functionally identical to that defined in IETF RFC 2615 for point-to-point protocol (PPP) over SONET/SDH, and in IETF RFC 1662 for PPP in HDLC-like framing, except as follows:

- a) Link parameters are statically configured. Link control protocol (LCP) is not used.
- b) The mapping does not carry PPP frames; it carries only RPR frames.
- c) PPP-specific and legacy support is not provided or defined.
- d) The address, control, and FCS fields are part of the RPR frame and therefore are not used in this implementation of HDLC-like framing.
- e) The X⁴³+1 scrambler is always enabled.

Byte-synchronous HDLC-like framing for RPR shall be performed in accordance with IETF RFC 1662 using byte-stuffed framing, with references to PPP frames to be interpreted as RPR frames. The Flag sequence (used to indicate the beginning or end of a frame) and Control escape (used for transparency) are specified in Table 8.3.

Table 8.3—Flag and control escape sequences for HDLC-like framing

Byte sequence	Name	Value
Flag sequence	<i>flag</i>	7E ₁₆
Control escape	<i>control</i>	7D ₁₆

A diagram of the HDLC-like framing structure is shown in Figure 8.2. Instead of using LCP for link negotiation, byte-synchronous HDLC-like framing for RPR shall use the following statically defined link parameters:

- a) Address and Control Field compression is always used. The fields are not used.
- b) The Protocol Field is not used.

- c) The FCS is neither computed nor appended to the frame.
- d) The asynchronous control character map (ACCM) is not used.

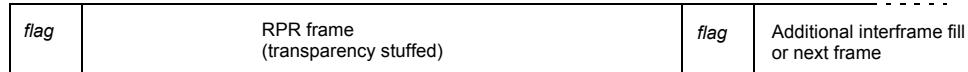


Figure 8.2—RPR in HDLC-like framing structure

8.4.2.3 LAPS framing adaptation sublayer

RPR frames may be mapped into the SONET/SDH layer using link access procedure-SDH (LAPS) framing, as specified by ITU-T X.85/Y.1321. LAPS performs byte-stuffed framing, like byte-synchronous HDLC-like framing, except that additional physical frame fields are defined.

LAPS framing for RPR shall be performed in accordance with ITU-T X.85/Y.1321. A diagram of the LAPS framing structure is shown in Figure 8.3.



Figure 8.3—RPR in LAPS framing structure

The Flag sequence and Control escape bytes are specified in Table 8.3 (same as byte-synchronous HDLC-like framing). The *address*, *control*, service access point identifier (*SAPI*), and *FCS* fields have the values specified in Table 8.4.

Table 8.4—LAPS frame field values for RPR

Field name	Value
<i>address</i>	04_{16}
<i>control</i>	03_{16}
<i>SAPI</i>	$FE03_{16}$
<i>FCS</i>	Calculated from beginning of address to end of RPR frame in accordance with ITU-TX.85/Y.1321.

8.4.3 SONET/SDH physical layer entities (PHYs)

The functionality performed in the SONET/SDH layer is beyond the scope of this standard. All valid SONET/SDH speeds, ranging from 150 Mb/s up to 10 Gb/s, are supported by this standard. All valid intermediate speeds, whether virtually or contiguously concatenated, are supported. Table 8.5 illustrates some of the supported speeds. The suffixes Nv and Pv are used to denote virtually concatenated payloads.

Table 8.5—SONET/SDH supported path layers and capacities

SONET path sublayer	SDH path sublayer	Bit rate
STS-3c-SPE	VC4	149.76 Mb/s
STS-12c-SPE	VC4-4c	599.04 Mb/s
STS-48c-SPE	VC4-16c	2396.160 Mb/s
STS-192c-SPE	VC4-64c	9584.640 Mb/s
STS-1-Nv SPE ^a	VC3-Nv ^a	N x 48.384 Mb/s ^a
STS-3c-Pv SPE ^b	VC4-Pv ^b	P x 149.76 Mb/s ^b

^aAll of the integer values of N are supported from N = 3 to N = 192.

^bAll of the integer values of P between 2 and 64 are supported. Virtual concatenated paths can support intermediate speeds from 300 Mb/s to 9.6 Gb/s.

8.5 Protocol Implementation Conformance Statement (PICS) proforma for Clause 8²⁰

8.5.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 8, MAC physical interface, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

8.5.2 Identification

8.5.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

8.5.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, MAC physical interface
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

²⁰Copyright release for PICS proformas: Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

8.5.3 Major capabilities/options

Item	Feature	Subclause	Value/Comment	Status	Support
C1*	PRS-1	8.3	1 Gb/s PacketPHY reconciliation sublayer	O.1	Yes [] No []
C2*	PRS-10	8.3	10 Gb/s PacketPHY reconciliation sublayer	O.1	Yes [] No []
C3*	SRS	8.4	SONET/SDH reconciliation sublayer	O.1	Yes [] No []
C4*	GRS	8.4	GFP reconciliation sublayer	O.1	Yes [] No []

8.5.4 PICS tables for Clause 8

8.5.4.1 Service interface

Item	Feature	Subclause	Value/Comment	Status	Support
SI1	Link status	8.2.3.3	PHY_LINK_STATUS.indication is generated when link status changes	M	Yes []

8.5.4.2 PRS-1

Item	Feature	Subclause	Value/Comment	Status	Support
G1	PRS-1 functional behavior	8.3.1.1	PRS-1 provides GMII functional behavior even if the interface is not implemented	C1:M	Yes []

8.5.4.3 PRS-10

Item	Feature	Subclause	Value/Comment	Status	Support
X1	PRS-10 functional behavior	8.3.1.2	PRS-10 provides XGMII functional behavior even if the interface is not implemented	C2:M	Yes []

8.5.4.4 GRS and SRS

Item	Feature	Subclause	Value/Comment	Status	Support
S1	SPI-3/4 functional behavior	8.4	GRS or SRS provides SPI-3/4 functional behavior even if the interface is not implemented	C3,C4:M	Yes []
S2	Matched ringlet rates	8.4	The same rate is required for both ringlets in SONET/SDH implementations	C3,C4:M	Yes []
S3	GFP framing compliance	8.4.2.1	GFP framing is compliant with ITU-T G.7041	C4:M	Yes []
S4	HDLC-like framing compliance	8.4.2.2	HDLC-like framing is compliant with IETF RFC 1662 with the clarifications noted in 8.4.2.2	C3:M	Yes []
S5	HDLC-like framing link parameters	8.4.2.2	HDLC-like framing uses the link parameters described in 8.4.2.2	C3:M	Yes []
S6	LAPS framing compliance	8.4.2.3	LAPS framing uses the link parameters described in 8.4.2.3	C3:M	Yes []

9. Frame formats

9.1 Overview

This clause defines in detail the frame structure for data communication systems using the RPR MAC. It defines the syntax and semantics of the various components of the MAC frame.

The following four frame formats are specified in this clause:

- a) data frame
- b) control frame
- c) fairness frame
- d) idle frame

The frame format does not include any layer 1 frame delineation.

9.2 Data frame format

Data frames are identified by the 2-bit *ft* field (see 9.6.3) being set to FT_DATA.

When the *daExtended* and *saExtended* fields are included, the data frame is qualified as an extended data frame. When the extended address fields are not included, the data frame is qualified as a basic data frame.

The basic data frame and extended data frame shall be implemented and transmitted as specified in 9.2.2.

9.2.1 Data frame sizes

The data frame size constraints are listed in Table 9.1. For rings on which regular frames have been negotiated (see 11.3.1), the maximum transfer length (MTU) is defined to be REGULAR_MAX. For rings on which jumbo frames have been negotiated (see 11.3.1), the MTU is defined as JUMBO_MAX. The maximum size negotiated for a ring is available in the variable, *mtuSize* (see 11.3.1).

Table 9.1—Data frame size constraints

Value	Name	Row	Description
24	DATA_MIN	1	Minimum data frame size
12	EXT_HDR_SIZE	2	Additional size of extended data frame
1616	REGULAR_MAX	3	Maximum regular data frame size
9216	JUMBO_MAX	4	Maximum jumbo data frame size

Row 9.1-1: The smallest data frame consists of a header, a 2-byte payload consisting of only the *protocolType* field, and a trailer.

Row 9.1-2: The number of additional bytes in an extended data frame header, beyond those in a basic data frame header. The extra bytes consist of the *daExtended* field and the *saExtended* field.

Row 9.1-3: The largest regular data frame has a 1500-byte *serviceDataUnit* field, and up to 92 reserved bytes.

Row 9.1-4: The largest jumbo data frame has a 9100-byte *serviceDataUnit* field, and up to 92 reserved

bytes.

NOTE—The number of reserved bytes depends upon whether the basic or extended data frame format is used. The basic data frame has 92 reserved bytes. The extended data frame has 80 reserved bytes.

The reserved bytes in REGULAR_MAX and JUMBO_MAX are not included in the transmitted or received frame. They exist only in the length definition, and they are intended to allow for future growth in the headers. The *serviceDataUnit* field cannot use the reserved bytes and cannot grow beyond 1500 bytes for a regular frame or 9100 bytes for a jumbo frame.

9.2.2 Data frame fields

The frame formats for data frames are illustrated in Figure 9.1.

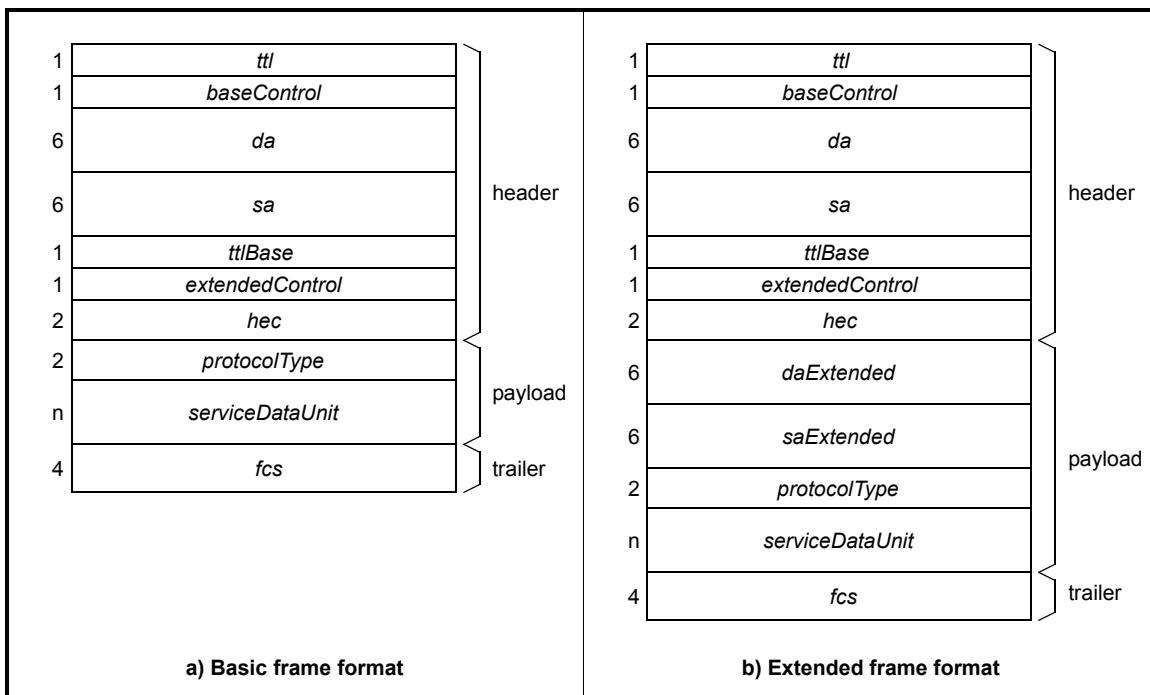


Figure 9.1—Data frame formats

9.2.2.1 *ttl*: An 8-bit (time to live) field that specifies the maximum number of hops the frame is expected to cover before reaching the destination. This field provides a mechanism to ensure that frames do not circulate forever on the ring (see 7.7.2.1).

9.2.2.2 *baseControl*: An 8-bit basic control field, specified in 9.6.

9.2.2.3 *da*: A 48-bit (destination address) field that specifies the station(s) for which the frame is intended. The *da* field contains either an individual or a group 48-bit MAC address (see 3.11) as specified in 9.2 of IEEE Std 802-2001.

9.2.2.4 *sa*: A 48-bit (source address) field that specifies the local station sending the frame. The *sa* field contains an individual 48-bit MAC address (see 3.11) as specified in 9.2 of IEEE Std 802-2001. The address contained in the *sa* field is always the MAC address of the station transmitting the frame onto the ring.

9.2.2.5 *ttlBase*: An 8-bit *ttlBase* field that is set to the initial value of the *ttl* field upon transmission of a data frame.

9.2.2.6 *extendedControl*: An 8-bit extended control field, specified in 9.7.

9.2.2.7 *hec*: A 16-bit (header error check) field that is a checksum of the header (see E.1). The *hec* is computed over the *ttl*, *baseControl*, *da*, *sa*, *ttlBase*, and *extendedControl* fields.

9.2.2.8 *daExtended*: A 48-bit (destination address, extended) field that specifies the final station(s) for which the frame is intended. The *daExtended* field contains either an individual or a group 48-bit MAC address (see 3.11) as specified in 9.2 of IEEE Std 802-2001.

The *daExtended* field is present only when the *ef* bit is 1.

9.2.2.9 *saExtended*: A 48-bit (source address, extended) field that specifies the original station sending the frame. The *saExtended* field contains an individual 48-bit MAC address (see 3.11) as specified in 9.2 of IEEE Std 802-2001. The address contained in the *saExtended* field is usually the MAC address of a non-local station and is located in this field because the value of the *sa* field is always the local station transmitting the frame onto the ring.

The *saExtended* field is present only when the *ef* bit is 1.

9.2.2.10 *protocolType*: A 16-bit field contained within the payload. When the value of *protocolType* is greater than or equal to 1536 (600_{16}), the *protocolType* field indicates the nature of the MAC client protocol (type interpretation), selecting from values designated by the IEEE Type Field Register. When less than 1536 ($0_{16} - 5FF_{16}$), the *protocolType* is interpreted as the length of the frame (length interpretation). The length and type interpretations of this field are mutually exclusive.

9.2.2.11 *serviceDataUnit*: A variable length field that contains the service data unit provided by the client.

9.2.2.12 *fcs*: A 32-bit (frame check sequence) field that is a cyclic redundancy check (CRC) of the frame (see E.2). The data frame *fcs* CRC32 is calculated starting from the byte following the *hec* through the end of the payload.

9.3 Control frame format

Control frames are identified by the 2-bit *ft* field (see 9.6.3) being set to FT_CONTROL. A control frame can be a broadcast or unicast frame.

The control frame shall be implemented and transmitted as specified in 9.3.2.

9.3.1 Control frame sizes

The control frame size constraints are listed in Table 9.2.

Table 9.2—Control frame size constraints

Value	Name	Row	Description
24	CONTROL_MIN	1	Minimum control frame size
1616	CONTROL_MAX	2	Maximum control frame size

Row 9.2-1: The smallest control frame consists of a header, a 2-byte payload consisting of *controlType* and *controlVersion* and a 0-byte *controlDataUnit*, and a trailer.

Row 9.2-2: The largest control frame has a 1500-byte *controlDataUnit* field and 92 reserved bytes.

The 92 reserved bytes in CONTROL_MAX are not included in the transmitted or received frame. They exist only in the length definition, and they are intended to allow for future growth in the headers. The *controlDataUnit* field cannot use the reserved bytes and cannot grow beyond 1500 bytes.

9.3.2 Control frame fields

Figure 9.2 shows the frame format for control frames.

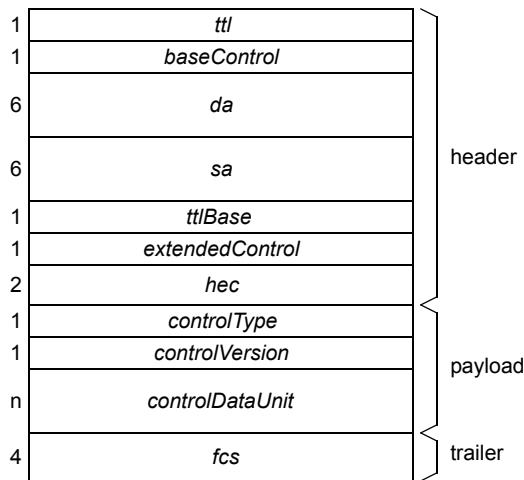


Figure 9.2—Control frame format

9.3.2.1 *ttl*: An 8-bit field specified in 9.2.2.1. The initial value is set as specified in each of the clauses describing control frames.

9.3.2.2 *baseControl*: An 8-bit field that affects frame-processing options, as specified in 9.6.

9.3.2.3 *da*: A 48-bit field specified in 9.2.2.3. Additionally, the *da* field in a control frame is constrained to be an address belonging to a ring-local station, or the broadcast address.

9.3.2.4 *sa*: A 48-bit field specified in 9.2.2.4.

9.3.2.5 *ttlBase*: An 8-bit field specified in 9.2.2.5.

9.3.2.6 *extendedControl*: An 8-bit extended control field specified in 9.7 and constrained as specified in Table 9.3.

Table 9.3—Control frame constraints for *extendedControl* subfield values

Field	Value	Description
<i>ef</i>	0	Control frames are generated only from ring-local stations.
<i>fi</i>	FI_NONE	Control frames are not flooded to clients.
<i>ps</i>	0	Control frames are initially sent without already having passed their source.
<i>so</i>	0	Control frames have no strict order or duplication avoidance requirements.

9.3.2.7 *hec*: A 16-bit field specified in 9.2.2.7.

9.3.2.8 *controlType*: An 8-bit field that identifies the type of control frame, selecting from values designated by this standard. Table 9.4 contains the currently defined control types.

Table 9.4—*controlType* values

Value	Name	Description
01 ₁₆	CT_STATION_ATD	Station attribute discovery frame
02 ₁₆	CT_TOPO_PROT	Topology and protection protocol frame
03 ₁₆	CT_TOPO_CHKSUM	Topology and protection checksum frame
04 ₁₆	CT_LRTT_REQ	Link round-trip time measurement request frame
05 ₁₆	CT_LRTT_RSP	Link round-trip time measurement response frame
06 ₁₆	CT_FDD	Fairness differential delay frame
07 ₁₆	CT_OAM_ECHO_REQ	OAM echo request frame
08 ₁₆	CT_OAM_ECHO_RSP	OAM echo response frame
09 ₁₆	CT_OAM_FLUSH	OAM flush frame
0A ₁₆	CT_OAM_ORG	OAM organization-specific frame
0B ₁₆	CT_OAM_SAS_NOTIFY	OAM SAS notify frame
0C ₁₆	CT_PIRC_MATE_STATUS	PIRC mate status notify frame
—	—	Reserved

9.3.2.9 *controlVersion*: An 8-bit field that is the version number associated with the *controlType* field. The *controlVersion* field provides a means of identifying future versions of the control frames. Initially, all control types are version 0. The handling of frames with other values is not specified by this standard.

9.3.2.10 *controlDataUnit*: A variable length field that is dependent on the value of the *controlType* field. The *controlDataUnit* field is specified in each of the clauses describing control frames.

9.3.2.11 *fcs*: A 32-bit (frame check sequence) field that is a cyclic redundancy check (CRC) of the frame (see E.2). The control frame *fcs* CRC is calculated starting from the byte following the *hec* through the end of the payload.

9.4 Fairness frame format

Fairness frames are identified by the 2-bit *fi* field (see 9.6.3) being set to FT_FAIRNESS.

The fairness frame shall be implemented and transmitted as specified in 9.4.2.

9.4.1 Fairness frame sizes

The size of an fairness frame is called FAIRNESS_SIZE and is defined to be 16 bytes.

9.4.2 Fairness frame fields

A fairness frame, as illustrated in Figure 9.3, is sent to MAC neighbors to convey data for the fairness algorithm specified in Clause 10. The values of the fairness frame fields are specified in 10.3.1.

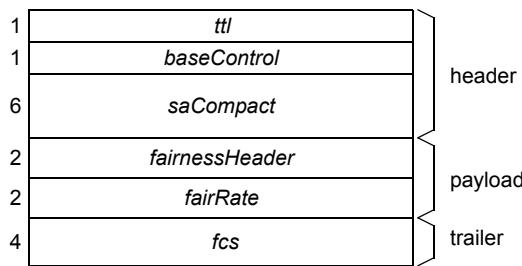


Figure 9.3—Fairness frame format

NOTE—The frame format for fairness frames is different from the frame format for data frames and other control frames. Fairness frames are not sent to specific destination nodes; they are sent to a station's nearest neighbor or broadcast to the entire ring. Therefore, the destination address does not contain any useful information and is omitted. Fairness frames are kept as small as possible to reduce jitter on other frames, to reduce their effective bandwidth consumption, and to minimize memory requirements for storing multiple fairness frames (especially when multi-choke fairness algorithms are used).

9.4.2.1 *ttl*: An 8-bit (time to live) field that is set to the value MAX_STATIONS by an originating station. Each subsequent station in a congestion domain sets the *ttl* to one less than the *ttl* of the last received SCFF. This allows a receiving station to compute the number of hops to the originating station as MAX_STATIONS - *frame.ttl*. A station is the origin of a fairness frame if it placed *myMacAddress* in the *frame.saCompact* field of the transmitted frame.

9.4.2.2 *baseControl*: An 8-bit field that affects frame-processing options, as specified in 9.6, and constrained as specified in Table 9.5.

Table 9.5—Fairness frame constraints for *baseControl* subfield values

Field	<i>stationProtectionConfig</i>	Field value	Description
<i>fe</i>	—	0	Fairness frames are excluded from fairness protocols.
<i>sc</i>	—	CLASS_A0	Fairness frames always use reserved bandwidth.
<i>we</i>	STEERING	0	Fairness frames on steering stations are not wrap eligible.
	WRAPPING	1	Fairness frames on wrapping stations are wrap eligible.

9.4.2.3 *saCompact*: A 48-bit field that contains an individual 48-bit MAC address (see 3.11) as specified in 9.2 of IEEE Std 802-2001. It specifies the station that provided the values contained in the *fairnessHeader* and *fairRate* fields. This station is not necessarily the station that generated this frame, which is always the upstream neighbor.

NOTE—The *saCompact* field is named differently from the *sa* field because of its different location within a frame.

9.4.2.4 *fairnessHeader*: A first 16-bit fairness-control field specified in 10.3.1.

9.4.2.5 *fairRate*: A second 16-bit fairness-control field specified in 10.3.1.

9.4.2.6 *fcs*: A 32-bit (frame check sequence) field (see E.2). The fairness frame *fcs* CRC is calculated starting from the byte following the *baseControl* field through the end of the payload.

9.5 Idle frame format

Idle frames are identified by the 2-bit *ft* field (see 9.6.3) being set to FT_IDLE.

The idle frame may be implemented, and if implemented, it is transmitted as specified in 9.5.2.

9.5.1 Idle frame sizes

The size of an idle frame is called IDLE_SIZE and is defined to be 16 bytes.

9.5.2 Idle frame fields

An idle frame, as illustrated in Figure 9.4, is sent to MAC neighbors to adjust the rate synchronization between the station and its neighbors as specified in 7.5.3.

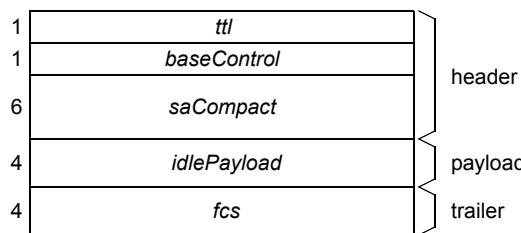


Figure 9.4—Idle frame format

NOTE—The frame format for idle frames is different from the frame format for data frames and other control frames. Idle frames are not sent to specific destination nodes, but they are sent to a station's nearest neighbor. Therefore, the destination address does not contain any useful information and is omitted. Idle frames are kept to a small, fixed size to reduce jitter on other frames and to reduce their effective bandwidth consumption.

9.5.2.1 *ttl*: An 8-bit (time to live) field specified in 9.2.2.1. The initial value is set to 1.

9.5.2.2 *baseControl*: An 8-bit field that affects frame-processing options, as specified in 9.6, and constrained as specified in Table 9.6.

Table 9.6—Idle frame constraints for *baseControl* subfield values

Field	Value	Description
<i>fe</i>	0	Idle frames are excluded from fairness protocols.
<i>sc</i>	CLASS_A0	Idle frames always use reserved bandwidth.

9.5.2.3 *saCompact*: A 48-bit field that contains an individual 48-bit MAC address (see 3.11) as specified in 9.2 of IEEE Std 802-2001. The *saCompact* field identifies the station that generated the frame, which is always the upstream neighbor.

NOTE—The *saCompact* field is named differently from the *sa* field because of its different location within a frame.

9.5.2.4 *idlePayload*: A 32-bit field reserved for future use. *idlePayload* shall be set to all zeros and ignored on receipt.

9.5.2.5 *fcs*: A 32-bit (frame check sequence) field (see E.2). The idle frame *fcs* CRC is calculated starting from the byte following the *baseControl* field through the end of the payload.

9.6 *baseControl* subfields

The 8-bit *baseControl* field consists of multiple subfields, as illustrated in Figure 9.5 and described in the remainder of this subclause.

**Figure 9.5—*baseControl* field format for data frames**

9.6.1 *ri*: A (ringlet identifier) bit that identifies the ringlet onto which the frame was originally transmitted. The *ri* values are specified in Table 9.7. The usage of the *ri* bit is described in 7.4.

Table 9.7—*ri* values

Value	Name	Description
0	RINGLET_0	Transmitted on ringlet0
1	RINGLET_1	Transmitted on ringlet1

9.6.2 *fe*: A (fairness eligible) bit that marks whether the frame is subject to the fairness algorithm. A value of 0 indicates that the frame is not fairness eligible, and a value of 1 indicates that the frame is fairness eligible (see 7.4). The defined meanings of the combined 2-bit *sc* field and *fe* bit are specified in Table 9.10. The combined 2-bit *sc* field and *fe* bit are interpreted as specified in Table 9.11.

9.6.3 *ft*: A 2-bit (frame type) field that identifies the type of the frame, as specified in Table 9.8. The usage of the *ft* field is specified in 7.3 and 7.4.

Table 9.8—*ft* values

Value	Name	Description
00_2	FT_IDLE	Idle frame
01_2	FT_CONTROL	Control frame
10_2	FT_FAIRNESS	Fairness frame
11_2	FT_DATA	Data frame

9.6.4 *sc*: A 2-bit (service class) field that identifies the service class of the frame (see 7.3). The *sc* values are specified in Table 9.9.

Table 9.9—*sc* values

Value	Name	Description
00_2	CLASS_C	classC
01_2	CLASS_B	classB
10_2	CLASS_A1	classA, subclassA1
11_2	CLASS_A0	classA, subclassA0

The defined meanings of the combined 2-bit *sc* field and *fe* bit are specified in Table 9.10. The combined 2-bit *sc* field and *fe* bit are interpreted as specified in Table 9.11.

Table 9.10—Service class and fairness eligible encodings

<i>sc</i>	<i>fe</i>	Name	Description
00_2	0	—	reserved
	1	CLASS_C	classC opportunistic
01_2	0	CLASS_B_CIR	classB—committed information rate
	1	CLASS_B_EIR	classB—excess information rate
10_2	0	CLASS_A1	classA reclaimable
	1	—	reserved
11_2	0	CLASS_A0	classA reserved
	1	—	reserved

Table 9.11—Service class and fairness eligible interpretations

Field Values		Interpretation		Description
sc	fe	Service class	Fairness eligible	
00 ₂	—	CLASS_C	yes	classC opportunistic
01 ₂	0	CLASS_B_CIR	no	classB—committed information rate
	1	CLASS_B_EIR	yes	classB—excess information rate
10 ₂	—	CLASS_A1	no	classA reclaimable
11 ₂	—	CLASS_A0	no	classA reserved

9.6.5 we: A (wrap eligible) bit that indicates whether the frame is eligible to be wrapped during a wrap condition. A value of 0 indicates that the frame is not wrap eligible, and a value of 1 indicates that the frame is wrap eligible. The usage of the *we* bit is described in Clause 7.

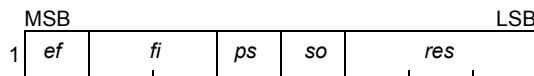
The *we* bit is ignored when received by steering stations. Its value has meaning only when received by wrapping stations. The value set for transmission from steering stations is 0.

NOTE—Although a frame that is not wrappable might have been steerable before being transmitted, once it has been transmitted it is no longer steerable. Therefore, this field is not to be interpreted to have any bearing on whether the frame was provided protection.

9.6.6 parity: In fairness frames and idle frames, a bit that protects the *ttl* and *baseControl* fields (because fairness frames and idle frames do not contain a *hec* field for header protection). The *parity* bit is set such that the total number of 1 bits in the *ttl* and *baseControl* fields, including the *parity* bit, is odd (see E.3). In data frames and control frames, the *parity* bit is reserved for future use and shall be set to 0, and ignored on receipt.

9.7 extendedControl subfields

The 8-bit *extendedControl* (extended ring control) field consists of multiple subfields (which have no interdependence with each other). The frame options are illustrated in Figure 9.6 and described in the remainder of this subclause.

**Figure 9.6—extendedControl field format for data and control frames**

9.7.1 ef: An (extended frame) bit indicates if the frame is an extended data frame. The MAC sets *ef* to 0 on transmit for frames using the basic data frame format illustrated in Figure 9.1-a. The MAC sets *ef* to 1 on transmit for frames using the extended data frame format illustrated in Figure 9.1-b. The determination of which frame format to use is described in 6.4.1.

9.7.2 *fi*: A 2-bit (flooding indication) field indicates whether the frame is flooded and, if so, by which means. The values of the *fi* field are specified in Table 9.12. The usage of the *fi* field is described in 7.6.3.

Table 9.12—*fi* values

Value	Name	Description
00_2	FI_NONE	no flood
01_2	FI_UNIDIR	unidirectional flood
10_2	FI_BIDIR	bidirectional flood
11_2	FI_RES	reserved type of flood

9.7.3 *ps*: A (passed source) bit that is used by wrapping systems (along with other fields) to prevent frame misorder and duplication by preventing the frame from being wrapped more than two times by the wrapping network. It is set to 0 when a frame is first transmitted by a station and set to 1 when a wrapped frame (i.e., a frame traveling on the secondary ringlet) passes the source station. The usage of the *ps* bit is described in 7.7.1.

9.7.4 *so*: A (strict order) bit that indicates whether the frame requires strict ordering requirements. A value of 1 indicates that all MACs through which the frame transits or through which it is received are to provide strict ordering for the frame. A value of 0 indicates that MACs are not required to provide strict ordering for the frame. The usage of the *so* bit is described in 7.4.

9.7.5 *res*: A 3-bit (reserved) field that shall be set to all zeros and ignored on receipt.

9.8 Protocol Implementation Conformance Statement (PICS) proforma for Clause 9²¹

9.8.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 9, Frame formats, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

9.8.2 Identification

9.8.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

9.8.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, Frame formats
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

²¹*Copyright release for PICS proforms:* Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

9.8.3 PICS tables for Clause 9

9.8.3.1 Frame formats

Item	Feature	Subclause	Value/Comment	Status	Support
FF1	basic data frame	9.2	basic data frames are implemented and are transmitted by the MAC in the format specified	M	Yes []
FF2	extended data frame	9.2	extended data frames are implemented and are transmitted by the MAC in the format specified	M	Yes []
FF3	control frame	9.3	control frames are implemented and are transmitted by the MAC in the format specified	M	Yes []
FF4	fairness frame	9.4	fairness frames are implemented and are transmitted by the MAC in the format specified	M	Yes []
FF5*	idle frame	9.5	idle frames are implemented and are transmitted by the MAC in the format specified	O	Yes [] No []

9.8.3.2 Reserved fields

Item	Feature	Subclause	Value/Comment	Status	Support
RES1	<i>fe</i> reserved	9.6.2	for CLASS_A0, CLASS_A1, FT_IDLE, or FT_FAIRNESS, <i>fe</i> is set to 0	M	Yes []
RES2	<i>parity</i> reserved	9.6.6	for FT_DATA or FT_CONTROL, <i>parity</i> is set to 0	M	Yes []
RES3	<i>res</i> reserved	9.7.5	<i>res</i> is set to all zeros	M	Yes []
RES4	<i>idlePayload</i> reserved	9.5.2.4	<i>idlePayload</i> is set to all zeros	FF5:M	Yes []

10. Fairness

10.1 Overview

This clause specifies an algorithm for the computation of fair rates of ringlet access for fairness eligible traffic. The use of fair rates prevents one station from occupying a disproportionate share of available ringlet capacity with respect to other stations on the ringlet.

The fairness procedures reside within the MAC control sublayer of the MAC, as illustrated by the shaded region of Figure 10.1.

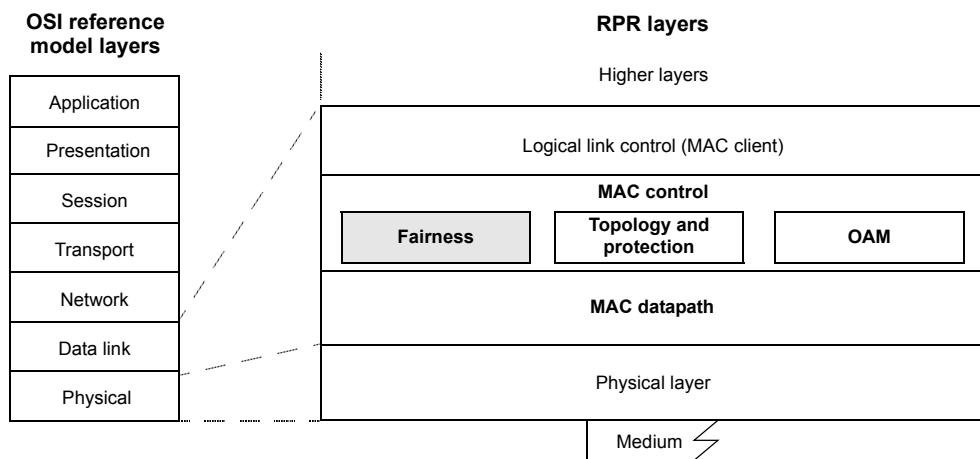


Figure 10.1—Fairness function relationship to the ISO/IEC OSI reference model

10.1.1 Fairness instances

A station maintains two instances of the fairness algorithm. Each instance regulates data traffic associated with one of the ringlets. A fairness instance is uniquely identified by the combination of the station address (*myMacAddress*) and the ringlet (*myRi*) whose traffic is regulated by that instance. Each fairness instance periodically sends congestion control information to its upstream neighbor via the opposing ringlet (*Other(myRi)*). The congestion control information is contained within a fairness frame carrying the ringlet identifier of the ringlet (*Other(myRi)*) on which it is sent.

Figure 10.2 illustrates the behavior of fairness instances on a closed ring. The ringlet0 fairness instance receives ringlet0 traffic rate information from a byte monitor associated with ringlet0. The ringlet0 fairness instance regulates traffic on ringlet0 by providing policing indications to the ringlet0 datapath instance. The ringlet0 fairness instance receives and sends fairness frames via the ringlet1 datapath instance. Fairness frames received and sent by the ringlet0 fairness instance carry a ringlet identification value of 1 (*frame.ri == 1*). The ringlet1 fairness instance behaves in a similar manner.

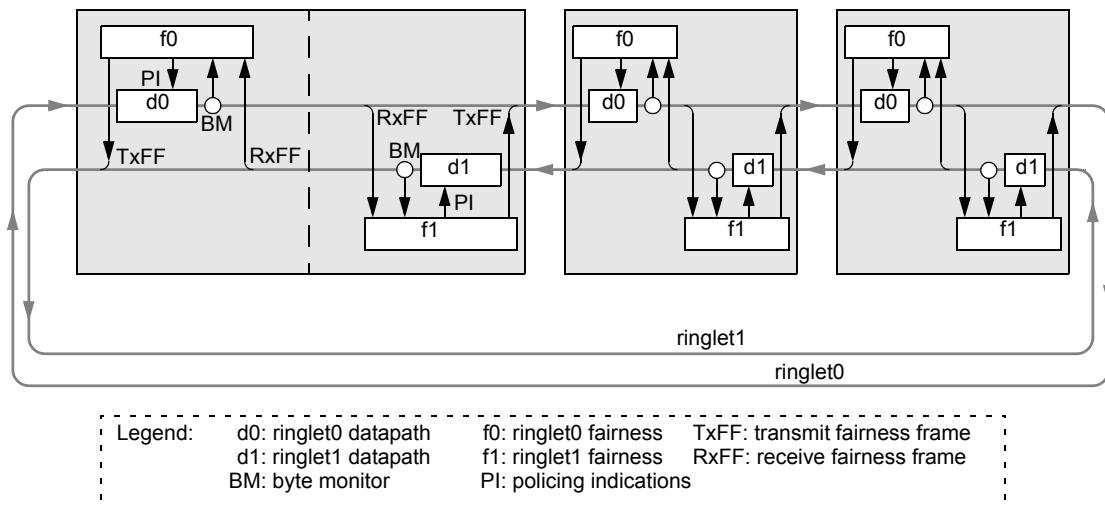
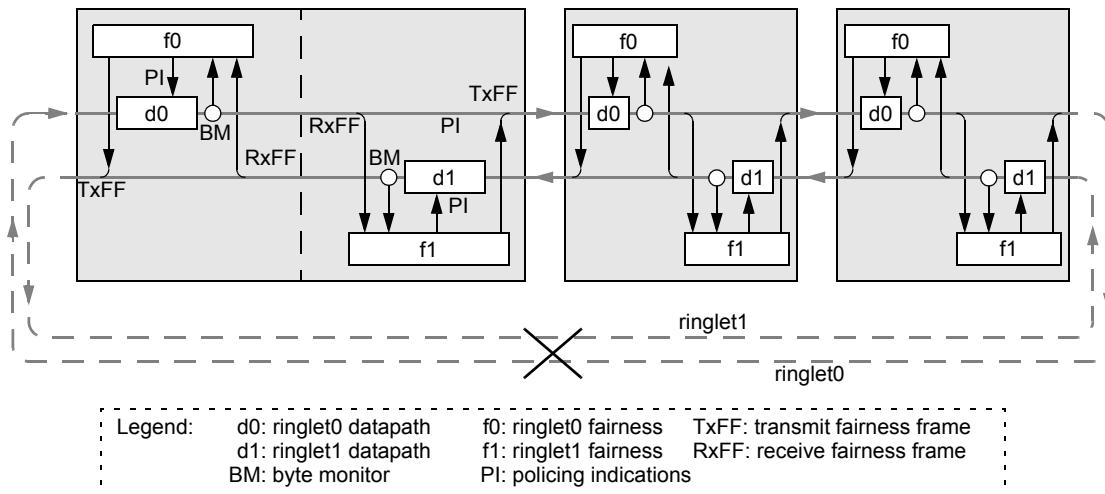
**Figure 10.2—Fairness instances on a closed ring**

Figure 10.3 illustrates fairness instances on an open ring using the steered method of protection. The identification and behavior of fairness instances is identical to that associated with a closed ring except that fairness frames transmitted by a fairness instance on an inactive span do not reach the upstream neighbor fairness instance and cannot, therefore, be received by the upstream neighbor fairness instance.

**Figure 10.3—Fairness instances on an open ring using steering**

NOTE—The fairness frames transmitted across a span are used by the protection protocol (see 11.6.2.2) in order to maintain a keepalive on the span. The use of fairness frames for this purpose is not visible to the fairness algorithm and is not described within this clause.

Figure 10.4 illustrates fairness instances on an open ring using the wrapped method of protection. The identification and behavior of fairness instances in stations that are not edge stations is identical to that of stations on a closed ring (see Figure 10.2). The identification and behavior of edge stations depends on whether the station performs edge wrapping (see 7.4.4.4) or center wrapping (see 7.4.4.2).

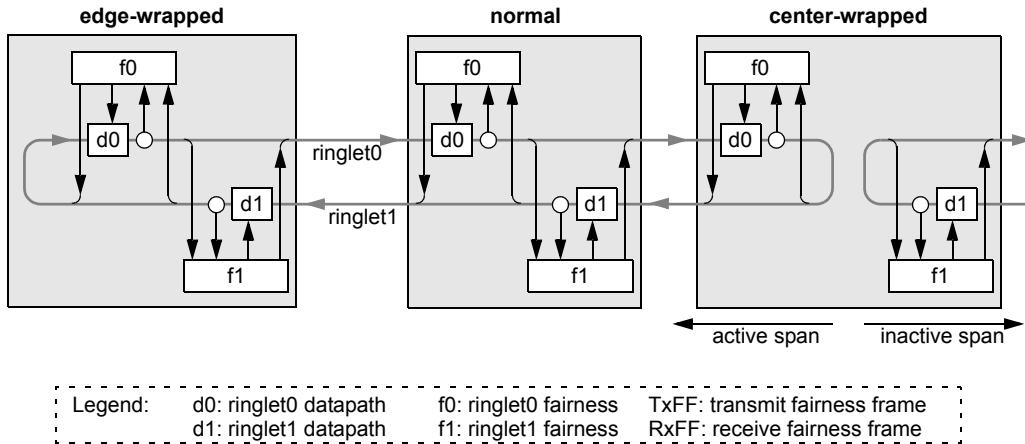


Figure 10.4—Fairness instances on an open ring using wrapping

As illustrated by the leftmost station of Figure 10.4, a wrapped station deploying an edge-wrapped datapath transits data through both datapath instances associated with the station.

As illustrated by the rightmost station of Figure 10.4, a wrapped station deploying a center-wrapped datapath transits data through a single datapath instance. The fairness instance associated with the active span receives and sends fairness frames via the active span. The fairness instance associated with the inactive span will not receive any fairness frames from its downstream neighbor due to the inactivity of the span. Fairness frames sent by the fairness instance associated with the inactive span will not reach the upstream neighbor fairness instance due to the inactivity of the span.

In the case of a closed ring, or an open ring deploying steering-based protection (see Figure 10.5-a), congestion is controlled independently on each ringlet. Congestion control information, like data traffic, is not transferred from one ringlet to the other. In the case of an open ring, congestion control information cannot cross a failed span on the ringlet.

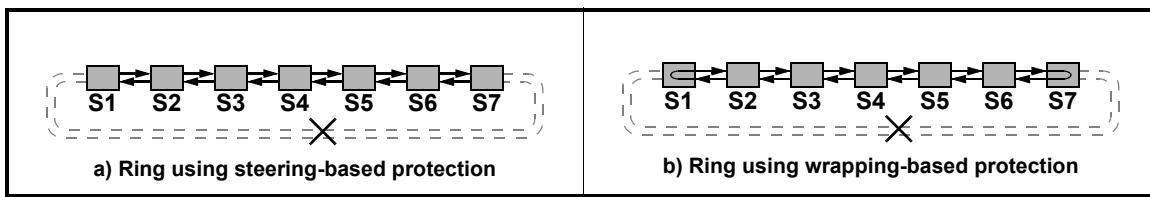


Figure 10.5—Path of fairness frames on an open ring

In the case of an open ring deploying the wrapping-based protection (see Figure 10.5-b), congestion control information can wrap from one ringlet to the other, allowing control of congestion associated with traffic that has wrapped from one ringlet to the other. In the case of an edge station deploying center wrapping, a fairness instance receives fairness frames on one ringlet and sends fairness frames on the opposing ringlet.

In the case of an edge station deploying edge wrapping, a fairness frame is processed sequentially by both fairness instances. In either case, fairness frames are propagated hop-by-hop along a path that is the reverse of that followed by the associated data traffic.

NOTE 1—In order to simplify descriptions within this clause, “station” is substituted for “fairness instance” when it is clear from the context that the reference is to an individual fairness instance within the station. This is generally the case, because fairness procedures are performed independently on each ringlet.

NOTE 2—This subclause contains forward references to terms defined in 10.2 and identifiers defined by the fairness state machines of 10.4.

10.1.2 Services and features

The fairness procedures described in this clause have the following characteristics:

- a) Support independent fairness operation per ringlet.
- b) Carry control information on the ringlet opposing that of the associated data flow.
- c) Regulate only classC and classB-EIR (i.e., fairness eligible) traffic.
- d) Compute fair rates associated with a source station (vs., for example, a source-destination station pair).
- e) Separately regulate aggregate fairness eligible traffic added to the ringlet and that portion of added fairness eligible traffic that transits a point of congestion.
- f) Scale fair rates in proportion to an administrative weight assigned to each fairness instance.
- g) Allow ringlet capacity not explicitly allocated to be treated as available capacity.
- h) Allow ringlet capacity explicitly allocated to subclassA1 or classB-CIR, but not in use, to be treated as available capacity (i.e., bandwidth reclamation).
- i) Support either single transit queue or dual transit queue deployment.
- j) Support either the aggressive or the conservative rate adjustment method.
- k) Optionally report rate information to the MAC client.
- l) Adjust fair rates within a few fairness round-trip times (FRTTs) of the occurrence of a change in traffic load (i.e., fast response time).

10.1.3 Fairness algorithm overview

The fairness algorithm regulates the addition of fairness eligible (see 7.3) traffic to a ringlet as follows:

- a) Congestion is controlled.
- b) Throughput is minimally affected by congestion control activities.
- c) Rate restrictions controlling congestion are applied fairly across stations contributing to congestion.

10.1.3.1 Identifying local congestion

A station deploying a dual queue MAC is congested when occupancy of the secondary transit queue (STQ; see 7.4.2) is excessive.

A station deploying a single queue MAC is congested when either or both of the following conditions are met:

- a) The rate of transmission is excessive relative to the capacity of the transmission link.
- b) Traffic is delayed excessively while awaiting transmission.

Congestion is undesirable as it can result in a failure to meet the end-to-end commitments associated with service classes (see 7.3). In the case of best-effort service, not associated with an explicit end-to-end commitment, congestion can allow stations upstream of the point of congestion to use more than their fair share of capacity available for best-effort traffic downstream of the point of congestion.

10.1.3.2 Congestion control

Figure 10.6 shows a congested station (S_6) and the set of contiguous stations (S_1 to S_6) contributing to the congestion at S_6 .

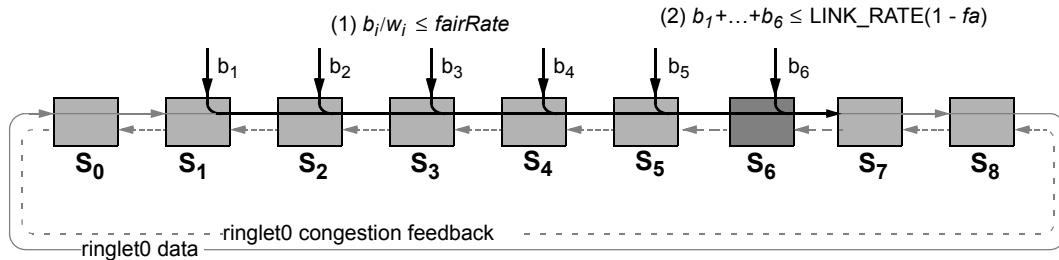


Figure 10.6—Congestion control objectives

Each contributing station is associated with a rate (b_i) at which it adds fairness eligible traffic crossing the outbound link of the congested station S_6 . The rate, b_i , is scaled by an administrative weight (w_i) that allows a station to add at a rate higher or lower than other stations without violating fairness principles. The **LINK RATE** represents the capacity of the ringlet, and fa represents the fraction of capacity consumed by higher precedence (classA and classB-CIR) allocated traffic. The objective of the fairness algorithm is to compute a **fairRate** applied to the contributing stations such that the following goals are met:

- $b_i/w_i \leq \text{fairRate}$: Contributing stations maximize their weight-adjusted rate without exceeding the **fairRate**.
- $b_1 + \dots + b_6 \leq \text{LINK_RATE}(1 - fa)$: The sum of fairness eligible traffic transmitted by the congested station maximizes use of available capacity without exceeding that capacity.

10.1.3.3 Advertising the **fairRate**

Figure 10.7 illustrates the case of a single congested station (S_6).

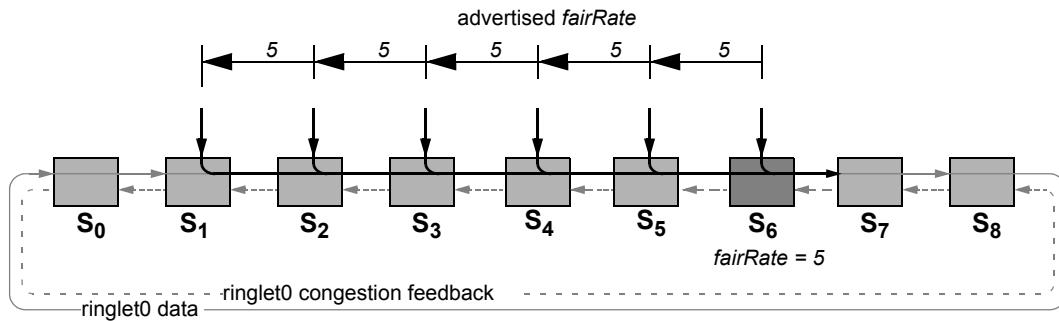


Figure 10.7—The **fairRate advertised upstream by a single congested station**

In order to meet the condition $b_i/w_i \leq fairRate$ at each contributing station (S_1 to S_6), the *fairRate* computed by the congested station is propagated hop by hop in the upstream direction, making the value known to each of the contributing stations.

The propagation of the *fairRate* is known as rate advertisement. Figure 10.8 illustrates the path of an advertisement propagated on ringlet1 in order to control congestion on ringlet0. The advertisement carries the identity of the ringlet on which it is transmitted (i.e., ringlet1).

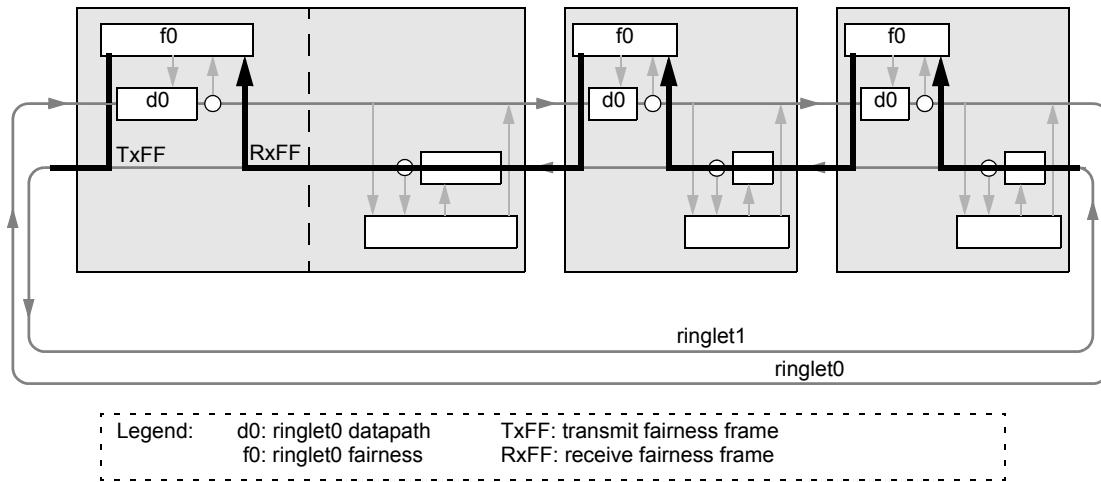


Figure 10.8—Path of a rate advertisement

The advertisement allows each contributing station to limit its rate, b_i , to the current weight-adjusted *fairRate*, resulting in changes to rate statistics measured on the downstream link of the congested station. The rate statistics are used to ensure that the condition $b_1 + \dots + b_6 \leq LINK_RATE(1 - fa)$ is met when adjusting the *fairRate*. The adjusted *fairRate* is then advertised upstream. As illustrated in Figure 10.9, the feedback between the congested station and the contributing stations allows continuous adjustment of rates to meet the fairness objectives.

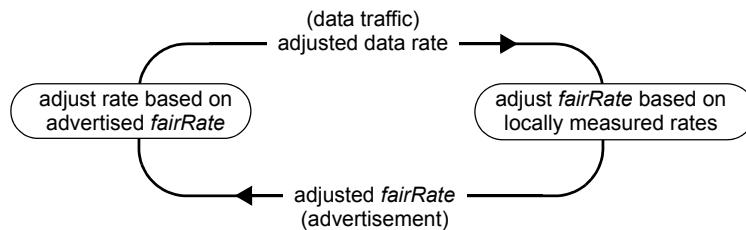


Figure 10.9—Congestion control feedback

Figure 10.10 provides an example in which multiple stations on the ringlet are congested. Each station independently computes a *fairRate*. Congested station S_4 computes a *fairRate* of 10 units and receives an advertised *fairRate* of 5 units from downstream neighbor S_5 . The advertisement originated at downstream station S_6 . Comparing the *fairRates*, S_4 determines that the *fairRate* of S_6 is more restrictive (i.e., smaller) than its own *fairRate*. S_4 propagates the advertised *fairRate* of S_6 (5 units) instead of advertising its own

fairRate (10 units). By advertising the more restricted *fairRate* of S_6 , it is ensured that the condition $b_i/w_i \leq fairRate$ is met for both sets of contributing stations (i.e., S_1 to S_4 and S_5 to S_6).

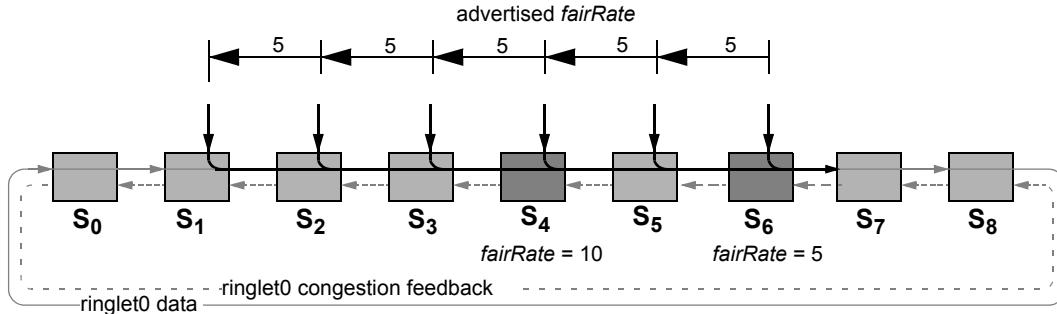


Figure 10.10—Received *fairRate* more restrictive than local *fairRate*

Figure 10.11 illustrates the case in which stations S_4 and S_6 are again congested but the *fairRate* computed by S_4 is more restrictive (i.e., smaller) than the *fairRate* computed by S_6 . Station S_4 computes a *fairRate* of 10 units and receives an advertised *fairRate* of 20 units originating from S_6 and propagated by S_5 . Comparing the *fairRates*, S_4 determines that its *fairRate* is smaller than that of S_6 . S_4 advertises its own *fairRate* rather than propagating the advertised *fairRate* of S_6 . By advertising the more restrictive *fairRate* of S_4 between S_4 and S_1 , and the less restrictive *fairRate* of S_6 between stations S_6 and S_4 , it is ensured that the condition $b_i/w_i \leq fairRate$ is met for both sets of contributing stations (i.e., S_1 to S_4 and S_5 to S_6), while not unnecessarily restricting the rates of stations S_5 and S_6 .

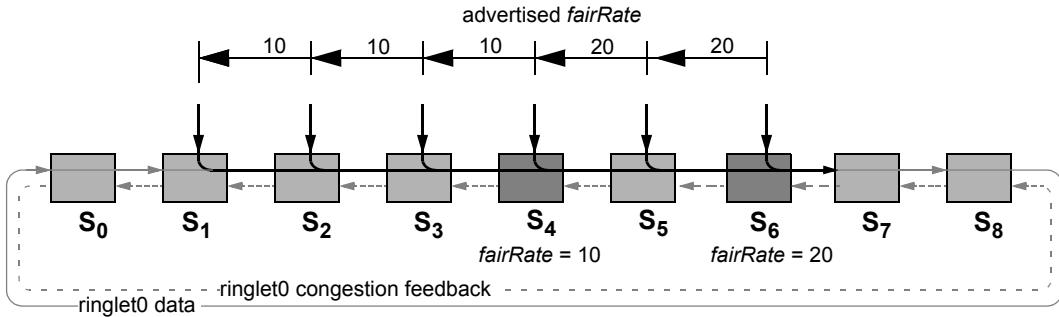


Figure 10.11—Local *fairRate* more restrictive than received *fairRate*

NOTE—The rate b_i (also known as the “contributing rate”) includes only that portion of added fairness eligible traffic crossing the outbound link of the congested station originating the advertised *fairRate* value received by station S_i .

As illustrated in Figure 10.12, an exception to the rule of propagating the more restrictive *fairRate* is made when it is determined that no station upstream of S_4 is adding traffic that passes through both S_4 and S_6 at a rate greater than the *fairRate* advertised by S_6 . In the example, the stations S_1 , S_2 , and S_3 add fairness eligible traffic at rates of 1 unit, 2 units, and 1 unit, respectively. Although S_4 does not have knowledge of these individual rates, it does measure the rate of transiting traffic bound for destinations beyond S_6 , in this case, 4 units. It can be inferred that no single contributing station, S_1 , S_2 , or S_3 , is contributing (i.e., adding fairness eligible traffic transiting S_6) at a rate greater than 4 units because it is known that the sum of this

traffic is not greater than 4 units. Propagation of the *fairRate* of 5 units beyond station S_4 would, therefore, not reduce the contributing rate of any station upstream of S_4 . S_4 advertises the value FULL_RATE, indicating to upstream stations that their contributing rates need not be restricted.

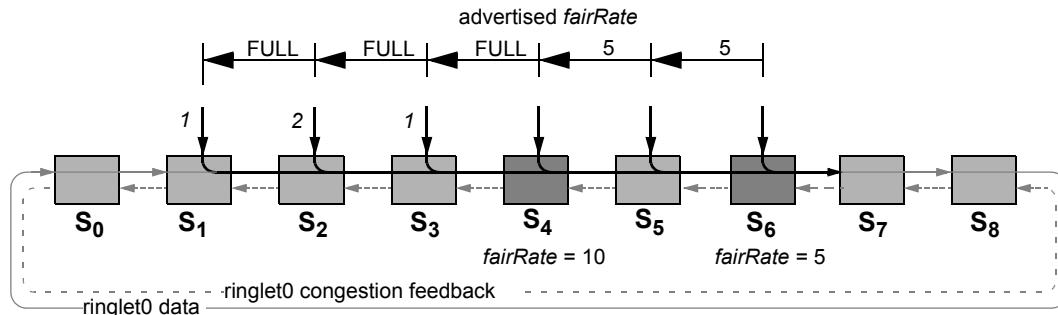


Figure 10.12—Advertising the FULL_RATE

Figure 10.12 also illustrates that uncongested stations receiving a FULL_RATE advertisement will themselves advertise the FULL_RATE upstream. Station S_3 , for example, receives a FULL_RATE advertisement from S_4 and advertises the FULL_RATE to S_2 .

As illustrated in the preceding examples, a station can advertise one of three possible values to its upstream neighbor:

- Its locally computed *fairRate*
- The *fairRate* advertised by its downstream neighbor
- The value FULL_RATE indicating that upstream stations are not contributing to downstream congestion

An advertisement carries the identity of its station of origin. A station advertising the FULL_RATE always identifies itself as the origin. In all other cases, the origin is the station whose locally computed *fairRate* and source MAC address is carried by the advertisement. The advertisement also carries a time to live field that is assigned the value MAX_STATIONS by the originating station and is decremented by each station through which it passes.

10.1.3.4 The congestion domain

A station advertising a locally computed non-FULL_RATE *fairRate* is said to lie at the head of a congestion domain. The downstream link of the head station is known as the congestion point.

The congestion domain extends upstream from the head station through all stations that propagate the advertisement until reaching a station that does not propagate the advertisement. The station at the upstream end of the congestion domain is known as the tail. A tail station receives an advertisement containing a non-FULL_RATE *fairRate* but does not propagate the received advertisement further upstream.

As illustrated in Figure 10.13, station S_1 is the tail of the congestion domain A. It receives from S_2 the advertisement originated by the head station S_3 of congestion domain A. S_1 terminates congestion domain A by advertising the FULL_RATE upstream to S_0 . Station S_3 illustrates a case in which the tail of downstream congestion domain B is also the head of upstream congestion domain A. Station S_3 receives an advertisement originated by S_4 but advertises its locally computed *fairRate* upstream to S_2 .

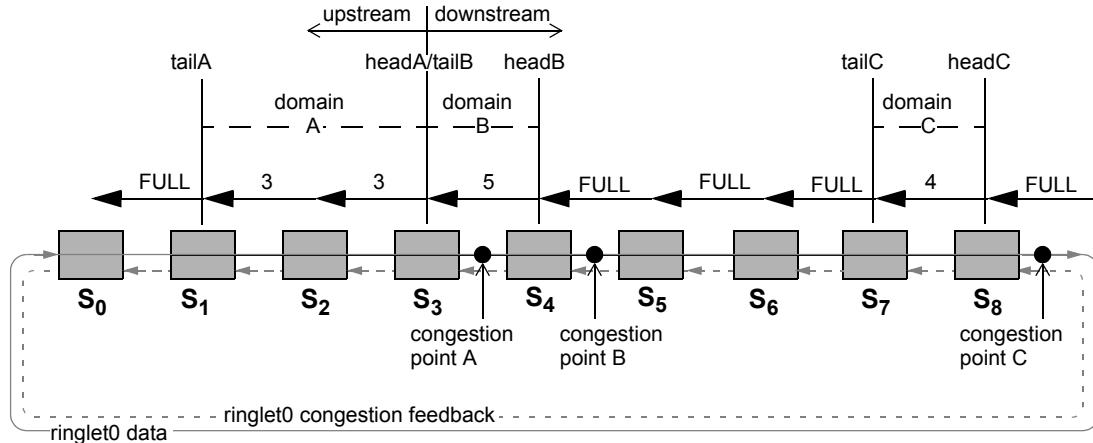


Figure 10.13—Congestion domains

A station propagates a received *fairRate* only when that *fairRate* is less than or equal to its locally computed *fairRate*. It is a property of a congestion domain that the *fairRate* of the head is not greater than (i.e., at least as restrictive as) that of any other station within the congestion domain. Non-head stations within the congestion domain may or may not be locally congested. Each station in the congestion domain sets its rate b_i based on the *fairRate* advertised by the head. The *FULL_RATE* is advertised on any link not within a congestion domain.

10.1.3.5 Identifying the tail of the congestion domain

The tail of a congestion domain is a station receiving a non-*FULL_RATE* advertisement but not propagating that advertisement (i.e., terminating the congestion domain). A congestion domain is terminated when either of the following conditions is met:

- a) The local station begins a new congestion domain (i.e., meets the conditions associated with the head of a congestion domain; see 10.1.3.4).
- b) No upstream station adds fairness eligible traffic transiting the congestion point at a rate exceeding the *fairRate* advertised by the head of the congestion domain.

When evaluating the latter condition, the local station is not aware of the add rates of individual upstream stations. Instead, the local station considers the measured total fairness eligible traffic transiting both itself and the congestion point. The traffic contributed by any individual upstream station cannot exceed the total traffic contributed by upstream stations. This total is taken as an upper bound on the fairness eligible add rate of any individual upstream station (see Figure 10.12 and accompanying text for an example of this case).

Normally, the fairness eligible traffic transiting both the local station and the congestion point is specified by *normLpFwRateCongested* (see 10.1.3.9 for a description of traffic rate measurements). If, however, the ring is wrapped between the head station and the local station, *normLpFwRateCongested* does not maintain an accurate count of frames transiting both the local station and the congestion point. *normLpFwRate* is known to be larger than *normLpFwRateCongested* as it includes all fairness eligible traffic transiting the local station. This value is taken as a conservative alternative to *normLpFwRateCongested*.

10.1.3.6 Allowed rate information

In addition to restricting fairness eligible add traffic transiting a congestion point, a station can also restrict the aggregate fairness eligible traffic it adds to the ringlet (i.e., independent of whether the traffic transits a congestion point). This aggregate rate is known as the *allowedRate*. The *allowedRate* cannot exceed the *unreservedRate* of the ringlet.

The rate, b_i , at which a station adds fairness eligible traffic transiting the congestion point is known within the fairness algorithm as the *allowedRateCongested*. The *allowedRate*, *allowedRateCongested*, and *hopsToCongestion* are collectively known as allowed rate information. A station periodically recomputes its *fairRate* and allowed rate information. The period between computations is known as the *agingInterval* because one of the computations performed is the aging of rate counters. The length of the *agingInterval* is based on the data rate of the ring and has the same value at all stations on the ring.

The allowed rate information is optionally transferred to the MAC client via an MA_CONTROL.indication having an opcode of SINGLE_CHOKE_IND (see Table 6.7). The allowed rate information can be used by the MAC client to perform fairness activities beyond the scope of the MAC (an example is provided in Annex J).

10.1.3.7 Reporting the *fairRate*

The *fairRate* computed by the station is reported by broadcast to all other stations on the ringlet using a multi-choke fairness frame (see 10.3.1.1). The time between successive broadcasts of a station's *fairRate* is known as the *reportingInterval* and is a configured multiple of the *advertisingInterval*. On learning the *fairRate* of a station, the *fairRate* is optionally transferred to the MAC client via an MA_CONTROL.indication having an opcode of MULTI_CHOKE_IND (see Table 6.7). The information can be used by the client to perform fairness activities beyond the scope of the MAC (an example is provided in Annex J).

10.1.3.8 Rate normalization

A rate communicated from one station to another is normalized in order to accomplish the following:

- Ensure that the rate is uniformly interpreted by stations on the ringlet.
- Scale the rate value to allow efficient encoding as an integer value within the 16-bit *fairRate* field (see 10.3.1.3) of the fairness frame.

Normalization of a *fairRate* is performed by dividing the locally significant rate by a normalization coefficient (*normCoef*). The *normCoef* is computed as the product of three values, as specified by Equation (10.1).

$$\text{normCoef} = \text{localWeight} * \text{rateCoef} * \text{ageCoef} \quad (10.1)$$

localWeight—Normalizes the rate consistent with a standard *localWeight* value of one.

rateCoef—Normalizes the rate consistent with a standard *lineRate* of 2.5 Gb/s.

ageCoef—Normalizes the rate consistent with a standard unit of bytes-per-*agingInterval*.

When received by a station, a normalized *fairRate* may be used as follows:

- Converted to a locally significant rate (i.e., localized) through multiplication by the local *normCoef*.
- Maintained in the normalized form for propagation to other stations.
- Compared with a local rate that has been normalized.

A normalized *fairRate* can be converted to a locally significant rate by multiplying the rate by the *normCoef* of the local station. The locally significant form of the *fairRate* is known as the *localFairRate*. The normalized form of the *fairRate* obtained by dividing the *localFairRate* by the *normCoef* is known as the *normLocalFairRate*.

NOTE—In order to simplify the explanations, this introductory text does not explicitly distinguish between the *localFairRate* and the *normLocalFairRate*. The more generic term *fairRate* is used instead. The terms *localFairRate* and *normLocalFairRate* are always used in the state machine descriptions of 10.4 to avoid ambiguity. Other rates, such as *LINK_RATE* (10.2.1) and the station bandwidth ATT fields *ringlet0ReservedBw* and *ringlet1ReservedBw* (11.4.2) are normalized by *rateCoef** *ageCoef*, without *localWeight*.

10.1.3.9 Measurement of traffic rates

A station makes use of locally collected rate statistics when computing rates. The following rates are maintained:

- a) *addRate*: Fairness eligible traffic added by the local station and bound for any station on the ringlet (see Figure 10.14-a).
- b) *addRateCongested*: Fairness eligible traffic added by the local station bound for destinations beyond the congestion point (see Figure 10.14-b).
- c) *fwRate*: Fairness eligible traffic transiting the local station and bound for any destination on the ringlet (see Figure 10.14-c).
- d) *fwRateCongested*: Fairness eligible traffic transiting the local station and bound for a destination that is beyond the congestion point (see Figure 10.14-d).
- e) *nrXmitRate*: All traffic except traffic of subclassA0 transmitted (i.e., added or transited) by the local station (see Figure 10.14-e).

Each byte added or transited by the station is evaluated to determine which, if any, of the five rate counters (*addRate*, *addRateCongested*, *fwRate*, *fwRateCongested*, and *nrXmitRate*) are to be incremented. It is permissible to delay the processing of the per-byte activities, but no delay of more than 256 bytes shall be allowed.

Periodically, each rate is smoothed with respect to past rate measurements. This is done by computing a weighted average of the current rate value and the previous smoothed rate value. The weight coefficient associated with the weighted average is a configured parameter. This smoothing method is known as low-pass filtering. The current smoothed rates are maintained separately from the rate counters. The smoothed rates are identified as the *lpAddRate*, *lpFwRate*, *lpFwRateCongested*, and *lpNrXmitRate*, respectively. The rate counters are themselves unchanged by low-pass filtering.

NOTE—The low-pass filtered value of *addRateCongested* is not used and is, therefore, not shown.

After the rate counters have been referenced in the computation of smoothed rates, an aging operation is applied to the rate counters. During the period between aging operations, known as an *agingInterval*, each rate counter maintains a byte count. Aging converts the byte counts to values that asymptotically approach the data rate, assuming that the rate of offered traffic on the ringlet is stable over a period of time.

In addition to maintaining the rate counter values as rates, aging has the effect of preventing the overflow of the rate counter and smoothing the value of the rate counter with respect to previous rate counter values. A rate counter is aged by multiplying its value by the expression $(ageCoef-1)/ageCoef$. The *ageCoef* specifies the relative weights assigned to (a) the change in value of the rate counter during the most recent *agingInterval* and (b) the value of the aged rate counter at the expiration of the previous *agingInterval*. The use of aging to derive rates is further explained in 10.5.

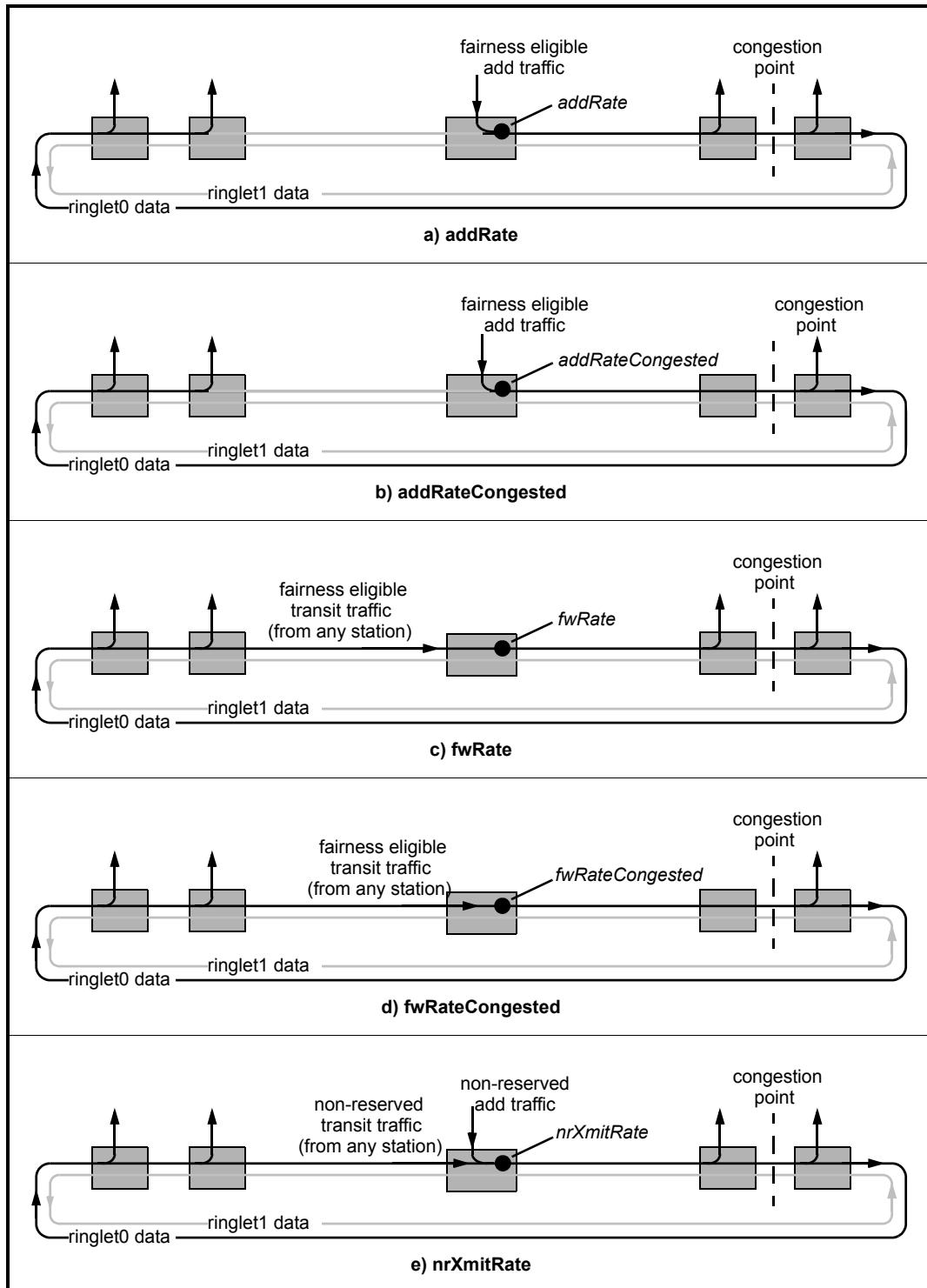


Figure 10.14—Traffic rate measurements

10.1.3.10 Computation of policing indications

The allowed rates of fairness eligible traffic (see 10.1.3.2) and the measured traffic rates (see 10.1.3.9) are used to maintain boolean policing indications as follows:

- a) *addRateOK*: Indicates whether a byte of a fairness eligible frame (i.e., fairness eligible marked frame) is allowed to be transferred from the MAC client to the MAC.
- b) *addRateCongestedOK*: Indicates whether a byte of a fairness eligible frame bound for a destination beyond the congestion point is allowed to be transferred from the MAC client to the MAC.

The indication *addRateOK* is set to allow traffic to pass if the measured *addRate* does not exceed the current *allowedRate*, and the station is determined to have sufficient downstream capacity to serve upstream stations. Sufficient downstream capacity is available when either of the following conditions is met:

- a) Upstream stations are not in need of more downstream capacity because traffic is not accumulating in the STQ.
- b) Upstream stations are not “starved” for downstream capacity, and the station is not fully congested with respect to STQ occupancy.

The indication *addRateCongestedOK* is set to allow traffic to pass if the *addRateOK* conditions have been met and the *addRateCongested* does not exceed the *allowedRateCongested*.

The policing indications are evaluated after a byte, or a group of up to 256 bytes, has been counted. The policing indications are referenced by the datapath in the setting of the *sendC* indication (see 7.5.7).

10.1.3.11 Adjusting the *fairRate*

As previously described in 10.1.3.2, a station adjusts its *fairRate* at each *agingInterval*. A station implements exactly one of the following two methods of rate adjustment:

- a) Aggressive: Provides responsive adjustments that favor utilization of capacity over rate stability.
- b) Conservative: Provides highly damped adjustments that favor rate stability over utilization of capacity.

The aggressive method is the simpler of the two methods and is described as follows:

- a) The station is congested (see 10.1.3.1): The *fairRate* is set to the *lpAddRate*. As previously described in 10.1.3.9, the *lpAddRate* is the smoothed rate of addition to the ringlet. If the station is the head of a congestion domain, its *fairRate* will be propagated to all stations within the congestion domain.
- b) The station is not congested: The *fairRate* need not be modified. A station deploying the aggressive method does not reference the *fairRate* value when in the un congested state.

NOTE—When in the un congested state, a station deploying the aggressive method does not reference the locally computed *fairRate* value. The *allowedRate* (see Table 10.8) is assigned the value of the *maxAllowedRate*. The *allowedRateCongested* (see Table 10.8) is either allowed to ramp toward the *maxAllowedRate* (in the absence of downstream congestion) or set to the localized (i.e., denormalized) value of the received *fairRate* (in the presence of downstream congestion). Neither rate depends upon the locally computed *fairRate*. The *fairRate* field of a transmitted SCFF (see Table 10.10) contains either the FULL RATE value (in the absence of downstream congestion) or the received *fairRate* (in the presence of downstream congestion). The *fairRate* field of a transmitted MCFF (see Table 10.11) contains the FULL RATE value.

The conservative method differs from the aggressive method in that a station can remain in a congested state after the congestion conditions (see 10.1.3.1) are removed. This provides hysteresis in the transition between congested and un congested states and prevents rate oscillation. In most cases, the conservative method requires that the *fairRate* not be adjusted until sufficient time has passed to ensure that the effect of any previous adjustment has been observed. This waiting period is known as the fairness round-trip time (FRTT) and is specified in 10.4.10. The conservative method is described as follows:

- a) A station in the uncongested state becomes locally congested: The *fairRate* is assigned an initial value equal to a weighted fair share of the *unreservedRate*.
- b) A station in the congested state is locally congested, and an FRTT has elapsed since the last *fairRate* adjustment: The *fairRate* is ramped down, i.e., reduced by a fraction of its current value. Ramping reduces congestion gradually in contrast to aggressive rate adjustment that reduces the *fairRate* directly to the *lpAddRate*.
- c) A station in the congested state is not locally congested, and an FRTT has elapsed since the last *fairRate* adjustment: The *fairRate* is ramped up, i.e., increased by a fraction of the difference between its current value and the *unreservedRate*. Ramping up increases rates gradually in contrast to aggressive rate adjustment, which increases the *fairRate* directly to the *unreservedRate*.
- d) A station is in the congested state, and the *fairRate* is close to or above the *unreservedRate* (i.e., is fully ramped up): The *fairRate* is set to the *unreservedRate*, and the station enters an uncongested state. This is in contrast to the aggressive method where the *fairRate* is set to the *unreservedRate* as soon as the station is no longer locally congested.
- e) A station is seriously congested due to high STQ occupancy: The *fairRate* is assigned a weighted fair share of the sum of the *lpAddRate* and *lpFwRate*, provided that value is less than the current *fairRate*. This condition does not occur in the aggressive method as the *fairRate* is immediately reduced when congestion occurs.

NOTE—The capability of computing a weighted fair share is optional. Stations without this capability set the *fairRate* to the *lpAddRate*.

Stations deploying the aggressive and conservative methods interoperate on the ringlet. Both methods converge to the *fairRate* value described in 10.1.3.2 when the offered traffic at stations on the ringlet is constant.

10.1.3.12 Computing the *allowedRate*

The *allowedRate* specifies the aggregate rate at which fairness eligible traffic can be added to the ringlet by the local station. As in the case of the *fairRate* computation (see 10.1.3.11), the method of computing the *allowedRate* depends on the method of rate adjustment deployed by the station. In the case of aggressive rate adjustment, the *allowedRate* is always assigned the configured maximum rate (*maxAllowedRate*) of the station.

In the case of conservative rate adjustment, the *allowedRate* is computed as follows:

- a) The station is in the uncongested state: The *allowedRate* is ramped up toward the *maxAllowedRate*. This is in contrast to the aggressive method in which the *allowedRate* is directly set to the *maxAllowedRate*.
- b) The station is in the congested state: The *allowedRate* is assigned the locally computed *fairRate* (or the *maxAllowedRate* if the *fairRate* exceeds the *maxAllowedRate*).

When using the conservative method, the hysteresis associated with the transition from the congested to the uncongested state prevents rate oscillation.

10.1.3.13 Computing the *allowedRateCongested*

The *allowedRateCongested* specifies the rate at which fairness eligible traffic destined for stations beyond the congestion point can be added to the ringlet. Unlike the *fairRate* and *allowedRate* computations, the computation of the *allowedRateCongested* does not depend on the rate adjustment method. If the station lies within a congestion domain, the *allowedRateCongested* is set to the *fairRate* of the head. For stations lying upstream of the head, this value is contained in the most recently received rate advertisement. This assignment reflects the rule that a station lying within a congestion domain is not permitted to add fairness

eligible traffic transiting the congestion point at a rate greater than the *fairRate* of the head. If the station does not lie within a congestion domain, the *allowedRateCongested* is ramped up, allowing stations to effectively use available capacity until congestion is encountered.

10.1.3.14 Computing the *hopsToCongestion*

The *allowedRateCongested* is accompanied by the *hopsToCongestion* identifying the distance, in hops, between the local station and the head of the congestion domain in which the station lies. The *hopsToCongestion* is normally computed as the difference between the *ttl* value inserted at the origin station (MAX_STATIONS) and the *ttl* value of the received advertisement after *ttl* decrement. An exception is made when the received advertisement has wrapped between the origin and the local station. In this case, the *hopsToCongestion* is set to the number of hops between the local station and the downstream wrap point (*ringInfo.totalHopsTx[myRi]*) maintained by the protection protocol (see Clause 11).

NOTE—The topology database downstream of a wrap point does not accurately reflect the sequence of stations traversed by a fairness frame. The database cannot be used to determine whether the congestion point is transited to reach a specific destination. It is known, however, that the congestion point does lie downstream of the wrap point. The distance to the wrap point is, therefore, taken as a conservative alternative to MAX_STATIONS – *frame.ttl* when computing the *hopsToCongestion*.

The *hopsToCongestion* is assigned the value MAX_STATIONS when the local station does not lie within a congestion domain.

10.1.3.15 The fairness state machines

The fairness state machines are described in 10.4. Except for the processing of received fairness frames, each state machine performs activities at fixed intervals. The computation of these intervals is summarized by Table 10.1.

Table 10.1—Fairness operations

Interval	Associated processing	Value
byte inter-arrival	datapath byte counts policing indications	$8 / \text{lineRate}$
<i>agingInterval</i>	local rate computation	specified by Table 10.2
<i>advertisingInterval</i>	<i>fairRate</i> advertisement	(size of the fairness frame in bits) / (<i>lineRate</i> in bits/second * the configured <i>advertisementRatio</i>)
<i>reportingInterval</i>	<i>fairRate</i> broadcast	<i>reportCoef</i> * <i>advertisingInterval</i>
<i>activeWeightsInterval</i>	optional <i>activeWeights</i> computation	<i>activeWeightsCoef</i> * <i>agingInterval</i> where the <i>activeWeightsCoef</i> is a configured value

10.2 Terms, definitions, variables, and routines

10.2.1 Common state machine definitions

The definitions listed in this subclause are referenced by multiple state machines.

FULL_RATE

A value of *fairRate* indicating the absence of congestion.

Value: FFFF₁₆

LINK_RATE

The rate of the link in units of bytes per *agingInterval*, as specified in Equation (10.2). A value of **LINK_RATE** is associated with each PHY type and is specified by Table 10.16.

$$(ageCoef * rateCoef * localWeight) \quad (10.2)$$

MAX_WEIGHT

The maximum value that can be assigned to a *localWeight*.

Value: 255

10.2.2 Common state machine variables

The variables listed in this subclause are referenced by multiple state machines.

activeWeights

A value nominally representing the sum of the weights of stations contributing fairness eligible traffic to the ringlet, optionally rounded to the next higher power of two.

Range: [1, MAX_WEIGHT * MAX_STATIONS]

activeWeightsCoef

A value indicating the number of *agingIntervals* that elapse between successive computations of *activeWeights*.

Range: [8, 512]

Default: 64

activeWeightsDetection

A boolean value indicating whether *activeWeights* computation is deployed by the station. This value is applicable only to the case of conservative rate adjustment.

TRUE—Active weights computation is deployed.

FALSE—(Otherwise.)

Default: Behaves as though set to FALSE.

addRate

The rate, in units of bytes per *ageCoef agingIntervals*, at which fairness eligible traffic is added to the ringlet.

addRateOK

Indicates whether fairness eligible traffic is allowed to be added to the ringlet.

TRUE—Traffic can be added.

FALSE—(Otherwise.)

addRateCongested

The rate, in units of bytes per *ageCoef agingIntervals*, at which fairness eligible traffic intended for destinations downstream of the congestion point, is added to the ringlet.

addRateCongestedOK

Indicates whether fairness eligible traffic bound for a destination beyond the congestion point is allowed to be added to the ringlet.

TRUE—Traffic can be added.

FALSE—(Otherwise.)

advertisingInterval

The interval at which SCFFs are sent by a station, as defined by Equation (10.3).

$$((8 * FAIRNESS_SIZE) / (lineRate * advertisementRatio)) \quad (10.3)$$

advertisementRatio

The ratio of the link capacity reserved for fairness frames to the *lineRate*.

Range: [0.00025, 0.01]

Default: 0.00125

ageCoef

The coefficient used by the aging procedure to specify the relative weights assigned to (a) the change in the value of a rate counter during the most recent *agingInterval* and (b) the value of the rate counter at the end of the previous *agingInterval*.

Allowed: {1, 2, 4, 8, 16}

Default: 4

agingInterval

The interval at which the *localFairRate* computation is performed. The *agingInterval* allowed values are specified by Table 10.2.

Table 10.2—The *agingInterval* as a function of *lineRate*

<i>lineRate</i>	<i>agingInterval</i>
$\geq 622 \text{ Mb/s}$	100 μs
$< 622 \text{ Mb/s}$	400 μs

allowedRate

The rate, in units of bytes per *ageCoef agingIntervals*, at which a station is allowed to add fairness eligible traffic.

allowedRateCongested

The rate, in units of bytes per *ageCoef agingIntervals*, at which a station is allowed to add fairness eligible traffic intended for destinations downstream of the congestion point.

downstreamCongested

A boolean value indicating whether or not the local station has detected downstream congestion.

TRUE—Downstream congestion has been detected.

FALSE—(Otherwise.)

fddInterval

The interval, in milliseconds, at which FDD frame pairs are sent by the tail of a congestion domain.

Allowed: [10, 1000]

Default: 100

fddMaxWait

The interval, in milliseconds, after which a received classA FDD is discarded if a classC FDD has not yet been received.

Allowed: [10, 1000]

Default: 100

forwardedFeFrame

An array having one entry per station on the ringlet recording whether the local station has received at least one fairness eligible frame from a remote station during the *activeWeightsInterval*. This information is input to the computation of *activeWeights*.

frtt

The value, in milliseconds, of the fairness round-trip time (see 3.2.32).

Allowed: [0, 1000]

Default: 1

fwRate

The rate, in units of bytes per *ageCoef agingIntervals*, at which fairness eligible traffic transits the local station.

fwRateCongested

The rate, in units of bytes per *ageCoef agingIntervals*, at which fairness eligible traffic intended for destinations beyond the congestion point, transits the local station.

hopsToCongestion

The distance, in hops, between the local station and the head of the congestion domain. The variable is assigned the value MAX_STATIONS if the local station does not lie within a congestion domain.

iAmTail

A boolean value indicating whether the local station is the tail of a congestion domain.

TRUE—The local station is the tail of a congestion domain.

FALSE—(Otherwise.)

localCongested

A boolean value indicating whether the local station is congested.

TRUE—The local station is congested.

FALSE—(Otherwise.)

localFairRate

The rate at which the station is allowed to add fairness eligible traffic to the ringlet in the absence of downstream congestion, specified in units of bytes per *ageCoef agingIntervals*.

localWeight

An administrative weight assigned to each fairness instance to permit the scaling of *fairRate* values among stations on the ringlet. This allows one station to use a larger share of available capacity than another station without violating fairness principles. A station may constrain *localWeight* values to a power of two.

Range: [1, 255]

Default: 1

lpAddRate

A smoothed version of the *addRate* obtained by applying the low-pass filter function to that variable.

lpCoef

The coefficient used by the low-pass filter procedure to specify the relative weights applied to (a) the increase in the rate-count value during the most recent *agingInterval* and (b) the previous low-pass filtered rate. The former is assigned a weight of 1 and the latter a weight of (*lpCoef*-1).

Allowed: {16, 32, 64, 128, 256, 512}

Default: 64

lpFwRate

A smoothed version of the *fwRate* obtained by applying the low-pass filter function to that variable.

lpFwRateCongested

A smoothed version of the *fwRateCongested* obtained by applying the low-pass filter function to that variable.

lpNrXmitRate

A smoothed version of the *nrXmitRate* obtained by applying the low-pass filter function to that variable.

maxAllowedRate

The maximum permitted value of the *allowedRate*.

Range: [1, LINK_RATE*rateCoef*ageCoef]

Default: LINK_RATE*rateCoef*ageCoef

normCoef

The coefficient used to normalize (see 10.1.3.8) a local rate value to a rate value that can be uniformly interpreted by all stations on the ringlet. The value of *normCoef* is computed as the product of *ageCoef*, *rateCoef*, and *localWeight*, as defined by Equation (10.4).

$$(ageCoef * rateCoef * localWeight) \quad (10.4)$$

normLocalFairRate

The normalized value of the *localFairRate*.

normLpFwRate

The normalized value of *lpFwRate*.

normLpFwRateCongested

The normalized value of *lpFwRateCongested*.

nrXmitRate

The rate, in units of bytes per *ageCoef agingIntervals*, at which unreserved (i.e., not of subclassA0) traffic is transmitted to the ringlet from the add queues and transit queues of the local station.

rampDnCoef

The coefficient used for ramping-down a rate.

Allowed: {16, 32, 64, 128, 256, 512}

Default: 64

rampUpCoef

The coefficient used for ramping-up a rate.

Allowed: {16, 32, 64, 128, 256, 512}

Default: 64

rateCoef

The coefficient used to scale a locally computed rate to a standard *lineRate* of 2.5 Gb/s. The value of *rateCoef* is a function of the PHY type (see Table 10.16).

rateHighThreshold

Rate at or above which congestion on the outbound link is declared.

Range: [0.4 * *unreservedRate*, 0.99 * *unreservedRate*]

Default: 0.95 * *unreservedRate*

rateLowThreshold

Rate at or above which congestion on the outbound link is imminent.

Range: [0.5 * *rateHighThreshold*, 0.99 * *rateHighThreshold*]

Default: 0.9 * *rateHighThreshold*

receivedRate

The value of the *fairRate* field carried by the most recently received SCFF.

receivedRi

The value of the *ri* field carried by the most recently received SCFF. This value represents the ringlet on which the SCFF originated.

receivedSa

The value of the *sa* field carried by the most recently received SCFF.

receivedTtl

The value of the *ttl* field carried by the most recently received SCFF after decrement at the receiving station.

reportCoef

A value indicating the number of *advertisingIntervals* that elapse between the sending of successive MCFFs.

Range: [8, 512]

Default: 10

roundUpActiveWeights

A boolean value indicating whether *activeWeights* is rounded up to the nearest power of two.

TRUE—The *activeWeights* value is rounded up.

FALSE—(Otherwise.)

Default: Behaves as though set to FALSE.

starveFactor

The fraction of the *allowedRate* below which a station is said to be starved with respect to the addition of fairness eligible traffic to the ringlet.

Allowed values: $1/(2^{(n-1)})$ or $1 - (1 / (2^{(n-1)}))$ for n in the range [1, 8]

Default: 0.5

stqHighThreshold

A level of STQ occupancy at or above which fairness eligible add frames are no longer admitted.

Defined only for a dual queue implementation.

Range: [3 * *mtuSize*, *stqFullThreshold* - *mtuSize*]

Default: 0.25 * *stqFullThreshold*

stqHighWatermark

The highest level of STQ occupancy since the last reset of this value.

stqLowThreshold

A level of STQ occupancy at or above which congestion on the outbound link is imminent. Defined only for dual queue implementations.

Range: [*mtuSize*, *stqMedThreshold - mtuSize*]

Default: $0.5 * \text{stqHighThreshold}$

stqLowWatermark

The lowest level of STQ occupancy since the last reset of this value.

stqMedThreshold

A level of buffer occupancy in a dual queue deployment, at or above which congestion on the outbound link is declared.

Range: [*stqLowThreshold + mtuSize*, *stqHighThreshold - mtuSize*]

Default: $0.5 * (\text{stqHighThreshold} + \text{stqLowThreshold})$

10.2.3 Common state machine routines

The routines listed in this subclause are referenced by multiple state machines.

AddByteAvailable()

Indicates whether a byte of a locally sourced frame is available for per-byte processing.

TRUE—A byte is available.

FALSE—(Otherwise.)

FeAddByteAvailable()

Indicates whether a byte of a fairness eligible locally sourced frame is available for per-byte processing.

TRUE—A byte is available.

FALSE—(Otherwise.)

FefwByteAvailable()

Indicates whether a byte of a fairness eligible transit frame is available for per-byte processing.

TRUE—A byte is available.

FALSE—(Otherwise.)

FwByteAvailable()

Indicates whether a byte of a transit frame is available for per-byte processing.

TRUE—A byte is available.

FALSE—(Otherwise.)

IsCongested()

Indicates whether or not the station is locally congested.

TRUE—The station is locally congested.

FALSE—(Otherwise.)

```
IsCongested()
{
    if (myDualQueueStation &&
        ((stqDepth > stqLowThreshold) || (nrXmitRate > unreservedRate[myRi])))
        return (TRUE);
    if (!myDualQueueStation &&
        ((lpNrXmitRate > rateLowThreshold) || 
         classBAccessDelayTimerExpired || 
         classCAccessDelayTimerExpired))
        return (TRUE);
    return (FALSE);
}
```

(10.5)

10.2.4 Variables and routines defined in other clauses

This clause references the following variables and routines defined in Clause 7:

CENTER_WRAP
classBAccessDelayTimerExpired
classCAccessDelayTimerExpired
currentTime
Min()
Dequeue()
DequeueRi()
Enqueue()
EntryInQueue()
frame
FROM_EDGE
INTO_EDGE
lineRate
Max()
myDualQueueStation
myMacAddress
myRi
myProtectMethod
Other()
passAddFe
passAddFeCongested
passAddFeOrSfq
Q_TX_STQ
sendD
stqDepth
stqFullThreshold

This clause references the following variables, routines, and literals defined in Clause 9:

CLASS_A0

This clause references the following variables, routines, and literals defined in Clause 11:

CLOSED_RING
FROM_EDGE
INTO_EDGE
MAX_STATIONS
OPEN_RING
mtuSize
myTopoInfo.conservativeMode
myTopoInfo.protConfig
numStations
ringInfo.multichokeUser
ringInfo.topoType
ringInfo.totalHopsRx[ri]
ringInfo.totalHopsTx[ri]
ringInfo.unreservedRate[ri]
topoEntry.conservativeMode[ri][hops]
topoEntry.lrtt[ri][hops]

NOTE—Clause 11 uses the variable *ri* to represent the associated ringlet and *hops* to represent the number of hops between the local station and the edge station on the ringlet *ri*. The higher level structures *myTopoInfo*, *ringInfo*, and *topoEntry* are omitted when there is no ambiguity. In the case of *myTopoInfo.conservativeMode* and *topoEntry.conservativeMode[ri][hops]*, the ambiguity is resolved by the array indices, allowing references to *conservativeMode* and *conservativeMode[ri][hops]*.

10.3 Frame formats

10.3.1 Fairness frame format

The fairness frame format (see 9.4) payload contains *fairnessHeader* and *fairRate* values, as illustrated by Figure 10.15.

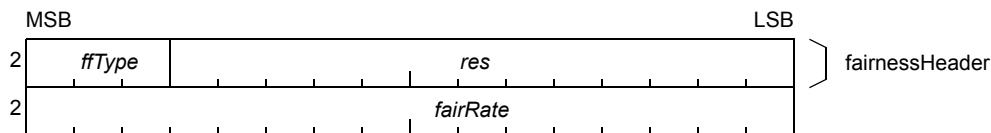


Figure 10.15—Fairness frame payload

The *ttl* (time to live) field is set to the value MAX_STATIONS by an originating station. Each subsequent station in a congestion domain sets the *ttl* to one less than the *ttl* of the last received SCFF. This allows a receiving station to compute the number of hops to the originating station as MAX_STATIONS - *frame.ttl*. The value of the *ri* field in an SCFF is specified by Table 10.10. The value of the *ri* field in an MCFF is Other(*myRi*). A station is the origin of a fairness frame if it has placed *myMacAddress* in the *frame.saCompact* field of the transmitted frame.

10.3.1.1 *ffType*: A 3-bit *ffType* field that identifies fairness frame types, as specified by Table 10.3.

Table 10.3—Fairness frame type (*ffType*) values

Value	Name	Usage
000_2	SINGLE_CHOKE	Advertises the <i>fairRate</i> of a station to the upstream neighbor, once per <i>advertisingInterval</i> .
001_2	MULTI_CHOKE	Reports the <i>normLocalFairRate</i> of a station to all other stations on the ringlet, once per <i>reportingInterval</i> .
$010_2 - 111_2$	reserved	For future use.

10.3.1.2 *res*: A 13-bit (reserved) field is ignored on receipt and set to zero on transmit.

10.3.1.3 *fairRate*: The 16-bit field that carries a normalized rate encoded as a 16-bit quantity. A value of FULL_RATE (FFFF₁₆) indicates the full line rate.

10.3.2 Fairness differential delay (FDD) frame format

Fairness differential delay (FDD) frames are control frames identified by a *controlType* equal to CT_FDD. The FDD frame contents are illustrated in Figure 10.16.

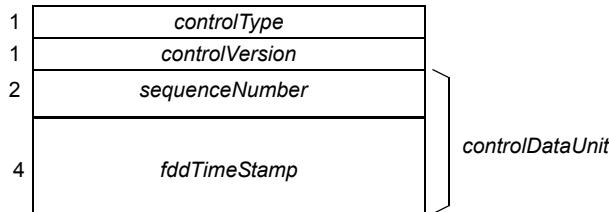


Figure 10.16—Fairness differential delay (FDD) payload

The header fields (specified in 9.4, but not illustrated here) are additionally constrained as specified in Table 10.4.

Table 10.4—Fairness differential delay frame *baseRingControl* subfield values

Field	<i>stationProtectionConfig</i>	classA FDD	classC FDD
<i>fe</i>	—	FALSE	TRUE
<i>sc</i>	—	CLASS_A0	CLASS_C
<i>we</i>	STEERING	0	
	WRAPPING	1	

The value of *hopsToCongestion* is a preferred initial value for the *tll* (time to live) field, but the use of this value is not required.

10.3.2.1 *sequenceNumber*: The 16-bit field that identifies a pair of FDD frames, one of classA and the other of classC, used to compute the FDD at the destination station.

10.3.2.2 *fddTimeStamp*: A 32-bit field provided for a station to mark the time at which it sent an FDD frame. If a timestamp is placed herein, that value is specified in microseconds. If a station is unable to timestamp the frame, this field shall be cleared to zero.

FDD frames are sent in pairs, with one frame in the pair having an *sc* value of CLASS_A0 and an *fe* value of FALSE, and the other having an *sc* value of CLASS_C and an *fe* value of TRUE. An FDD frame is invalid if it is received and its associated FDD frame of the other service class is not received.

10.4 Fairness state machines

This clause specifies the following state machines:

- PerByte
- PerAgingInterval
- AggressiveRateAdjust

ConservativeRateAdjust
 PerAdvertisingInterval
 ActiveWeightsComputation
 PerReportingInterval
 FairnessFrameReceive
 FddFrameTransmit
 FrttComputation

The fairness state machines are specified by the state tables of this subclause. All state tables shall be deployed except that:

- Either Table 10.8 or Table 10.9 is deployed.
- If Table 10.8 is deployed, then Table 10.12 and Table 10.15 are not deployed.
- If Table 10.9 is deployed, then Table 10.12 is optionally deployed and Table 10.15 is deployed.

In the case of any ambiguity between the text and the state tables, the state tables shall take precedence. The notation used in the state table is described in 3.4.

10.4.1 PerByte state machine

The activities specified by the PerByte state machine are performed for each byte of data that transits the station or becomes available for addition to the ringlet. These activities include the increment of rate counters (see 10.1.3.9) and the setting of policing indications (see 10.1.3.10). It is permissible to delay the processing of bytes by the PerByte state machine, but no more than 256 bytes shall be processed at one time and processing must be performed on transmission of the final byte of a frame.

NOTE—Transmitting a frame outbound from the stage queue triggers PerByte state machine processing. Either AddByteAvailable() or FwByteAvailable() will return TRUE. The rules for triggering PerByte state machine processing are the same as those for triggering debit of the fairness eligible shapers (see 7.5.1).

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

10.4.1.1 PerByte state machine definitions

CLASS_A0
 See 10.2.4.
 Q_TX_STQ
 See 10.2.4.

10.4.1.2 PerByte state machine variables

activeWeightsDetection
 See 10.2.2.
addRate
addRateCongested
addRateCongestedOK
addRateOK
 See 10.2.2.
admissionMethod
 An enumerated value used to specify the traffic admission method:
 RATE_BASED—Admission is based on rate policing.
 SHAPER_BASED—Admission is based on rate shaping.
allowedRate

allowedRateCongested

See 10.2.2.

conservativeMode

See 10.2.4.

forwardedFeFrame

See 10.2.2.

frame

The contents of an RPR frame.

fwRate

fwRateCongested

nrXmitRate

See 10.2.2.

passAddFe

passAddFeCongested

passAddFeOrStq

See 10.2.4.

ringInfo.unreservedRate[ri]

See 10.2.4.

sendD

See 10.2.4.

stqDepth

See 10.2.4.

stqHighThreshold

See 10.2.2.

10.4.1.3 PerByte state machine routines

AddByteAvailable()

See 10.2.3.

EntryInQueue()

See 10.2.4.

FeAddByteAvailable()

FeFwByteAvailable()

FwByteAvailable()

See 10.2.3.

10.4.1.4 PerByte state table

Table 10.5—PerByte

Current state		Row	Next state	
state	condition		action	state
START	—	1	addRate = 0; addRateCongested = 0; fwRate = 0; fwRateCongested = 0; nrXmitRate = 0; addRateOK = TRUE; addRateCongestedOK = TRUE;	TEST
TEST	FeAddByteAvailable() && frame.ttl > hopsToCongestion	2	addRate += 1; addRateCongested += 1;	THRU
	FeAddByteAvailable()	3	addRate += 1;	
	FeFwByteAvailable() && frame.ttl > hopsToCongestion	4	fwRate += 1; fwRateCongested += 1;	FEFW
	FeFwByteAvailable()	5	fwRate += 1;	
	AddByteAvailable()	6	—	THRU
	FwByteAvailable()	7	—	
	—	8	—	SETS
FEFW	activeWeightsDetection && conservativeMode;	9	forwardedFeFrame[(frame.ttlBase - frame.ttl) + 1] = TRUE;	THRU
	—	10	—	
THRU	AddByteAvailable() && frame.sc != CLASS_A0	11	nrXmitRate += 1;	SETS
	FwByteAvailable() && frame.sc != CLASS_A0	12	—	
	—	13	—	
SETS	admissionMethod == RATE_BASED	14	addRateOK = addRate < allowedRate && nrXmitRate < unreservedRate[myRi] && (!EntryInQueue(Q_TX_STQ) (fwRate > addRate && stqDepth < stqHighThreshold)); addRateCongestedOK = addRateOK && (addRateCongested < allowedRateCongested);	TEST
	// admissionMethod == SHAPER_BASED	15	addRateOK = passAddFe && sendD && passAddFeOrStq; addRateCongestedOK = addRateOK && passAddFeCongested;	

Row 10.5-1: Initialize rate counters and policing indicators.

Row 10.5-2: A byte of fairness eligible add traffic bound for a destination beyond the congestion point is counted in both the *addRate* and the *addRateCongested*.

Row 10.5-3: A byte of fairness eligible add traffic bound for a destination short of the congestion point is counted in the *addRate* but not the *addRateCongested*.

Row 10.5-4: A byte of fairness eligible transit traffic bound for a destination beyond the congestion point is counted in both the *fwRate* and the *fwRateCongested*.

Row 10.5-5: A byte of fairness eligible transit traffic bound for a destination short of the congestion point is counted in the *fwRate* but not the *fwRateCongested*.

Row 10.5-6: A byte of added traffic is not fairness eligible and results in no increment to the rate counters associated with fairness eligible traffic.

Row 10.5-7: A byte of transited traffic is not fairness eligible and results in no increment to the rate counters associated with fairness eligible traffic.

Row 10.5-8: Policing parameters are adjusted without regard to the availability of traffic.

Row 10.5-9: *ActiveWeightsDetection* is deployed, and the station uses the conservative rate computation method. The source station identified by the fairness eligible transit frame is marked for inclusion in the next *activeWeights* computation (see Table 10.12).

NOTE—The detection of active weights can be done either once per frame or on a per-byte basis; however, per-byte-based implementation may mark a station active in two consecutive *activeWeightsIntervals* as the result of a single transmitted frame, which is acceptable.

Row 10.5-10: The station does not mark the source of the transited frame for inclusion in the *activeWeights* computation.

Row 10.5-11: A byte of unreserved (i.e., not subclassA0) add traffic is included in the *nrXmitRate*.

Row 10.5-12: A byte of unreserved (i.e., not subclassA0) transit traffic is included in the *nrXmitRate*.

Row 10.5-13: A byte of reserved (i.e., subclassA0) traffic is not counted in the *nrXmitRate*.

Row 10.5-14: The policing parameters are computed for the rate-based admission method as specified by 10.1.3.10. Some variables and routines referenced by this computation, such as *EntryInQueue* (see 7.2.3) and *stqDepth* (see 7.2.2), are defined within Clause 7. Clause 7, in turn, uses the computed policing parameters in admitting traffic.

Row 10.5-15: The policing parameters are computed for the shaper-based admission method (see 7.5.7 for a description of the fairness eligible shaper). Shaping parameters referenced by this computation, including *passAddFe* (see 7.5.7.2.2), *sendD* (see 7.2.1), *passAddFeOrStq* (see 7.5.7.3.2), and *passAddFeCongested* (see 7.5.7.4.2) are defined by the fairness eligible shaper (see 7.5.7). Clause 7, in turn, uses the computed policing parameters in admitting traffic.

10.4.2 PerAgingInterval state machine

The PerAgingInterval state machine performs low-pass filtering (see 10.1.3.9) of rate counters, normalization (see 10.1.3.8) of smoothed rates, rate adjustment specific to the mode (i.e., aggressive or conservative) of the station, computation of the *allowedRateCongested*, optional sending of an MA_CONTROL.indication (see Table 6.7) to the client, aging of rate counters (see 10.1.3.9), and setting of high and low watermarks of the STQ.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

NOTE—The actions described by Table 10.6 are independent of the method of rate computation (i.e., aggressive or conservative) of the station.

10.4.2.1 PerAgingInterval state machine definitions

There are no definitions associated with this state machine.

10.4.2.2 PerAgingInterval state machine variables

addRate
addRateCongested
 See 10.2.2.
ageCoef
 See 10.2.2.
agingStartTime
 Records the start of an *agingInterval*.
currentTime
 See 10.2.4.
frame
 The contents of an RPR frame.
fwRate
fwRateCongested
 See 10.2.2.
lineRate
 See 10.2.4.
localWeight
 See 10.2.2.
lpAddRate
 See 10.2.2.
lpCoef
 See 10.2.2.
lpFwRate
lpFwRateCongested
lpNrXmitRate
 See 10.2.2.
normCoef
 See 10.2.2.
normLpFwRate
normLpFwRateCongested
nrXmitRate
 See 10.2.2.
rateCoef
 See 10.2.2.
stqDepth
 See 10.2.4.
stqHighWatermark
stqLowWatermark
 See 10.2.2.

10.4.2.3 PerAgingInterval state machine routines

AgeRates()

Age rate values.

```
AgeRates()
{
    addRate -= addRate / ageCoef;
    addRateCongested -= addRateCongested / ageCoef;
    fwRate -= fwRate / ageCoef;
    fwRateCongested -= fwRateCongested / ageCoef;
    nrXmitRate -= nrXmitRate / ageCoef;
}
```

(10.6)

InitLpFilteredRates()

Initializes the values of all low-pass filtered rates to zero.

```
InitLpFilteredRates()
{
    lpAddRate = 0;
    lpFwRate = 0;
    lpFwRateCongested = 0;
    lpNrXmitRate = 0;
}
```

(10.7)

InitNormalizedRates()

Initializes the values of all normalized low-pass filtered rates to zero.

```
InitNormalizedRates()
{
    normLpFwRate = 0;
    normLpFwRateCongested = 0;
}
```

(10.8)

LowPassFilterRates()

Compute the smoothed values of aged rates.

```
LowPassFilterRates()
{
    lpAddRate += (addRate - lpAddRate) / lpCoef;
    lpfwRate += (fwRate - lpFwRate) / lpCoef;
    lpfwRateCongested += (fwRateCongested - lpFwRateCongested) / lpCoef;
    lpNrXmitRate += (nrXmitRate - lpNrXmitRate) / lpCoef;
}
```

(10.9)

NOTE—The calculations of Equation (10.9) require signed arithmetic for correct operation. These equations can be rephrased to only use unsigned arithmetic. For example: lpfwRate = ((lpCoef - 1) * lpfwRate + fwRate) / lpCoef.

NormalizeLowPassFilteredRates()

Normalize low-pass filtered rates.

```
NormalizeLowPassFilteredRates()
{
    normLpFwRate = lpfwRate / normCoef;
    normLpFwRateCongested = lpFwRateCongested / normCoef;
}
```

(10.10)

RateAdjust(conservativeMode)

Invoke the ConservativeRateAdjust state machine (Table 10.9) when the station uses the conservative method of rate adjustment or the AggressiveRateAdjust state machine (Table 10.8) when the station uses the aggressive method of rate adjustment.

```
RateAdjust()
{
    if (myTopoInfo.conservativeMode)
        ConservativeRateAdjust;
    else
        AggressiveRateAdjust;
}
```

(10.11)
RecordStqWatermarks()

Update the STQ occupancy high and low watermark values.

```
RecordStqWatermarks()
{
    stqHighWatermark = Max(stqHighWatermark, stqDepth);
    stqLowWatermark = Min(stqLowWatermark, stqDepth);
}
```

(10.12)
ResetStqWatermarks()

Reset the *stqHighWatermark* and *stqLowWatermark* values to *stqDepth*. This routine is invoked by the MLME in response to an operator request to reset the watermarks. See the description of *rprFairnessResetWaterMarks* in D.6.

SetAllowedRateCongested()

Computes the rate at which fairness eligible traffic transiting the congestion point can be added to the ringlet. In the presence of downstream congestion, the rate is based on the most recently received rate advertisement. In the absence of downstream congestion, the rate is ramped.

```
SetAllowedRateCongested()
{
    if (receivedRate != FULL_RATE)
        allowedRateCongested = Min(maxAllowedRate, receivedRate * normCoef);
    else
        allowedRateCongested +=
            (maxAllowedRate - allowedRateCongested) / rampUpCoef;
}
```

(10.13)
SingleChokeInd()

Transfer to the MAC client an MA_CONTROL.indication having an opcode of SINGLE_CHOKE_IND (see Table 6.7).

10.4.2.4 PerAgingInterval state table**Table 10.6—PerAgingInterval**

Current state		Row	Next state	
state	condition		action	state
START	conservativeMode	1	localFairRate = 0; allowedRate = 0;	INIT
	—	2	allowedRate = maxAllowedRate;	
INIT	—	3	allowedRateCongested = 0; hopsToCongestion = MAX_STATIONS; localCongested = FALSE; InitLPFilteredRates(); InitNormalizedRates(); agingStartTime = currentTime;	WAIT
WAIT	(currentTime - agingStartTime) >= agingInterval	4	LowPassFilterRates(); NormalizeLowPassFilteredRates(); RateAdjust(conservativeMode); SetAllowedRateCongested (receivedRate); SingleChokeInd(); AgeRates(); RecordStqWatermarks(); agingStartTime = currentTime;	WAIT
	—	5	—	

Row 10.6-1: Perform initialization specific to conservative rate adjustment.

Row 10.6-2: Perform initialization specific to aggressive rate adjustment. The *allowedRate* maintains its initial value of *maxAllowedRate* (see 10.1.3.12). When using the aggressive method, traffic not transiting a congestion point is restricted only by the *maxAllowedRate*.

Row 10.6-3: The *allowedRateCongested* and the *hopsToCongestion* should be initialized to the values specified in the table. Initialize congestion state variable. Initialize the low-pass filtered and normalized rates.

Row 10.6-4: Perform low-pass filtering (see 10.1.3.9) of rate counters, perform normalization (see 10.1.3.8) of smoothed rates, invoke rate adjustment specific to the mode (i.e., aggressive or conservative) of the station, compute the value of the *allowedRateCongested*, optionally send an MA_CONTROL.indication (see Table 6.7) to the client, age rate counters (see 10.1.3.9), and set high and low STQ watermarks as appropriate. In the absence of downstream congestion, the *allowedRateCongested* is ramped-up toward the *maxAllowedRate*. In the presence of downstream congestion, the *allowedRateCongested* is set to the weight-adjusted value of the *receivedRate* but is not allowed to exceed the *maxAllowedRate*.

Row 10.6-5: Remain in the current state if the *agingInterval* has not expired. In the presence of downstream congestion, the *allowedRateCongested* is assigned the weight-adjusted *fairRate* of the head. In the absence of downstream congestion, the *allowedRateCongested* is ramped up (see 10.1.3.13).

10.4.3 AggressiveRateAdjust state machine

The AggressiveRateAdjust state machine (see Table 10.8) is invoked by the PerAgingInterval state machine by a station using the aggressive method of rate computation. The activities performed by the AggressiveRateAdjust state machine include the computation of the *localFairRate* (see 10.1.3.11) and normalization of the *localFairRate* (see 10.1.3.8). The *allowedRate* maintains its initial value of *maxAllowedRate* (see 10.1.3.12). When using the aggressive method, traffic not transiting a congestion point

is restricted only by the *maxAllowedRate*.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

10.4.3.1 AggressiveRateAdjust state machine definitions

There are no definitions associated with this state machine.

10.4.3.2 AggressiveRateAdjust state machine variables

allowedRate

See 10.2.2.

localCongested

See 10.2.2.

localFairRate

See 10.2.2.

lpAddRate

See 10.2.2.

maxAllowedRate

See 10.2.2.

normCoef

See 10.2.2.

normLocalFairRate

See 10.2.2.

receivedRate

See 10.2.2.

10.4.3.3 AggressiveRateAdjust state machine routines

IsCongested()

See 10.2.3.

10.4.3.4 AggressiveRateAdjust state table

Table 10.8—AggressiveRateAdjust

Current state		Row	Next state	
state	condition		action	state
START	IsCongested()	1	localCongested = TRUE; localFairRate = lpAddRate; normLocalFairRate = localFairRate / normCoef;	RETURN
	—	2	localCongested = FALSE;	

Row 10.8-1: The *localFairRate* of a congested station is set to the smoothed fairness eligible add rate of the station.

Row 10.8-2: The *localFairRate* of an uncongested station is unchanged.

NOTE—The actions described by Table 10.9 are performed only when the aggressive method of rate computation is used by the station.

10.4.4 ConservativeRateAdjust state machine

The ConservativeRateAdjust state machine (see Table 10.9) is invoked by the PerAgingInterval state machine in a station using the aggressive method of rate computation. The activities performed by the ConservativeRateAdjust state machine include the computation of the *localFairRate* (see 10.1.3.11) and the normalization of the *localFairRate* (see 10.1.3.8).

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

10.4.4.1 ConservativeRateAdjust state machine definitions

There are no definitions associated with this state machine.

10.4.4.2 ConservativeRateAdjust state machine variables

activeWeights

activeWeightsDetection

See 10.2.2.

allowedRate

See 10.2.2.

currentTime

See 10.2.4.

frtt

See 10.2.2.

frttStartTime

Records the value of *currentTime*, in microseconds, in order to determine, at a later time, whether an *frtt* has elapsed.

localCongested

See 10.2.2.

localFairRate

See 10.2.2.

localWeight

See 10.2.2.

lpAddRate

lpFwRate

See 10.2.2.

maxAllowedRate

See 10.2.2.

myDualQueueStation

See 10.2.4.

normCoef

See 10.2.2.

normLocalFairRate

See 10.2.2.

nrXmitRate

See 10.2.2.

rampDnCoef

rampUpCoef

See 10.2.2.

rateHighThreshold

rateLowThreshold

See 10.2.2.

starveFactor

See 10.2.2.

stqDepth

See 10.2.4.

*stqLowThreshold**stqMedThreshold**stqHighThreshold*

See 10.2.2.

unreservedRate[ri];

See 10.2.4.

*upperBound*The highest value that the *localFairRate* can assume when severe congestion occurs.

10.4.4.3 ConservativeRateAdjust state machine routines

IsCongested()

See 10.2.3.

Min()

See 10.2.4.

10.4.4.4 ConservativeRateAdjust state table

The conservative rate adjustment state machine is specified by Table 10.9.

Table 10.9—ConservativeRateAdjust

Current state		Row	Next state	
state	condition		action	state
START	localCongested	1	—	CGST
	—	2	—	UNCG
UNCG	IsCongested()	3	if (activeWeightsDetection) localFairRate = (unreservedRate[myRi] / activeWeights) * localWeight; else localFairRate = lpAddRate; localCongested = TRUE; frttStartTime = currentTime;	FINAL
	—	4	allowedRate += (maxAllowedRate - allowedRate) / rampUpCoef;	RETURN
CGST	localFairRate/localWeight >= unreservedRate[myRi] - (unreservedRate[myRi] / rampUpCoef)	5	localCongested = FALSE; allowedRate += (maxAllowedRate - allowedRate) / rampUpCoef;	RETURN
	(currentTime - frttStartTime) >= frtt	6	frttStartTime = currentTime;	AFTER
	—	7	—	EARLY

Table 10.9—ConservativeRateAdjust (continued)

Current state		Row	Next state	
state	condition		action	state
AFTER	myDualQueueStation && stqDepth > stqMedThreshold	8	localFairRate -= localFairRate / rampDnCoef;	FINAL
	!myDualQueueStation && (addRate + fwRate > rateHighThreshold)	9		
	myDualQueueStation && stqDepth < stqLowThreshold	10	localFairRate += ((localWeight * unreservedRate[myRi]) - nrXmitRate) / rampUpCoef;	
	!myDualQueueStation && (addRate + fwRate < rateLowThreshold)	11		
	—	12	—	EARLY
EARLY	myDualQueueStation && stqDepth > stqHighThreshold && activeWeightsDetection && (lpAddRate / localWeight < (lpFwRate / (activeWeights - localWeight)))	13	upperBound = (lpAddRate + lpFwRate) * (localWeight / activeWeights); localFairRate = Min(localFairRate, upperBound);	FINAL
	myDualQueueStation && stqDepth > stqHighThreshold && !activeWeightsDetection && lpAddRate < starveFactor * allowedRate	14	localFairRate = Min(localFairRate, lpAddRate);	
	—	15	—	
FINAL	—	16	normLocalFairRate = localFairRate / normCoef; allowedRate = Min(unreservedRate[myRi], localFairRate);	RETURN

Row 10.9-1: Reestablish the congested state based on the final value of *localCongested* at the last invocation of this machine.

Row 10.9-2: Reestablish the uncongested state based on the final value of *localCongested* at the last invocation of this machine.

Row 10.9-3: An uncongested station becomes congested (see 10.1.3.2). A station performing *activeWeightsDetection* sets the *localFairRate* to a fair share of the *unreservedRate*. A station not performing *activeWeightsDetection* sets the *localFairRate* to the smoothed fairness eligible add rate of the station.

Row 10.9-4: An uncongested station remains uncongested. The *localFairRate* is unchanged, but the *allowedRate* is ramped-up toward the *maxAllowedRate*.

Row 10.9-5: Hysteresis is applied to the transition between congested and uncongested states by requiring that the *fairRate* increase to a value close to the *unreservedRate* before the transition to the uncongested state is made. The *fairRate* approaches the *unreservedRate* asymptotically with a granularity of *rampUpCoef*. For this reason, the *fairRate* is considered to have reached the *unreservedRate* when it is within one *rampUpCoef* of that rate. On transition to the uncongested state, the *localFairRate* is unchanged, but the *allowedRate* is ramped-up toward the *maxAllowedRate*, as in the case of an uncongested station remaining in the uncongested state (see Row 10.9-4).

Row 10.9-6: A congested station remains congested. If the *localFairRate* has not been adjusted for a period of one FRTT or more (i.e., the FRTT timer has expired), the *localFairRate* is updated (see Rows 10.9-8–10.9-11), and the FRTT timer is reset.

Row 10.9-7: A congested station remains congested. If the *localFairRate* has been adjusted within a period of one FRTT (i.e., the FRTT timer has not expired), the *localFairRate* may, in some cases, be updated (see Rows 10.9-8–10.9-11).

Row 10.9-8: STQ occupancy, in the case of a dual queue MAC, grows beyond a specified threshold value and an FRTT has elapsed since the last change in the value of the *localFairRate*. The *localFairRate* is ramped down to reduce congestion but is not reduced below the station's fair share of the *unreservedRate*.

Row 10.9-9: The combined add and forward rate of fairness eligible traffic, in the case of a single queue MAC, grows beyond a specified threshold value and an FRTT has elapsed since the last change in the value of the *localFairRate*. The *localFairRate* is ramped down to reduce congestion.

Row 10.9-10: STQ occupancy, in the case of a dual queue MAC, falls below a specified threshold value and an FRTT has elapsed since the last change in the value of the *localFairRate*. The *localFairRate* is ramped up to increase throughput.

Row 10.9-11: The combined add and forward rate of fairness eligible traffic, in the case of a single queue MAC, falls below a specified threshold value and an FRTT has elapsed since the last change in the value of the *localFairRate*. The *localFairRate* is ramped up to increase throughput.

Row 10.9-12: None of the conditions associated with FRTT expiration (Rows 10.9-8–10.9-11) are met. Test conditions associated with severe congestion.

Row 10.9-13: A dual queue station performing *activeWeightsDetection* is experiencing severe congestion. Severe congestion is indicated when the high threshold of the STQ is exceeded and the station is adding less than its fair share of traffic. In this case, the station recomputes the *localFairRate* as a fair share of the sum of transited and added traffic, provided this fair share is less than the current *localFairRate*. When comparing the weight-adjusted *addRate* to the weight-adjusted *fwRate*, the *fwRate* is scaled by the expression *activeWeights - localWeight*, reflecting that the local station is not a contributor to the measured *fwRate* but its weight is included in the *activeWeights*.

Row 10.9-14: A dual queue station not performing *activeWeightsDetection* is experiencing severe congestion. Severe congestion is indicated when the station is adding less than a specified fraction (*starveFactor*) of the *allowedRate*. In this case, the station sets the *localFairRate* to the smoothed fairness eligible add rate.

Row 10.9-15: The default case in which the FRTT timer has not expired and severe congestion has not been detected. The *localFairRate* is not changed.

Row 10.9-16: Actions within the table have modified the value of *localFairRate*. The value of *normLocalFairRate* is recomputed, and the *allowedRate* is set to the value of the *localFairRate*, except where the *localFairRate* exceeds the *unreservedRate* of the local ringlet, in which case the *allowedRate* is set to the *unreservedRate*.

NOTE 1—The actions described by Table 10.9 are performed only when the conservative method of rate computation is used by the station.

NOTE 2—The variable *localCongested* reflects the current congestion state of the station. Its value need only be changed in the event of a transition in the congestion state.

10.4.5 PerAdvertisingInterval state machine

At the expiration of each *advertisingInterval*, the station performs one of the following actions (see 10.1.3.3):

- Advertises its locally computed *fairRate*.
- Propagates the *fairRate* received from the downstream neighbor.
- Advertises the FULL_RATE to the upstream neighbor.

The action reflects the rule that a station advertises the *fairRate* associated with the head of the congestion domain (see 10.1.3.4) in which it lies, unless it is the tail of that domain. A station that does not lie within a

congestion domain, or that is the tail of a congestion domain but not the head of another (i.e., upstream) congestion domain, advertises the FULL_RATE.

NOTE—A station at the tail of one congestion domain and at the head of another propagates its *fairRate* upstream, consistent with the station's role as the head of a congestion domain.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

10.4.5.1 PerAdvertisingInterval state machine constants

CLOSED_RING

See 10.2.4.

MAX_STATIONS

See 10.2.1.

OPEN_RING

See 10.2.4.

FAIRNESS_SIZE

See 10.2.4

FULL_RATE

See 10.2.1.

10.4.5.2 PerAdvertisingInterval state machine variables

advertisingInterval

See 10.2.2.

advertisingStartTime

Records the start of an *advertisingInterval*.

advertisementRatio

See 10.2.2.

downstreamCongested

See 10.2.2.

currentTime

See 10.2.4.

frame

The contents of an RPR frame.

iAmTail

See 10.2.2.

lineRate

See 10.2.4.

localCongested

See 10.2.2.

localWeight

See 10.2.2.

myRi

myMacAddress

See 10.2.4.

normLocalFairRate

normLpFwRateCongested

normLpFwRate

See 10.2.2.

receivedRate

receivedRi

receivedTtl

receivedSa

See 10.2.2.

ringInfo.topoType

See 10.2.4.

10.4.5.3 PerAdvertisingInterval state machine routines

EnqueueRi()

See 10.2.4

Other()

See 10.2.4

10.4.5.4 PerAdvertisingInterval state table

The PerAdvertisingInterval state table is specified by Table 10.10.

Table 10.10—PerAdvertisingInterval

Current state		Row	Next state	
state	condition		action	state
START	—	1	receivedRate = FULL_RATE; receivedRi = Other(myRi); receivedSa = myMacAddress; receivedTtl = MAX_STATIONS; iAmTail = FALSE; localFairRate = 0; advertisingStartTime = currentTime;	FIRST
FIRST	(currentTime - advertisingStartTime) >= advertisingInterval	2	advertisingStartTime = currentTime;	MORE
	—	3	—	FIRST
MORE	localCongested && !downstreamCongested	4	iAmTail = FALSE;	MINE
	localCongested && normLocalFairRate <= receivedRate	5	iAmTail = TRUE;	
	downstreamCongested && !localCongested	6	—	DOWN
	downstreamCongested && normLocalFairRate > receivedRate	7	—	
	— // !downstreamCongested && // !localCongested	8	iAmTail = FALSE;	FULL

Table 10.10—PerAdvertisingInterval (continued)

Current state		Row	Next state	
state	condition		action	state
DOWN	topoType == CLOSED_RING && receivedRate >= localWeight * normLpFwRateCongested	9	iAmTail = TRUE;	FULL
	topoType == OPEN_RING && receivedRi != Other(myRi) && receivedRate >= localWeight * normLpFwRate	10		
	—	11	iAmTail = FALSE;	FORW
MINE	—	12	frame.fairRate = normLocalFairRate; frame.saCompact = myMacAddress; frame.ttl = MAX_STATIONS; frame.ri = Other(myRi);	SEND
FORW	—	13	frame.fairRate = receivedRate; frame.saCompact = receivedSa; frame.ttl = receivedTtl; frame.ri = receivedRi;	SEND
FULL	—	14	frame.fairRate = FULL_RATE; frame.saCompact = myMacAddress; frame.ttl = MAX_STATIONS; frame.ri = Other(myRi);	SEND
SEND	—	15	frame.ffType = SINGLE_CHOKE; EnqueueRi(Other(myRi), Q_TX_FAIR, frame);	FIRST

Row 10.10-1: Prior to the receipt of a rate advertisement, local variables representing the value of fields within the most recently received advertisement are set consistent with the absence of downstream congestion.

Row 10.10-2: An *advertisingInterval* has expired. The *advertisingInterval* is a configured fraction of the LINE_RATE. The machine performs Rows 10.10-4–10.10-8 of the table to determine whether the received *fairRate* is propagated, the FULL-RATE is advertised, or the normalized *localFairRate* is sent.

Row 10.10-3: An *advertisingInterval* has not yet expired. The machine continues to test for the expiration of the *advertisingInterval*.

Row 10.10-4: The station is locally congested, but not downstream congested. The station is a head and not a tail, and advertises its *normLocalFairRate*.

Row 10.10-5: The station is locally congested, and the *normlocalFairRate* is less than or equal to the *receivedRate*. The station advertises its *normLocalFairRate* and is the tail of the congestion domain.

Row 10.10-6: The station is downstream congested but not locally congested. The value of the advertised *fairRate* is determined by Rows 10.10-9 to 10.10-11 of the state machine.

Row 10.10-7: The station is downstream congested, and the *normLocalFairRate* is greater than the *receivedRate*. The value of the advertised *fairRate* is determined by Rows 10.10-9–10.10-11 of the state machine.

Row 10.10-8: The station is neither downstream nor locally congested. The station advertises the FULL_RATE, indicating the absence of congestion, and is not the tail of a congestion domain.

Row 10.10-9: The station is downstream congested on a closed (i.e., unbroken) ring. The *fairRate* received from the downstream neighbor is greater than or equal to the weight-adjusted rate of fairness eligible traffic transiting both the local station and the head of the congestion domain (*normLpFwRateCongested*). In this case, the FULL RATE is advertised and the station is the tail of a congestion domain.

Row 10.10-10: The station is downstream congested on an open (i.e., broken) ring and the received advertisement arrived on the local ringlet rather than the opposing ringlet. The *fairRate* received from the downstream neighbor is greater than or equal to the weight-adjusted rate of fairness eligible traffic transiting the local station (*normLpFwRate*). In this case the FULL_RATE is advertised and the station is the tail of a congestion domain. Normally (i.e., in the case of a closed ring), a station is a tail if the *fairRate* received from the downstream neighbor is greater than or equal to the weight-adjusted rate of fairness eligible traffic transiting both the local station and the head of the congestion domain (*normLpFwRateCongested*). If, however, the ring is wrapped between the head station and the local station, the *normLpFwRateCongested* does not maintain an accurate count of frames transiting both the local station and the head station. The *normLpFwRate* is known to be larger than *normLpFwRateCongested* as it includes all fairness eligible traffic transiting the local station. This value is taken as conservative alternative to the use of the *normLpFwRateCongested*.

Row 10.10-11: The station is downstream congested, but the conditions of Rows 10.10-9 and 10.10-10 are not met. In this case, the *receivedRate* is propagated and the station is not the tail of a congestion domain.

Row 10.10-12: The station sends its *normLocalFairRate*. The advertisement carries the local station MAC address and the identity of the opposing ringlet.

Row 10.10-13: The station propagates the received advertisement.

Row 10.10-14: The station sends the FULL_RATE. The advertisement carries the local station MAC address and the identity of the opposing ringlet (as in the case of Row 10.10-12).

Row 10.10-15: Send the SCFF on the opposing ringlet.

10.4.6 PerReportingInterval state machine

At the expiration of a *reportingInterval*, the local station broadcasts a rate report to all stations on the ringlet. Rate reporting is further described in 10.1.3.7.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

10.4.6.1 PerReportingInterval state machine definitions

MAX_STATIONS

See 10.2.1.

FULL_RATE

See 10.2.1.

10.4.6.2 PerReportingInterval state machine variables

advertisingInterval

See 10.2.2.

currentTime

See 10.2.4.

frame

The contents of an RPR frame.

localCongested

See 10.2.2.

myRi

myMacAddress

See 10.2.4.

normLocalFairRate

See 10.2.2.

reportingStartTime

Records the start of an *advertisingInterval*.

reportingInterval

The time between the sending of MCFFs. The value of the *reportingInterval* is $reportCoef * advertisingIntervals$ (i.e., $reportingInterval = reportCoef * advertisingInterval$).

reportCoef

See 10.2.2.

ringInfo.multichokeUser

See 10.2.4.

10.4.6.3 PerReportingInterval state machine routines*EnqueueRi()*

See 10.2.4

10.4.6.4 PerReportingInterval state table

The PerReportingInterval state table is specified by Table 10.11.

Table 10.11—PerReportingInterval

Current state		Row	Next state	
state	condition		action	state
START	!multichokeUser	1	reportingStartTime = currentTime;	START
	(currentTime - reportingStartTime) >= reportingInterval	2	reportingStartTime = currentTime;	NEXT
	—	3	—	START
NEXT	localCongested	4	frame.fairRate = normLocalFairRate;	FINAL
	—	5	frame.fairRate = FULL_RATE;	
FINAL	—	6	frame.ffType = MULTI_CHOKE; frame.saCompact = myMacAddress; frame.ttl = MAX_STATIONS; EnqueueRi(Other(myRi), Q_TX_FAIR, frame);	START

Row 10.11-1: Multi-choke reporting is suppressed because there are no stations on the ringlet whose clients make use of multi-choke reporting. The report timer is reset and will expire at the end of the next *reportingInterval*.

Row 10.11-2: There are stations on the ringlet whose clients make use of multi-choke reporting, and the *reportingInterval* has expired. The reporting timer is reset, and multi-choke reporting is performed.

Row 10.11-3: The *reportingInterval* has not yet expired.

Row 10.11-4: A locally congested station reports its *normLocalFairRate* to all stations on the ringlet.

Row 10.11-5: An uncongested station reports the *FULL_RATE* to all stations on the ringlet.

Row 10.11-6: An MCFF is sent.

10.4.7 ActiveWeightsComputation state machine

The ActiveWeightsComputation state machine is applicable only when using the conservative rate adjustment method and when *activeWeightsDetection* is TRUE.

At the expiration of an *activeWeightsInterval*, the local station computes the sum of the weights of stations from which a fairness eligible frame has been received during the past *activeWeightsInterval*. The *activeWeightsInterval* is configured as an integer multiple of the *agingInterval*. The *activeWeights* value is referenced by the PerAgingInterval state machine.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

10.4.7.1 ActiveWeightsComputation state machine definitions

The ActiveWeightsComputation state machine contains no definitions.

10.4.7.2 ActiveWeightsComputation state machine variables

activeWeights

activeWeightsDetection

See 10.2.2.

activeWeightsInterval

The time between successive computations of *activeWeights*. Applicable only in the case of conservative rate adjustment and *activeWeightsDetection*. The *activeWeightsInterval* is an integer multiple of the *advertisingInterval* (i.e., $\text{activeWeightsInterval} = \text{activeWeightsCoef} * \text{agingInterval}$).

activeWeightsTime

Records the start of an *activeWeightsInterval*.

currentTime

See 10.2.4.

10.4.7.3 ActiveWeightsComputation state machine routines

ComputeActiveWeights()

Sums the weights of stations from which fairness eligible frames have been received during the most recent *activeWeightsInterval*. The local station weight is always included in this sum. The deployment may round the *activeWeights* to the next higher power of two if the value is not already a power of two.

```
ComputeActiveWeights() (10.14)
{
    activeWeights = localWeight;
    for (station = 1; station < numStations; station += 1)
    {
        if (forwardedFeFrame[station])
            activeWeights += weights[station];
        forwardedFeFrame[station] = FALSE;
    }
    if (roundUpActiveWeights)
        RoundUpToPowerOfTwo(activeWeights);
}
```

RoundUpToPowerOfTwo()

Returns the argument rounded up to the next higher power of two.

10.4.7.4 ActiveWeightsComputation state table

The ActiveWeightsComputation state table is specified by Table 10.12.

Table 10.12—ActiveWeightsComputation

Current state		Row	Next state	
state	condition		action	state
START	—	1	activeWeights = 1; activeWeightsTime = currentTime;	WAIT
WAIT	(currentTime - activeWeightsTime) >= activeWeightsInterval	2	computeActiveWeights(); activeWeightsTime = currentTime;	WAIT
	—	3	—	

Row 10.12-1: The value of *activeWeights* is initialized to provide a value until an *activeWeightsInterval* has expired and *activeWeights* is computed.

Row 10.12-2: An *activeWeightsInterval* has expired. The *activeWeightsInterval* is a configured multiple of the *agingInterval*. The value of *activeWeights* is computed.

Row 10.12-3: The *activeWeightsInterval* has not yet expired.

10.4.8 FairnessFrameReceive state machine

This table describes the processing of a received SCFF (rate advertisement) or a received MCFF (rate report). The information contained in an advertisement is saved for processing at the expiration of the next *advertisingInterval*. The multi-choke information associated with a rate report is optionally transferred to the MAC client.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

10.4.8.1 FairnessFrameReceive state machine definitions

FULL_RATE

See 10.2.1.

MAX_STATIONS

See 10.2.1.

10.4.8.2 FairnessFrameReceive state machine variables

downstreamCongested

See 10.2.2.

frame

The contents of an RPR frame.

hopsToCongestion

See 10.2.2.

hops

See 10.2.2.

myRi

myMacAddress

myProtectMethod

See 10.2.4.

normLocalFairRate

See 10.2.2.

*receivedRate**receivedRi**receivedTtl**receivedSa*

See 10.2.2.

topoType

See 10.2.4.

10.4.8.3 FairnessFrameReceive state machine routines

DequeueRi()

See 10.2.4

MultiChokeInd()

Transfer to the MAC client via an MA_CONTROL.indication having an opcode of MULTI_CHOKE_IND (see Table 6.7).

Other()

See 10.2.4

10.4.8.4 FairnessFrameReceive state table

The FairnessFrameReceive state table is specified by Table 10.13.

Table 10.13—FairnessFrameReceive state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = DequeueRi(Other(myRi), Q_RX_FAIR)) != NULL && frame.ft == FT_FAIRNESS && frame.ttl != 0	1	—	FIRST
	—	2	—	START
FIRST	frame.ffType == MULTI_CHOKE	3	—	MCFF
	frame.ri == Other(myRi)	4	receivedTtl = frame.ttl - 1;	NEXT
	—	5	receivedTtl = frame.ttl;	
NEXT	—	6	receivedRate = frame.fairRate; receivedSa = frame.saCompact; receivedRi = frame.ri;	SCFF

Table 10.13—FairnessFrameReceive state table (continued)

Current state		Row	Next state	
state	condition		action	state
SCFF	frame.saCompact == myMacAddress && frame.ri == Other(myRi)	7	receivedRate = FULL_RATE; hopsToCongestion = MAX_STATIONS; downstreamCongested = FALSE;	START
	frame.saCompact == myMacAddress && myEdgeState == INTO_EDGE && myProtectMethod == CENTER_WRAP	8		
	frame.saCompact == myMacAddress && myEdgeState == FROM_EDGE && myProtectMethod == CENTER_WRAP	9		
	frame.fairRate != FULL_RATE && topoType == OPEN_RING && frame.ri != Other(myRi)	10	downstreamCongested = TRUE; hopsToCongestion = totalHopsTx[Other(myRi)];	
	frame.fairRate != FULL_RATE	11	downstreamCongested = TRUE; hopsToCongestion = MAX_STATIONS - receivedTtl;	
	—	12	downstreamCongested = FALSE;	
MCFF	frame.saCompact == myMacAddress && topoType == OPEN_RING && myProtectMethod == CENTER_WRAP	13	—	START
	—	14	MultiChokeInd();	

Row 10.13-1: A fairness frame with an unexpired *ttl* is dequeued for processing.

Row 10.13-2: A fairness frame is not available for processing.

Row 10.13-3: The frame is identified for processing as an MCFF.

Row 10.13-4: The *ttl* value carried by an SCFF is decremented if the received SCFF did not wrap between the receiving station and the origin.

Row 10.13-5: The *ttl* value carried by an SCFF is not decremented if the received SCFF wrapped between the receiving station and the origin. In this case, the received *ttl* is not used in the computation of *hopsToCongestion* (see 10.1.3.14).

Row 10.13-6: Fields of the received SCFF are saved for use at the expiration of subsequent *agingIntervals* and *advertisingIntervals*.

Row 10.13-7: The received SCFF is originated by the local fairness instance as indicated by receipt of the fairness frame on the opposing ringlet, i.e., *Other(myRi)*, with source MAC address equal to the local station MAC address. The *receivedRate* is set to the FULL_RATE value, without regard to the *fairRate* value carried by the SCFF. Thus, a fairness instance cannot be the tail of a congestion domain of which it is also the head.

Row 10.13-8: The received SCFF is originated by the local fairness instance as indicated by a receipt of the fairness frame on the local ringlet (i.e., *myRi*) with source MAC address equal to the local station MAC address. The local station performs center-wrap due to a span failure on the receive side of the datapath. In this case, the originating station sends the SCFF on the local ringlet due to the span failure that prevents it from transmitting on the opposing ringlet. The *receivedRate* is set to the FULL_RATE value (see Row 10.13-7).

Row 10.13-9: The received SCFF is originated by the local fairness instance, as in the case of Row 10.13-8, but the span failure is on the transmit side of the datapath. The *receivedRate* is set to the FULL_RATE value (see Row 10.13-7). In this case, the local station receives the SCFF on the local ringlet due to the span

failure that prevents it from receiving on the opposing ringlet. This case is illustrated by the rightmost station in Figure 10.3, which performs center wrap and whose transmit side of the datapath is inactive.

Row 10.13-10: An SCFF indicating downstream congestion is wrapped at some point between the station of origin and the local station. The value of *hopsToCongestion* is set to the number of hops between the local station and the downstream wrap point (see 10.1.3.14).

Row 10.13-11: An SCFF received on the opposing ringlet indicates downstream congestion. The value of *hopsToCongestion* is computed as the difference between the *ttl* value placed in the frame at the origin station (i.e., MAX_STATIONS) and the *receivedTtl*.

Row 10.13-12: A received SCFF indicates the absence of downstream congestion. The absence of downstream congestion is recorded, but the *hopsToCongestion* is not updated. This allows the most recent congestion point to be referenced until the *allowedRateCongested* is fully ramped up.

Row 10.13-13: No action is taken when an MCFF sourced by a center-wrapped station is received by that station.

Row 10.13-14: An MA_CONTROL.indication (see Table 6.7) is issued to the client.

10.4.9 FddFrameTransmit state machine

On becoming the tail of a congestion domain, a station periodically sends a pair of FDD (see 10.3.2) frames to the head of the congestion domain while the head of the congestion domain continues to be a station deploying the conservative method of rate computation. The two frames are assigned the same sequence number, but one is transmitted with a service class of subclassA0 and the other with a service class of classC. The classA FDD frame is sent before the classC FDD. The classC FDD frame is sent as soon after the classA frame as possible (within the bounds of the MAC datapath transmit rules). A head station receiving the FDD frames computes the FDD value [see Equation (10.26) and Equation (10.27)]. The FDD is a control frame with a *controlType* value of CT_FDD and a *controlDataUnit* consisting of a two-byte sequence number (*sequenceNumber*). The head station considers an FDD frame to be valid only when both the classA and classC FDD frames, carrying the same sequence number, have arrived.

NOTE—It is recommended that the time delay between the sending of the classA FDD and the sending of the classC FDD be as small as possible.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

10.4.9.1 FddFrameTransmit state machine constants

This state machine specifies no definitions.

10.4.9.2 FddFrameTransmit state machine variables

fddInterval

See 10.2.2.

fddStartTime

Records the value of *currentTime*, in microseconds, at the start of an *fddInterval*.

iAmTail

See 10.2.2.

10.4.9.3 FddFrameTransmit state machine routines

IsHeadConservative(ri, mac)

Returns the value of the field *topoEntry.conservativeMode[ri][hops]* for the *topoEntry* identified by the ringlet identifier *receivedRi* and the MAC address identified by *receivedSa*.

TRUE—The head is conservative.

FALSE—(Otherwise.)

OtherRi

See 10.2.2.

*SendFddPair()*Send an FDD frame pair on *myRi* with fields populated (see Table 10.14 and 10.3.2).

10.4.9.4 FddFrameTransmit state table

The send fairness differential delay (FDD) frame state table is specified by Table 10.14.

Table 10.14—FddFrameTransmit

Current state		Row	Next state	
state	condition		action	state
START	iAmTail && IsConservativeMode(receivedRi, receivedSa)	1	frame.ri = myRi; frame.da = receivedSa; SendFddPair(); fddStartTime = currentTime;	TAIL
	—	2	—	START
TAIL	(currentTime - fddStartTime) >= fddInterval && iAmTail	3	frame.ri = myRi; frame.da = receivedSa; SendFddPair(); fddStartTime = currentTime;	TAIL
	(currentTime - fddStartTime) >= fddInterval	4	—	START
	—	5	—	TAIL

Row 10.14-1: The local station is a tail. A pair of FDD frames is sent via ringlet *myRi* (i.e., the ringlet whose traffic is regulated by the fairness instance) to the head station identified by *receivedSa*. An *fddInterval* timer is set in order to send FDD frames periodically to the head until the local station is no longer the tail.

Row 10.14-2: The local station is not a tail. The station continues to test for the tail condition.

Row 10.14-3: The FDD interval timer has expired, and the local station is still a tail. The local stations sends a pair of FDD frames via ringlet *myRi* to the head station identified by *receivedSa*. The *fddInterval* timer is reset.

Row 10.14-4: The FDD interval timer has expired, but the local station is no longer a tail. The station continues to test for the *iAmTail* condition.

Row 10.14-5: The station continues to wait for FDD interval expiration.

10.4.10 FrttComputation state machine

The FrttComputation state machine describes the computation of the FRTT value that is used by the conservativeRateAdjust state machine (see Table 10.9). The FRTT is computed only by a station that performs conservative rate computation and is the head of a congestion domain. The FRTT value is recomputed when a valid pair of FDD frames from the tail station has been received by the head station. The FDD pair is sent by the tail every *fddInterval*. The FRTT value is smoothed over the previous *ageCoef* computations. The head flushes previous results (i.e., does not use them for smoothing purposes) if the tail of the congestion domain changes.

The head station estimates the FRTT by referencing the measured and smoothed fairness differential delay (FDD) and the loop round-trip time (LRTT). FRTT is the sum of FDD and LRTT, rounded to the nearest multiple of the *agingInterval*:

$$\text{FRTT} = \text{FDD} + \text{LRTT} \quad (10.15)$$

FDD is a measure of the difference in delay between the classA and classC paths from the tail station to the head station. LRTT is a measure of link delay experienced by classA frames from the head of the congestion domain to the tail of the congestion domain and back (see 3.2.54). LRTT values are maintained in the topology database, and the calculation of the LRTT is described in 11.6.14.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

10.4.10.1 Computing the FDD

Upon receipt of a pair of FDD frames associated with the same sequence number, the head station calculates the FDD, in microseconds, based on the last received FDD frame pair as specified by Equation (10.16).

$$\begin{aligned} \text{lastReceivedFdd} &= \text{fddArrivalClassC} - \text{fddArrivalClassA} \\ &- (\text{timeStampFddClassC} - \text{timeStampFddClassA}) \end{aligned} \quad (10.16)$$

Where *fddArrivalClassC* and *fddArrivalClassA* are the arrival times of classC and classA FDD frames, respectively and *timeStampFddClassC* and *timeStampFddClassA* are the values of the *fddTimeStamp* field of the received classC and classA frames, respectively. The FDD value is smoothed by each received pair of FDD frames using the smoothing algorithm described by Equation (10.17).

$$\text{FDD} = (\text{FDD} * (\text{ageCoef} - 1) / \text{ageCoef}) + (\text{lastReceivedFdd} / \text{ageCoef}) \quad (10.17)$$

10.4.10.2 Computing the LRTT associated with a wrapped path

The *lrrt* values maintained in the topology database are valid when the path between the head station and the tail station is not wrapped, as illustrated by Figure 10.17.

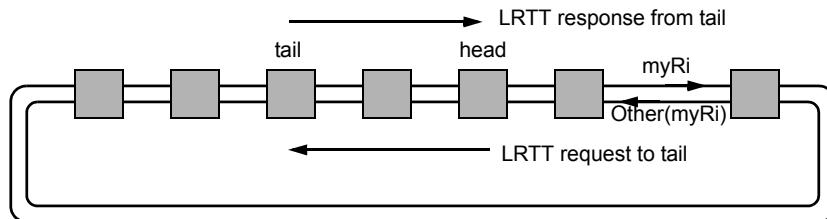


Figure 10.17—LRTT associated with path that is not wrapped

Although not maintained in the topology database, the *lrrt* associated with a wrapped path can be inferred from *lrrt* values associated with paths that are not wrapped. Three distinct cases of *lrrt* computation for wrapped paths are specified by rows 13, 15, and 16 of Table 10.15, respectively.

Figure 10.18 illustrates the case in which the tail station lies downstream of the head station and both stations lie on the same ringlet. In this case, an LRTT request sent upstream (i.e., from the head station toward the tail station on the ringlet opposing that of the data traffic) would transit both the upstream wrap point and the downstream wrap point before arriving at the tail. Similarly, the LRTT response would transit

both the downstream wrap point and the upstream wrap point before arriving at the head. The resulting *lrrt* value is computed as:

$$2 * \text{topoEntry}[\text{Other}(myRi), usWrapPoint].lrrt + 2 * \text{topoEntry}[myRi, dsWrapPoint].lrrt - \text{topoEntry}[myRi, tail].lrrt \quad (10.18)$$

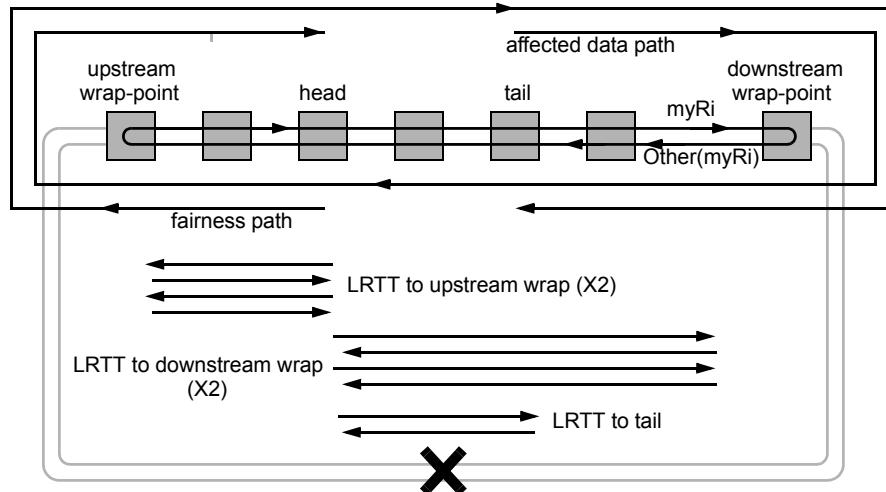


Figure 10.18—Tail downstream of head on same ringlet

The arrows labeled “fairness path” and “affected data path” represent the wrapped path between the head and tail for which an *lrrt* value is to be computed. The arrows labeled “LRTT to upstream wrap point,” “LRTT to downstream wrap point,” and “LRTT to tail” represent valid *lrrt* values maintained in the topology database from which the required *lrrt* value is constructed according to Equation (10.18).

Figure 10.19 illustrates the case in which the head and tail stations are on opposing ringlets and the tail station lies upstream of the head station with respect to the ringlet on which the head station lies. In this case, an LRTT request sent upstream from the head station would transit the tail station and the upstream wrap point before arriving at the tail station. Similarly, an LRTT response would transit the upstream wrap point and the tail station before arriving at the head station. The resulting *lrrt* value is computed as:

$$2 * lrrt[\text{Other}(myRi), usWrapPoint] - lrrt[myRi, tail] \quad (10.19)$$

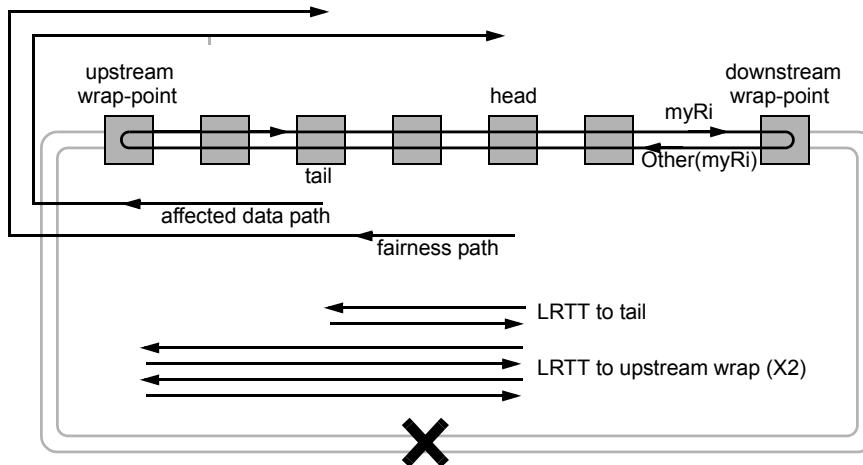


Figure 10.19—Tail upstream of head on opposing ringlet

Figure 10.20 illustrates the case in which the head and tail stations are on opposing ringlets and the tail station lies downstream of the head station with respect to the ringlet on which the head station lies. In this case, an LRTT request sent upstream from the head station would not first transit the tail station, i.e., on *Other(myRi)*, before arriving at the tail station. Similarly, an LRTT response would not first transit the tail station (i.e., on *myRi*) before arriving at the head station. The resulting *lrtt* value is computed as:

$$2 * \text{lrtt}[\text{Other}(\text{myRi}), \text{usWrapPoint}] + \text{lrtt}[\text{myRi}, \text{tail}] \quad (10.20)$$

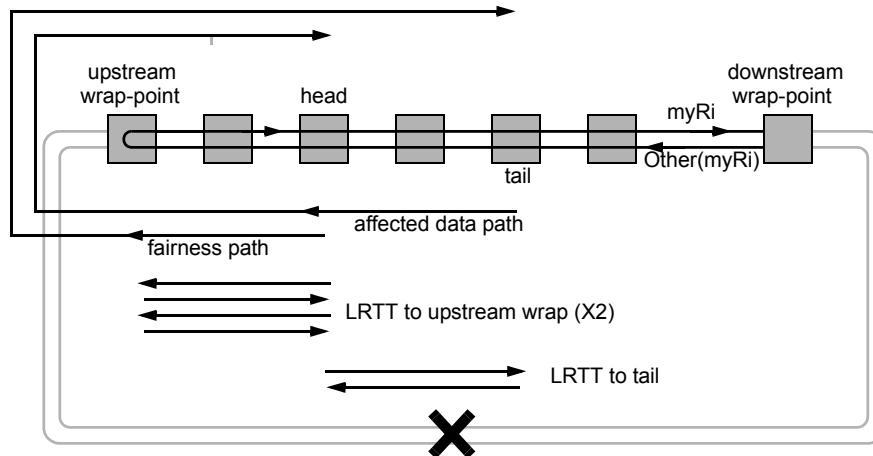


Figure 10.20—Tail downstream of head on opposing ringlet

10.4.10.3 FrrtComputation state machine definitions

No definitions are specified for this state machine.

10.4.10.4 FrrtComputation state machine variables

classASeqNum

Sequence number associated with a received classA FDD.

classATimeStamp

Value of the *currentTime* when a classA FDD was sent.

classCSeqNum

Sequence number associated with a received classC FDD.

classCTimeStamp

Value of the *currentTime* when a classC FDD was sent.

dsWrapPoint

Number of hops to the edge lying downstream of the local station on a wrapped ring.

fdd

The difference in delay, in microseconds, between the classA and classC paths from the tail station to the head station.

fddMaxWait

See 10.2.2.

frrt

See 10.2.2.

lrrtToTail

The *lrrt* value measured between head and tail and adjusted for the case of a wrapped ring.

myRi

See 10.2.4.

newFdd

Most recent value of FDD input to FDD aging.

protConfig

See 10.2.4.

tailIndex

Index into *topoEntry* for the current tail station.

tailRi

Ringlet on which the tail station sent the FDD.

tailSa

The MAC address of the tail station.

timeFddA

Value of *currentTime* when a classA FDD is received.

timeFddC

Value of *currentTime* when a classC FDD is received.

usWrapPoint

Number of hops to the edge lying upstream of the local station on a wrapped ring.

10.4.10.5 FRTT computation state machine routines

AgeFdd(fdd, newFdd)

Age the FDD value.

$$\text{fdd} = (\text{fdd} * (\text{ageCoef} - 1) / \text{ageCoef}) + (\text{newFdd} / \text{ageCoef}) \quad (10.21)$$

GetTopoIndex(tailRi, tailSa)

Return the index of the topology entry corresponding to the specified ringlet identifier and source MAC address.

10.4.10.6 FrttComputation state table

Table 10.15—FrttComputation

Current state		Row	Next state	
state	condition		action	state
START	—	1	classASeqNum = 0; classATimeStamp = 0; classCSeqNum = 0; classCTimeStamp = 0; fdd = 0; frtt = 0;	RCVA
RCVA	(frame = Dequeue(Q_RX_FDD)) != NULL && frame.ttl != 0	2	—	TSTA
	—	3	—	RCVA
TSTA	frame.ft == FT_CONTROL && frame.controlType == CT_FDD && frame.sc == CLASS_A0	4	classASeqNum = frame.sequenceNumber; classATimeStamp = frame.fddTimeStamp; timeFddA = currentTime; tailSa = frame.sa; tailRi = frame.ri; tailIndex = GetTopoIndex(tailRi, tailSa);	RCVC
	—	5	—	RCVA

Table 10.15—FrttComputation (continued)

Current state		Row	Next state	
state	condition		action	state
RCVC	currentTime - timeFddA > fddMaxWait	6	—	RCVA
	(frame = Dequeue(Q_RX_FDD)) != NULL && frame.ttl != 0	7	—	TSTC
TSTC	frame.ft == FT_CONTROL && frame.controlType == CT_FDD && frame.sc == CLASS_C && frame.sequenceNumber == classASeqNum && frame.sa == tailSa && frame.ri == tailRi	8	classCSeqNum = frame.sequenceNumber; classCTimeStamp = frame.fddTimeStamp; timeFddC = currentTime;	PAIR
	—	9	—	RCVC
	classCTimeStamp < classATimeStamp	10	—	RCVA
PAIR	frame.sa == lastSa	11	newFdd = (timeFddC - timeFddA) - (classCTimeStamp - classATimeStamp); fdd = AgeFdd(fdd, newFdd); lastSa = frame.sa;	LRTT
	—	12	fdd = (timeFddC - timeFddA) - (classCTimeStamp - classATimeStamp); lastSa = frame.sa;	
	tailRi == myRi && topoType == OPEN_RING && protConfig == WRAPPING && tailIndex < totalHopsTx[myRi]	13	usWrapPoint = totalHopsRx[myRi]; dsWrapPoint = totalHopsTx[myRi]; lrrtToTail = 2*topoEntry[myRi][usWrapPoint].lrrt + 2*topoEntry[otherRi][dsWrapPoint].lrrt - topoEntry[otherRi][tailIndex].lrrt ;	FRTT
LRTT	tailRi == myRi	14	lrrtToTail = topoEntry[myRi][tailIndex].lrrt ;	
	topoType == OPEN_RING && protConfig == WRAPPING && totalHopsRx[myRi] < tailIndex	15	wrapPoint = totalHopsRx[myRi]; lrrtToTail = 2*topoEntry[myRi][wrapPoint].lrrt - topoEntry[myRi][tailIndex].lrrt ;	
	topoType == OPEN_RING && protConfig == WRAPPING	16	wrapPoint = totalHopsRx[myRi]; lrrtToTail = 2*topoEntry[myRi][wrapPoint].lrrt + topoEntry[otherRi][tailIndex].lrrt ;	
	—	17	lrrtToTail = topoEntry[myRi][tailIndex].lrrt ;	
FRTT	—	18	frtt = fdd + lrrtToTail;	RCVA

Row 10.15-1: Initialize variables.**Row 10.15-2:** A frame with nonzero TTL is received. Proceed to test for valid classA FDD frame.**Row 10.15-3:** A frame has not been received, or the TTL of a received frame is zero. Remain in the receiving state.**Row 10.15-4:** The frame is a classA FDD. The sequence number and timestamp carried by the frame and the current time are recorded.

Row 10.15-5: The frame is not a classA FDD. Return to the receiving state.

Row 10.15-6: More than $fddMaxWait$ time has elapsed between the arrival of the classA and classC FDD frames. Return to the classA receive state as a classC FDD frame has not arrived within $fddMaxWait$ of the classA FDD frame arrival time.

Row 10.15-7: A frame with nonzero TTL is received. Proceed to test for valid classC FDD frame.

Row 10.15-8: The frame is a classC FDD with a sequence number matching that of the previously received classA FDD. The timestamp carried by the frame and the current time are recorded.

Row 10.15-9: Remain in the classC receive state as the received frame does not form an FDD pair with the previously received classA FDD frame.

Row 10.15-10: The classA and classC FDD frames form a pair, but the classC frame was sent before the classA frame and is therefore invalid. Return to the classA receive state to receive a new pair.

Row 10.15-11: The FDD frames form a pair. The head has not changed since the last fdd computation. The fdd is computed and aged.

Row 10.15-12: The FDD frames form a pair. The head has changed since the last fdd computation. The fdd is computed but not aged.

Row 10.15-13: The head and tail lie on the same ringlet on a wrapped ring. Traffic passing from tail to head wraps to the opposing ringlet and wraps back from the opposing ringlet. Traffic traveling downstream from the tail encounters an edge before it encounters the head station. Traffic traveling downstream from the head station encounters the tail before it encounters the head. The latter condition provides the test for this case.

Row 10.15-14: The head and tail lie on the same ringlet. Traffic does not wrap when traveling from head to tail, and the $lrrt$ value of the tail station can be used without modification.

Row 10.15-15: The head and tail lie on different ringlets of a wrapped ring. Traffic traveling from the tail instance to the head instance transits the station containing the tail instance en route to the head instance (i.e., the tail station lies upstream of the head station with respect to the ringlet on which the head instance is located). The $lrrt$ value is derived from available $lrrt$ values (see Figure 10.19).

Row 10.15-16: The head and tail lie on different ringlets of a wrapped ring. Traffic traveling from the tail instance to the head instance does transit the station containing the tail instance en route to the head instance (i.e., the tail station lies downstream of the head station with respect to the ringlet on which the head instance is located). The $lrrt$ value is derived from available $lrrt$ values (see Figure 10.20).

Row 10.15-17: Cases not meeting the conditions of Rows 10.13-13 to 10.13-16.

Row 10.15-18: The $frtt$ is computed as the sum of the fdd and the $lrrt$ to the tail.

10.5 Explanation of aging and rates (informative)

The rate counters $addRate$, $addRateCongested$, $fwRate$, $fwRateCongested$, and $nrXmitRate$ are conditionally incremented with each byte of a data frame that is dequeued by the datapath for addition to the ringlet or transited by the station (see 10.1.3.9). There are a number of ways that a rate counter could provide a rate value. One method would be to allow the rate counter to maintain a simple count over time. The rate could be derived by dividing the count by the number of time intervals that have elapsed. This method has the disadvantage that the counter will overflow, as there is no provision for reducing the count. An alternative would be to store the counts associated with the previous N intervals. When an interval is complete, the new count is recorded, and the oldest count is dropped. Although this method avoids the overflow of the rate counter, it also introduces the requirement to store N counts. The aging method employed by the fairness procedures ensures that the rate counter will not overflow and does not require more than a single count to be maintained for any individual rate counter.

The aging method has the following three objectives:

- Ensuring that the rate counters do not overflow.
- Smoothing the rate counter values.
- Transforming the rate counter values to rates.

Taking $addRate$ as an example, the rate counter is aged as follows at the expiration of each $agingInterval$:

$$addRate = (addRate_{-1} + \Delta) \cdot k \quad (10.22)$$

$$k = \frac{ageCoef - 1}{ageCoef} \quad (10.23)$$

where $addRate_{-1}$ denotes the value of $addRate$ at the start of the current $agingInterval$ and Δ denotes the number of bytes added during the current $agingInterval$. When adding bytes at a constant rate (Δ bytes per $agingInterval$) and the initial value of $addRate$ is 0, $addRate$ can be expressed as a geometric series:

$$addRate_n = \Delta + \Delta k + \Delta k^2 + \dots + \Delta k^{n-1} \quad (10.24)$$

$$\lim_{n \rightarrow \infty} addRate_n = \frac{\Delta}{1-k} = \frac{\Delta}{1 - \frac{ageCoef - 1}{ageCoef}} = \Delta \cdot ageCoef, |k| < 1 \quad (10.25)$$

As an example, assume that $ageCoef = 4$, $addRate$ is initially 0, and that the $agingInterval$ is 100 μ sec (1 / 10^{**4} seconds). Assume also that bytes are added at a constant rate of 10^7 bytes/second, that is, 1000 bytes are added every $agingInterval$.

This gives $k = 0.75$ and $\Delta = 1000$ and $\lim_{n \rightarrow \infty} addRate_n = 4000$

During the first $agingInterval$, $addRate$ is incremented from 0 to 1000, but aging at the end of the interval sets it back to $1000 * 3/4 = 750$.

The next interval increments the counter to 1750. Aging sets it back to $1750 * 3/4 = 1312$.

The next 1000 bytes increases the counter to 2312. Aging sets it back to 1734.

The next 1000 bytes increases the counter to 2734. In subsequent $agingIntervals$, the count takes the values (at the end of the $agingInterval$, but before aging) 3050, 3287, 3465, 3598, 3698, 3773, and so on. The first 28 values are illustrated in Figure 10.21.

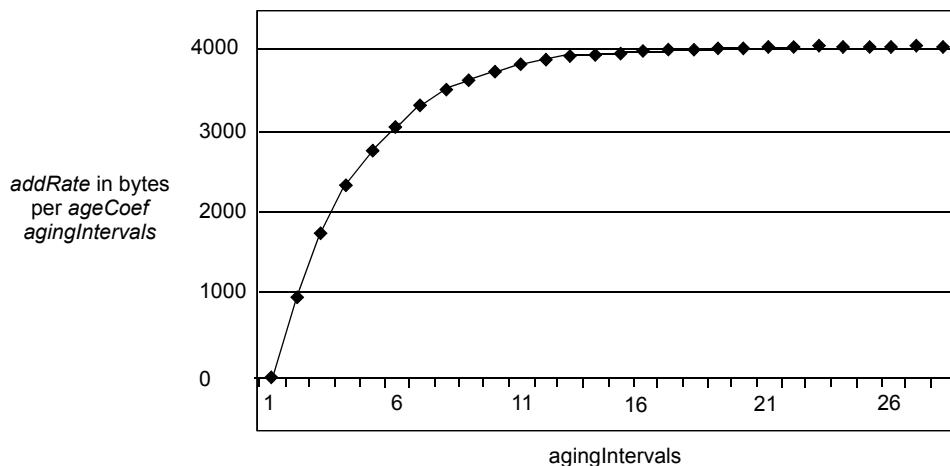


Figure 10.21—Rate counter asymptotically approaching rate

The value of *addRate* asymptotically approaches 4000. If it becomes 4000, aging decreases it to $4000 * 3/4 = 3000$. At the addition of 1000 bytes at the next *agingInterval*, the *addRate* again reaches 4000. Equation (10.25) shows that *addRate* will not exceed 4000 (i.e., $\Delta * ageCoef = 4000$).

It can be observed also that the value of the *addRate* is *ageCoef* times the actual rate of byte addition in units of bytes per *agingInterval*. It follows that the unit associated with the *addRate* (and rate counters in general) is bytes per *ageCoef agingIntervals*. Many rates used in the fairness procedures are derived from the rate counters, so it follows that this unit is commonly used in the fairness procedures.

It follows also that the value of a rate counter can be converted to the more conventional unit of bytes per second as follows:

$$\text{rate bytes/second} = \text{addRate} / (\text{ageCoef} * \text{agingIntervals}) \quad (10.26)$$

Using the values from the example above, we have the following:

$$\text{rate} = 4000_{\text{bytes per ageCoef agingIntervals}} / (4 * 10^{-4}_{\text{seconds per agingInterval}}) = 10^7_{\text{bytes/second}} \quad (10.27)$$

Examples of rates associated with specific PHY types are provided by Table 10.16.

Table 10.16—PHY types and associated rates

PHY	<i>lineRate</i> (Mb/s)	LINK_RATE	<i>rateCoef</i>	<i>agingInterval</i> (microseconds)
STS-3c	149.76	59 904	1/8	400
STS-12c	599.04	59 904	1/8	100
1 Gb/s PacketPHY	1000	50 000	1/4	100
STS-48c	2396.16	59 904	1/2	100
STS-192c	9584.64	59 904	2	100
10 Gb/s PacketPHY	10000	62 500	2	100

The *lineRate* for the SONET PHY is its payload capacity, not its physical line rate.

The *fairRate* field of the fairness frame is encoded as a 16-bit integer value (see 10.3.1). The LINK_RATE (i.e., the full-scale rate value in units of bytes per *ageCoef agingIntervals* associated with a specific PHY type) can assume different values depending on the PHY type. This value is always less than the maximum value 65 535 that can be encoded in the *fairRate* field.

10.6 Protocol Implementation Conformance Statement (PICS) proforma for Clause 10²²

10.6.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 10, Fairness, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

10.6.2 Identification

10.6.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

10.6.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, Fairness
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

²²Copyright release for PICS proformas: Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

10.6.3 PICS tables for Clause 10

Item	Feature	Subclause	Value/Comment	Status	Support
FA1	Rate statistics	10.4.1	Does the MAC maintain per-byte rate statistics?	M	Yes []
FA2	Rate statistics maintenance delay	10.4.1	Does the MAC delay the maintenance of rate statistics by no more than 256 bytes?	M	Yes []
FA3	Policing indicators	10.4.1	Does the MAC maintain policing indicators?	M	Yes []
FA4	Aging	10.4.2.4	Does the MAC periodically age rate-counts?	M	Yes []
FA5	Low-pass filtering	10.4.2.4	Does the MAC periodically smooth rate-counts by low-pass filtering?	M	Yes []
FA6	Aging interval	10.4 Table 10.2	Does the MAC deploy an aging interval conforming to Table 10.2?	M	Yes []
FA7a	Aggressive rate computation	10.4.3	Does the MAC deploy the aggressive rate computation method?	O.1	Yes [] No []
FA7b	Conservative rate computation	10.4.4	Does the MAC deploy the conservative rate computation method?	O.1	Yes [] No []
FA8	Rate advertisement	10.4.5	Does the MAC send a rate advertisement (i.e., an SCFF) to its upstream neighbor at the expiration of each advertising interval?	M	Yes []
FA9	Rate report	10.4.6	Does the MAC broadcast a rate report (i.e., an MCFF) on the ringlet at the expiration of each reporting interval?	M	Yes []
FA10	Active weights computation	10.4.7	Does the MAC support active weights computation?	FA7a:X FA7b:O	Yes [] No []
FA11	Single-choke fairness frames	10.4.8	Does the MAC receive and process single-choke fairness frames?	M	Yes []
FA12	Multi-choke fairness frames	10.4.8	Does the MAC receive and process multi-choke fairness frames?	O	Yes [] No []

Item	Feature	Subclause	Value/Comment	Status	Support
FA13	FDD frame transmit	10.4.9	Does the MAC support the transmission of FDD frames?	M	Yes []
FA14	FRTT computation	10.4.10	Does the MAC support the computation of the fairness round-trip time?	FA7a:X FA7b:M	Yes []
FA15a	SHAPER_BASED admission	10.4.3	Does the MAC deploy the SHAPER_BASED admission method?	O.2	Yes [] No []
FA15b	RATE_BASED admission	10.4.3	Does the MAC deploy the RATE_BASED admission method?	O.2	Yes [] No []
FA16	Local weight value limits	10.2.2	Are the local weight values limited to powers of two?	O	Yes [] No []

11. Topology discovery and protection

11.1 Overview

This clause specifies the topology discovery and protection functions of the MAC, including:

- a) Topology discovery for plug-and-play topology modification.
- b) Protection switching in under 50 milliseconds, with no reordering/duplication of strict data frames.
- c) Attribute discovery (ATD) for advertising of additional, less time-critical information.
- d) Topology checksum (TC) exchange to minimize loss of strict data frames due to topology changes.
- e) Loop round-trip time (LRTT) measurement to enhance conservative-mode fairness performance.

The topology, protection, ATD, TC, and LRTT measurement functions reside in the MAC control sublayer, as shown in the shaded region of Figure 11.1.

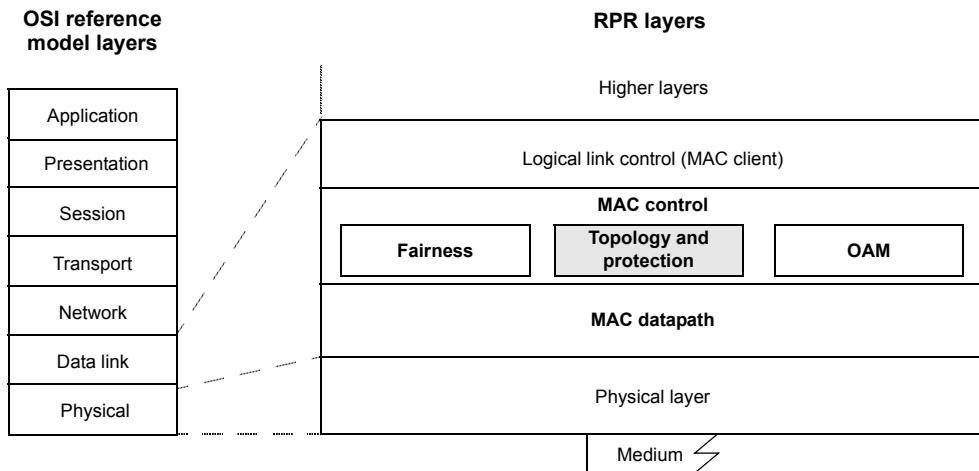


Figure 11.1—Topology and protection entities relationship to the ISO/IEC OSI reference model

RPR protection provides reliable mechanisms for under 50 millisecond protection switching (not counting configurable holdoff time or failure detection time) for all protected traffic on a ring. It enables a mandatory protection mechanism called steering and an optional protection mechanism called wrapping. This protocol ensures that each station receives span status change information, e.g., span failure or restoration information, required to make protection switching decisions reliably and in the time frame required. This protocol also ensures that stations steer or wrap in accordance with the protection hierarchy.

RPR topology discovery provides a reliable and accurate means for all stations on a ring to discover the topology of the stations on the ring and any changes to that topology. It also provides a mechanism for rapid detection of topology changes. The topology discovery and protection functions share a common control messaging mechanism.

The information gathered by the topology discovery, protection, ATD, TC, and LRTT measurement functions are stored together in a shared topology database. Information in the topology database is also used by the MAC datapath (see Clause 7), the fairness protocol (see Clause 10), and OAM (see Clause 12).

The topology and protection entity has the following features:

- a) Responsive
 - 1) Restoration of protected traffic in less than 50 milliseconds.
 - 2) Quick dissemination on the ring of changes in protection state.
- b) Robust
 - 1) Support of a comprehensive protection hierarchy.
 - 2) Support of dynamic addition and removal of stations to/from the ring.
 - 3) Tolerant of control frame loss.
 - 4) Operation without any master station on the ring.
 - 5) Operation independent of any management systems.
 - 6) Validation of topology, including detection of miscabling between stations.
 - 7) Assurance that all stations on the ring converge to a uniform and current image of the topology.
 - 8) Context containment to prevent duplication and reordering of strict mode traffic.
- c) Flexible
 - 1) Support of revertive and nonrevertive protection switching.
 - 2) Support of closed-ring and open-ring topologies.
 - 3) Scalability from 1 to 255 stations.
 - 4) Means of sharing additional information between stations.
- d) Efficient
 - 1) Requires insignificant ring traffic.
 - 2) Consumes minimal software execution time.
 - 3) Requires minimal hardware facilities.

11.1.1 Protocol overview

11.1.1.1 Discovery protocol

The topology discovery protocol provides each station on the ring with knowledge of the number, basic capabilities, and arrangement of other stations on the ring. This collection of information is referred to as the topology database. When the basic content of the database (i.e., station connectivity) ceases to change, the topology is called stable.

The critical topology database information is derived from information supplied by topology and protection (TP) control frames received on each ringlet. TP frames include span protection state, edge status, and protection configuration information. TP frames are sent periodically and when triggered by protection-state changes.

The attribute discovery (ATD) control frame communicates less time-critical information for placement into the topology database. The ATD frames include standard station preferences and optional organization-specific information, although the use of organization-specific information is beyond the scope of this standard. ATD frames are sent periodically and when triggered by changes in their content.

11.1.1.2 Protection protocol

The protection protocol uses span state information from the topology database to determine whether its local spans can be used to transport data frames. This information is given to the MAC datapath sublayer, so that it can steer or wrap protected traffic as required. The protection protocol uses configuration parameters, such as hold-off time and wait-to-restore time, and a hierarchy of states derived from link monitoring and operator commands.

11.1.1.3 Topology discovery and protection time

All stations are notified of changes in protection state within approximately one ring round-trip time (RRTT) plus processing delays. For a 1000 km ring with no lost TP frames, the maximum notification time is approximately 5 milliseconds. Lost TP frames cause the notification time to increase by the fast retransmit time of the TP frame (default of 10 milliseconds).

Protection of relaxed mode traffic occurs upon notification of a topology change, whereas protection of strict mode traffic occurs after topology is validated.

Topology convergence time consists of notification time (generally one RRTT) and stabilization time (default 40 milliseconds). If a station enters passthrough, the maximum notification time is approximately $RRTT * MAX_STATIONS$ divided by the number of stations on the ring, due to delays in removing stale TP frames sourced from the passthrough station. If a station enters passthrough, the passthrough convergence time is approximately 120 milliseconds, for a 16-station 1000 km ring.

11.1.1.4 Topology validation

Topology validation is performed after the topology has stabilized. Topology validation reports topology defects to OAM. Examples of defects include miscabling, topology instability, topology inconsistency, and too many stations (see 11.2.9).

Topology must be valid before context containment can be cleared (see 11.1.3).

11.1.1.5 Edges

An edge is a span not currently being used for data traffic, due to its protection state. The detection of an edge in a ring makes it an open ring and results in protection of traffic.

TP frames continue to be transmitted between neighbors across an edge to determine when the edge state can be cleared. Single-choke fairness frames (SCFF) are also transmitted between neighbors across an edge to monitor span/station availability. No other frames are transmitted or received across an edge span (see Clause 7).

11.1.2 Topology database maintenance

11.1.2.1 Topology database construction

A station creates an initial topology database consisting of only itself, and then it broadcasts TP frames and ATD frames on both ringlets. Each station uses received TP frames to update the topology database.

11.1.2.2 Addition of a station

When a station is added to an existing ring, its neighbors detect the station as a new neighbor by receiving its TP frame (indicating the frame has traveled exactly one hop) and an *sa* (source MAC address) different from that of the previously observed neighbor. When a station exits passthrough, it behaves as a newly initialized station.

A new station on the ring transmits TP frames on both ringlets. Other stations detect the new station by receiving its TP frame, and each responds with TP frames on both ringlets. Received TP frames are used to update the topology database (see 11.5).

11.1.2.3 Span and station failures

When a span is determined to be an edge, topology database information beyond the edge (except for neighbor information) is marked as invalid, and TP frames are sent to report the edge. This includes the case of a station being removed from a ring.

11.1.2.4 Passthrough

When one or more stations enter passthrough, they effectively disappear from the ring. Passthrough is detected when a station receives a TP frame with a different *sa* (source MAC address) than expected for the received hop count. Each station transmits TP frames when it detects a passthrough event.

Passthrough detection speed depends on the setting of the slow TP frame transmission period (default of 100 milliseconds). Passthrough detection is not covered by the 50 millisecond traffic restoration requirement.

11.1.3 Context containment

Context containment assists in preventing duplication and reordering of strict mode traffic during topology changes (see 7.6.1.1). Context containment is entered on passthrough, when an edge state changes on a steering ring, or when an edge is removed on a wrapping ring. Detection that a station's checksum does not match the neighbor checksums does not trigger context containment.

Context containment is not exited until the following conditions are met:

- a) The topology is determined to be stable and valid.
- b) The local topology database checksum is determined to match those of its reachable neighbors.

In the absence of a passthrough event, context containment duration is dominated by topology stabilization. Otherwise, the duration is significantly impacted by a larger topology discovery time.

The topology checksum is contained within topology checksum (TC) frames. TC frames are sent periodically and when its contents change.

Figure 11.2 illustrates a scenario of entering and exiting context containment.

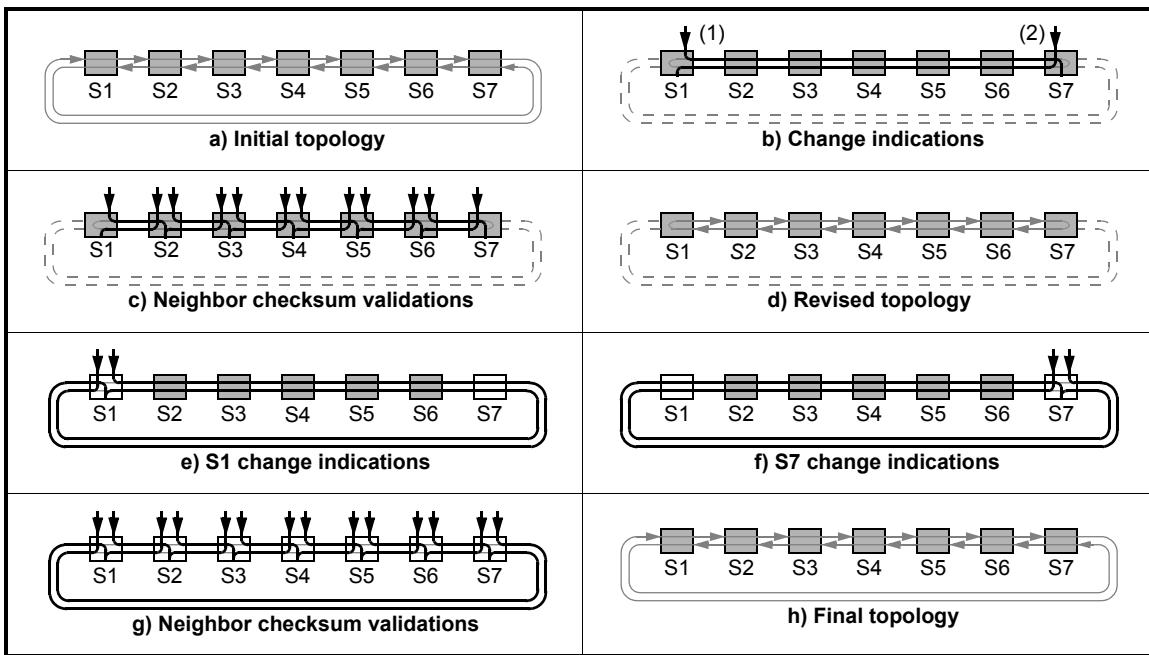


Figure 11.2—Context containment sequence

A closed-ring topology (see Figure 11.2-a) is disrupted by a fault on the S7-S1 span. Stations S1 and S7 broadcast TP frames indicating an edge (see Figure 11.2-b). These TP frames trigger context containment at all steering stations. Context containment is not triggered at any wrapping station.

When the topology becomes stable and valid, each station compares its topology checksum with those of its neighbors (see Figure 11.2-c). Context containment is cleared when the checksums match, yielding an open-ring topology (see Figure 11.2-d).

When the fault on the S7-S1 span is cleared, stations S1 (see Figure 11.2-e) and S7 (see Figure 11.2-f) broadcast TP frames indicating the removal of the edge, triggering context containment at all stations.

When the topology becomes stable and valid, each station once again compares its topology checksum with those of its neighbors (see Figure 11.2-g). Context containment is cleared when the checksums match, restoring the initial closed-ring topology (see Figure 11.2-h).

11.1.4 Secondary MAC addresses

Secondary MAC addresses support higher layer redundancy protocols on a ring, by allowing stations to advertise secondary MAC addresses, using the ATD frame. If supported, secondary MAC addresses (see 11.6.13) are validated and stored in the topology database.

11.1.5 LRTT measurement protocol

Conservative mode fairness relies on measurements of loop round-trip time (LRTT). LRTT (see 11.6.14) is measured by sending request frames to each station on each ringlet. Stations respond to each request by sending a response. LRTT values are computed from the content and arrival time of the responses and stored in the topology database.

If required, LRTT is measured when topology is determined to be stable, and optionally thereafter.

11.1.6 Fault response mechanisms

11.1.6.1 Steering protection

Steering stations direct unicast traffic onto ringlet0 or ringlet1 on a per destination basis to avoid failed spans. Multicast frames are sent in both directions, with the *ttl* set to the number of stations to the defective span, on each ringlet.

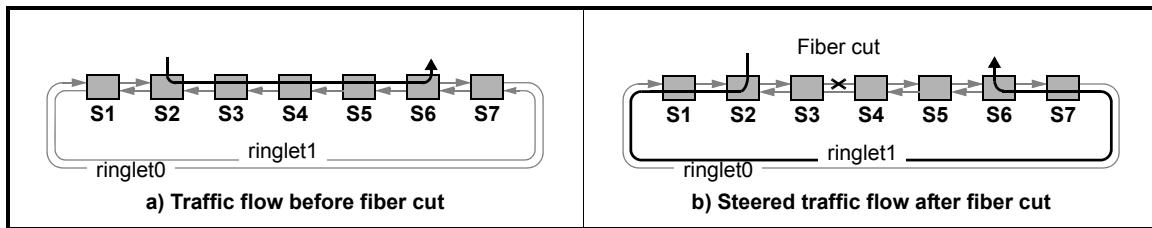


Figure 11.3—Steering protection of unicast traffic

As illustrated in Figure 11.3-a, station S2 normally sends to station S6 via the ringlet0 path $S2 \Rightarrow S3 \Rightarrow S4 \Rightarrow S5 \Rightarrow S6$. After the fiber cut, station S2 updates its topology database and steers protected S6-destined traffic via the ringlet1 path $S2 \Rightarrow S1 \Rightarrow S7 \Rightarrow S6$ (see Figure 11.3-b). In flight frames, destined to stations beyond the point of failure, are dropped at the edge.

All RPR stations shall support steering.

11.1.6.2 Wrapping protection

In addition to steering, RPR stations may support wrapping. Wrapping stations direct traffic onto ringlet0 or ringlet1 on a per destination basis, regardless of failed spans. An edge station wraps eligible frames that would otherwise be transmitted across the edge. A frame is wrapped by transferring it to the other ringlet.

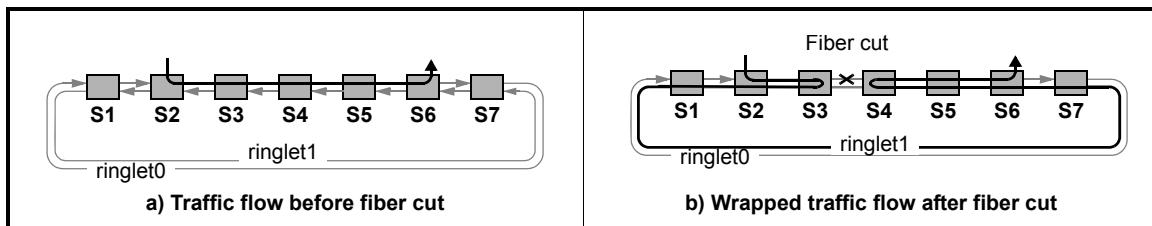


Figure 11.4—Wrapping protection of traffic

As illustrated in Figure 11.4-a, station S2 normally sends to station S6 via the ringlet0 path $S2 \Rightarrow S3 \Rightarrow S4 \Rightarrow S5 \Rightarrow S6$. After the fiber cut, station S2 sends the traffic via the ringlet0 path $S2 \Rightarrow S3 \Rightarrow S2 \Rightarrow S1 \Rightarrow S7 \Rightarrow S6 \Rightarrow S5 \Rightarrow S4 \Rightarrow S5 \Rightarrow S6$ (see Figure 11.4-b). Station S6 does not strip frames from the opposite ringlet of that indicated in the wrapped frame (ringlet1 in this case) to avoid frame misordering during the protection event. In flight frames, destined to stations beyond the point of failure, are wrapped at the edge.

Subsequently, a new ring topology is discovered and the client can choose to resteer the traffic. For example, a new optimal path $S2 \Rightarrow S1 \Rightarrow S7 \Rightarrow S6$ can be used. However, a flush before resteering can be necessary to ensure that frame ordering is preserved (see Annex K). Note that resteering after wrapping is optional.

11.1.6.3 Performance

An RPR ring shall provide protection for relaxed data frames within the combination of detection time and restoration time, and for strict data frames within the combination of detection time, restoration time, and topology stabilization time.

11.1.6.4 Protection hierarchy

The MAC supports the following protection hierarchy (see 11.6.5), listed in order of decreasing severity:

- a) Forced switch (FS)—A management directive that forces a span to be deactivated.
- b) Signal fail (SF)—A signal failure that deactivates the span.
- c) Signal degrade (SD)—A signal degradation that can deactivate the span.
- d) Manual switch (MS)—A management directive that can deactivate the span.
- e) Wait to restore (WTR)—A timer that improves stability in the presence of transient failures.
- f) IDLE—None of the above.

11.1.6.5 Passthrough mode

The optional passthrough mode enables stations to enter or exit the ring without disconnection of fibers (e.g., upon detection of internal failure conditions), without triggering a signal fail event. Passthrough allows a station to leave the ring while maintaining a closed-ring topology, avoiding a protection switch.

Periodic transmission of TP frames allows detection of a station entering passthrough. When another station appears to have changed its location, TP frame transmission is triggered to facilitate fast rediscovery of the topology and restoration of strict mode traffic.

In-flight TP frames (sent by a station that has since entered passthrough) remain on the ring until *ttl*-stripped and can thus circulate many times. These frames extend topology discovery time in passthrough scenarios (see 11.1.1.3) and context containment duration (see 11.1.3).

11.2 Terminology and variables

11.2.1 Terminology

11.2.1.1 Span and ringlet parameterization

For clarity and conciseness of state-machine notation and routine definitions, multiple span-related variables are parameterized by ringlet identifier *ri*. Information associated with west and east spans of a station correspond to *ri* values of 0 and 1, respectively, as illustrated in Figure 11.5.

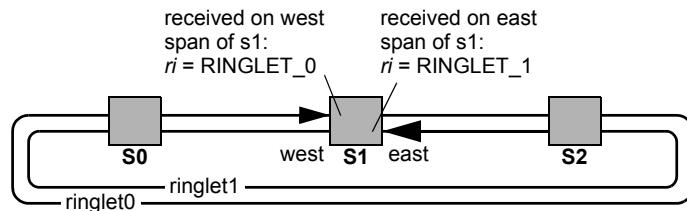


Figure 11.5—Relationship between *ri* value and east/west

For example, *myTopoInfo.spanProtState[0]* contains the protection state of the west span of the station, and *wtr[0].timeout* contains the configured wait to restore value applied to the west span of the station. The same variables parameterized by 1 apply to the east span of the station.

When the context of a state machine is based on the span of a station, *ri* is used to identify the span. An example of this is the ProtectionUpdate state machine (see 11.6.5). Each time this state machine is called, it runs for a span *ri*.

Information received on a given ringlet from other stations is parameterized by the topology database ringlet index *rid*. For example, *topoEntry[rid][hops].spanProtState[ri]* contains the protection state reported in a TP frame received on ringlet *rid* from a distance of *hops*.

Local station information that has scope beyond a span of the station is also parameterized by *rid*. For example, *myTopoInfo.reservedRate[rid]* contains the bandwidth reserved by the station on ringlet *rid*.

When the context of a state machine is based on the ringlet from which a TP frame is received, as opposed to a current ringlet context, *rid* is used to identify the ringlet. An example of this is the ParseTpFrame state machine (see 11.6.4).

11.2.1.2 Detection, restoration, and stabilization times

11.2.1.2.1 Detection time:

Amount of time to detect a change in protection status of a span.
Value: <= 10 milliseconds

11.2.1.2.2 Restoration time:

Amount of time to steer or wrap frames away from a failed span or toward a span whose failure has been removed.

Value: <= 50 milliseconds

11.2.1.2.3 Topology stabilization time:

Amount of time required for topology to stabilize across all stations in a ring, and during which context containment is maintained.

Range: [10 milliseconds, 100 milliseconds]

Default: 40 milliseconds

11.2.2 Common state machine definitions

The following state machine definitions are used multiple times within this clause.

CLOSED_RING

Indicates that the topology is a closed ring, as defined in 3.2. When used to represent a binary value, the value represented is 1.

FS

A (forced switch) administrative request condition set by the operator (see 11.1.6.4 and 11.3.1.3).

IDLE

The normal (idle) operating protection state of a span (see 11.1.6.4 and 11.3.1.3).

JUMBO

Indicates that a ring supports jumbo frames, or that a station is configured to support jumbo frames if all other stations on the ring are configured to support jumbo frames (see 9.2). When representing a binary value, its value is 1.

MAX_SEC_MAC

The maximum number of secondary MAC addresses allowed on a ring. If supported:

Range: [16, MAX_STATIONS]

Default: 32

If not supported:

Value: 0

MAX_STATIONS

The maximum number of stations allowed on a ring. This is 255.

MS

A (manual switch) administrative request condition set by the operator (see 11.1.6.4 and 11.3.1.3).

OPEN_RING

The topology is an open ring. When used to represent a binary value, the value represented is 0.

Q_RX_TP_PARSE

Selects the queue for passing frames between ReceiveTpFrame and TopologyControl.

REGULAR

Indicates that a ring does not support jumbo frames, or that a station is not configured to support jumbo frames (see 9.2). When representing a binary value, its value is 0.

SD

A (signal degrade) protection status condition of a span (see 11.1.6.4 and 11.3.1.3).

SF

A (signal fail) protection status condition of a span (see 11.1.6.4 and 11.3.1.3).

STEERING

Indicates that the protection type of the ring or the protection configuration of a station is steering, as defined in 3.2. When used to represent a binary value, the value represented is 0.

WRAPPING

Indicates that the protection type of the ring or the protection configuration of a station is wrapping, as defined in 3.2. When used to represent a binary value, the value represented is 1.

WTR

A (wait to restore) protection state that occurs upon clearing SF or SD (see 11.1.6.4 and 11.3.1.3).

11.2.3 Common state machine variables

adminReqProtection[ri]

An administrative request value, serviced and cleared by the MAC, with the following values:

FS—Indicates a forced switch command.

MS—Indicates a manual switch command.

IDLE—Indicates the clearing of a forced switch or manual switch command.

NULL—No administrative protection request is available.

atdTimerTimeout

The configured value of the timer used for ATD frame transmission.

Range: [1 second, 10 seconds]

Resolution: 1 second

Default: 1 second

containmentActive

Indication that context containment is active. This can be set by both stages of the topology database update. This is used by Clause 7.

TRUE—Context containment is active.

FALSE—(Otherwise.)

frame

Local: Contents of a frame received from or transmitted to the ring (see 11.3).

holdoffTimeout[ri]

A configurable value for each per span that delays the generation of protection triggers.

Range: [0 milliseconds, 200 milliseconds].

Resolution: 10 milliseconds

Default: 0 milliseconds

keepaliveTime[ri]

The time at which the last keepalive is received (see 7.6.3.6 and 7.6.3.9).

lrrtRequest

A TopologyValidation state machine (see 11.6.6) indication that the topology is stable and a conservative mode station's LRTT measurements (see 11.6.14) can safely start.

TRUE—A new LRTT measurement can start.

FALSE—(Otherwise.)

needSecondaryMacValidation

Indication from the TopologyValidation state machine in Table 11.16 that the topology has just become valid, and that execution of the SecondaryValidation state machine is required.

TRUE—The condition described above is in effect.

FALSE—(Otherwise.)

newNeighbor[ri]

Indication of a new neighbor associated with the span (see 11.6.4 and 11.6.5).

TRUE—There is a new neighbor.

FALSE—(Otherwise.)

pircTopologyChange

Indicates that a topology change has occurred requiring an update of the PIRC protection status.

TRUE—A topology changed has occurred requiring an update of the PIRC protection status.

FALSE—(Otherwise.)

protectChanged

A ParseTpFrame state machine (see 11.6.4) indication used to invoke actions in the TopologyControl state machine (see 11.6.3) when other stations' protection conditions have changed.

TRUE—Protection conditions have changed.

FALSE—(Otherwise.)

revertive

Configured to indicate if the station is in revertive or nonrevertive protection mode.

TRUE—The station is in revertive mode.

FALSE—(Otherwise.)

(default): Shall behave as though set to TRUE. Nonrevertive operation may be supported.

ri

Receive ringlet identifier corresponding to a physical span of a station.

rid

The first index value for *topoEntry[][]*, representing the applicable ringlet.

spanOperStatus[ri]

Operational status per span of the station.

IDLE—None or an insignificant error rate.

SD—A significant error rate indicating a degraded span (signal degrade).

SF—A unacceptable error rate indicating a span has failed (signal fail).

spanProtAdmin[ri]

Administrative state per span of the station.

IDLE—Normal operation, span in use.

MS—Indicates a desire to deactivate the span.

FS—Indicates a mandate to deactivate the span.

tempHopsRx[rid]

Temporary value of hops in the receive direction on ringlet *rid* before reaching oneself or before reaching an edge. This variable is updated during the topology validation process.

topoChanged

A ParseTpFrame (see 11.6.4) indication used to invoke actions in TopologyControl (see 11.6.3) and TopologyValidation (see 11.6.6) state machines when the topology has changed.

TRUE—Station positions and edge information in the topology database have changed.

FALSE—(Otherwise.)

topologyStable

A TopologyValidation (see 11.6.6) state machine indication of when the topology is stable.

TRUE—The topology is stable.

FALSE—(Otherwise.)

topoType

Variable that stores a computed value of ring topology type before topology is validated. At that point, the value of *topoType* is set into the topology database.

CLOSED_RING—The topology is a closed ring.

OPEN_RING—(Otherwise.)

topologyValid

A value set by the TopologyValidation state machine (see 11.6.6) state machine to indicate when the topology is valid.

TRUE—The topology is valid.

FALSE—(Otherwise.)

transmitTcFrame

Generated by the TopologyValidation state machine (see Table 11.16) to confirm the topology is valid, to generate new TC frames with a new topology checksum.

TRUE—The topology is valid.

FALSE—(Otherwise.)

transmitTpFrame

The trigger for TP frame transmissions. The *transmitTpFrame* value is set by ParseTpFrame and ProtectionUpdate state machines, and cleared by the TransmitTpFrame state machine.

TRUE—Transmission of TP frames is triggered.

FALSE—(Otherwise.)

txFastTimeout

The configured value of the timer used for fast TP and TC frame transmission.

Range: [1 millisecond, 20 milliseconds]

Resolution: 1 millisecond

Default: 10 milliseconds

txSlowTimeout

The configured value of the timer used for slow TP and TC frame transmission.

Range: [100 milliseconds, 1 second]

Resolution: 100 milliseconds

Default: 100 milliseconds

wtr[ri]

The wait to restore parameters per span, including the following:

time—The time at which a wait to restore (WTR) timeout was started.

timeout—The time delay for which a wait to restore (WTR) timeout is invoked.

Range: [0 seconds, 1440 seconds].

Resolution: 1 second

Default: 10 seconds

enabled—An indication of whether the WTR is operational.

TRUE—The WTR is operational.

FALSE—(Otherwise.)

11.2.4 *ringInfo* fields

The topology database (see 11.5) is divided into *ringInfo*, *myTopoInfo*, and *topoEntry* portions. The *ringInfo* portion of that database includes the following field values:

badFcsUser

Indicates whether any station on the ring copies data frames with invalid *fcs* fields to the client.

TRUE—At least one station that copies data frames with invalid *fcs* fields to the client.

FALSE—(Otherwise.)

jumboType

The indication whether a ring supports jumbo frames.

JUMBO—The ring supports jumbo frames.

REGULAR—(Otherwise.)

multichokeUser

Indication whether any station expects to receive multi-choke fairness frames.

TRUE—There is at least one station that expects to receive multi-choke fairness frames.

FALSE—(Otherwise.)

mtuSize

An indication of the MTU size on the ring.

JUMBO_MAX—The MTU size on the ring is JUMBO_MAX.

REGULAR_MAX—The MTU size on the ring is REGULAR_MAX.

numStations

Total number of stations on the ring.

topoType

The indication of the ring topology type.

OPEN_RING—Indicates an open ring topology.

CLOSED_RING—Indicates a closed ring topology.

totalHopsRx[rid]

Hops in the receive direction on ringlet *rid* before reaching oneself or before reaching an edge.

totalHopsTx[rid]

Hops in the transmit direction on ringlet *rid* before reaching oneself or before reaching an edge.

unreservedRate[rid]

The difference between LINK_RATE and the total reserved bandwidth on ringlet *rid*, as computed in 11.4.2.

11.2.5 *myTopoInfo* fields

The topology database (see 11.5) is divided into *ringInfo*, *myTopoInfo*, and *topoEntry* portions. The *myTopoInfo* portion of that database includes the following field values:

badFcsUser

Indicates whether the station copies data frames with invalid *fcs* fields to the client.

TRUE—The station copies data frames with invalid *fcs* fields to the client.

FALSE—(Otherwise.)

myChecksum

Topology checksum information for this station, including the following:

value—checksum value

valid—an indication of whether the checksum is valid:

TRUE—The checksum is valid.

FALSE—(Otherwise.)

conservativeMode

Indicates whether this station is configured to run conservative mode.

TRUE—The station is configured to run conservative mode.

FALSE—The station is configured to run aggressive mode.

interfaceIndex

Interface index.

jumboPrefer

Indicates whether this station prefers to support jumbo frames. Jumbo frames are defined in 9.2.1.

JUMBO—The station prefers to support jumbo frames.

REGULAR—(Otherwise.)

lastNeighborMac[ri]

MAC address of the last known neighbor station associated with the *ri* span.

macAddress

MAC address of the station.

managementAddressType

Management IP address type, as defined in 11.4.5.

managementIpAddr

Management IP address.

multichokeUser

Indicates whether this station expects to receive multi-choke fairness frames.

TRUE—This station expects to receive multi-choke fairness frames.

FALSE—(Otherwise.)

neighborCheck[ri]

Topology checksum information for the neighbor station on span *ri*, including the following:

value—checksum value

valid—an indication of whether the checksum is valid:

TRUE—The checksum is valid.

FALSE—(Otherwise.)

pircUser

Indicates whether the station is PIRC capable, based on the value of the *enablePirc* variable.

TRUE—The station is PIRC capable.

FALSE—(Otherwise.)

protConfig

Type of protection configuration.

STEERING—Indicates the station configured for steering.

WRAPPING—Indicates the station is configured for wrapping.

reservedRate[rid]

Reserved (subclassA0) bandwidth on ringlet *rid*.

sasUser

Indicates whether the station is SAS capable, based on the value of the *enableSas* variable.

TRUE—The station is SAS capable.

FALSE—(Otherwise.)

secMac[2]

Two secondary MAC address parameters, including the following:

address—secondary MAC address

state—secondary MAC state, indicating whether this address is in use:

NOT_USED—Secondary address is zero and thus not valid or used.

DUPLICATE—Secondary address is not unique and therefore not used.

OK_TO_USE—Secondary address is unique but not in use.

OK_IN_USE—Secondary address is unique and in use.

sequenceNumber

Sequence number used in transmitted TP frames.

spanEdge[ri]

Edge state associated with the *ri* span.

TRUE—An edge is present.

FALSE—(Otherwise.)

spanProtState[ri]

Protection state associated with the *ri* span of the station.

IDLE—None or an insignificant error rate.

MS—Indicates a desire to deactivate the span.

SD—A significant error rate indicating a degraded span (signal degrade).

SF—A unacceptable error rate indicating a span has failed (signal fail).

FS—Indicates a mandate to deactivate the span.

stationName

Station name.

weight[rid]

Station weight on ringlet *rid*.

11.2.6 *topoEntry[rid][hops]* fields

The topology database (see 11.5) is divided into *ringInfo*, *myTopoInfo*, and *topoEntry* portions as categorized in 11.2.4, 11.2.5, and 11.2.6. Each *topoEntry[rid][hops]* entry includes the following field values:

badFcsUser

Indicates whether a station copies data frames with invalid *fcs* fields to the client.

TRUE—Station copies data frames with invalid *fcs* fields to the client.

FALSE—(Otherwise.)

conservativeMode

Indicates whether a station is configured to run conservative mode.

TRUE—Station is configured to run conservative mode.

FALSE—The station is configured to run aggressive mode.

hops

The second index value for *topoEntry[][]*, representing the number of spans to the applicable station.

interfaceIndex

Interface index.

jumboPrefer

Indicates a station that prefers to support jumbo frames.

JUMBO—Indicates the station prefers to support jumbo frames.

REGULAR—(Otherwise.)

lrtt

Loop round-trip time to/from the station, derived from an LRTT frame received on ringlet *rid*.

macAddress

MAC address of the station.

managementAddressType

Management IP address type, as defined in 11.4.5.

managementIpAddr

Management IP address.

multichokeUser

Indicates whether a station expects to receive multi-choke fairness frames.

TRUE—Station expects to receive multi-choke fairness frames.

FALSE—(Otherwise.)

pircGroupMember

Indicates whether the station is PIRC capable, based on the value of the *enablePirc* variable.

TRUE—The station is PIRC capable.

FALSE—(Otherwise.)

protConfig

Type of protection configuration.

STEERING—Indicates a steering configured station.

WRAPPING—Indicates a wrapping configured station.

reachable

Provides an indication of whether this entry is reachable.

(This value is ignored when *topoEntry[rid][hops].valid* is FALSE.)

TRUE—Station is reachable.

FALSE—(Otherwise.)

reservedRate[rid]

Reserved subclassA0 bandwidth on ringlet *rid*.

sasUser

Indicates whether the station is SAS capable, based on the value of the *enableSas* variable.

TRUE—The station is SAS capable.

FALSE—(Otherwise.)

secMac[2]

Two secondary MAC address parameters, including the following:

address—secondary MAC address

state—secondary MAC state, indicating whether this address is in use:

NOT_USED—Secondary address is all ones and thus not valid or used.

DUPLICATE—Secondary address is not unique and therefore not used.

OK_TO_USE—Secondary address is unique but not in use.

OK_IN_USE—Secondary address is unique and in use.

sequenceNumber

Sequence number contained in TP frame received on ringlet *rid*.

spanEdge[ri]

Indicates whether an edge is associated with the *ri* span of the station.

TRUE—An edge is present.

FALSE—(Otherwise).

spanProtState[ri]

Protection state for the *ri* span of the station.

stationName

Station name.

valid

An indication of whether this entry is valid, based on a TP frame reception over a non-SF link.

TRUE—Station is valid.

FALSE—(Otherwise).

weight[rid]

Station weight on ringlet *rid*.

11.2.7 Common state machine routines

ChecksumStep(checksum, macAddress, sequenceNumber)

Updates the topology checksum for a single topology database entry, as specified by Equation (11.1).

```
uint32_t
ChecksumStep(checksum, macAddress, sequenceNumber) (11.1)
{
    uint64_t sum;

    sum = checksum & 0x00000000FFFFFFFFFF;
    sum += (macAddress >> 16);
    sum += ((macAddress & 0xFFFF) << 16) | sequenceNumber;
    sum = (sum & 0x00000000FFFFFF) + ((sum & 0xFFFFFFFF00000000) >> 32);
    sum = (sum & 0x00000000FFFFFF) + ((sum & 0xFFFFFFFF00000000) >> 32);
    return(sum);
}
```

ClearAtdInfo(rid, hops)

Clears all ATD information associated with this entry in the topology database.

ComputeTc()

Computes the topology database checksum of the local station , as specified by Equation (11.2).

```

uint32_t
ComputeTc()
{
    int ringlet, hops;
    uint32_t checksum = 0x00000000;

    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        if (ringlet == 0 || ringInfo.topoType == CLOSED_RING)
            checksum = ChecksumStep(checksum, myTopoInfo.macAddress,
                                     myTopoInfo.sequenceNumber);
        for (hops = 1; hops <= ringInfo.totalHopsRx[ringlet]; hops += 1)
        {
            if (topoEntry[ringlet][hops].valid)
                checksum = ChecksumStep(checksum,
                                         topoEntry[ringlet][hops].macAddress,
                                         topoEntry[ringlet][hops].sequenceNumber);
        }
    }
    return (checksum);
}

```

FindIndex(frame)

Determines the topology database index value corresponding to the received TP frame, as specified by Equation (11.3).

$$(MAX_STATIONS - frame.ttl + 1) \quad (11.3)$$
MismatchedProtection()

Indicates a presence of mismatched protection configuration stations, as specified by Equation (11.4).

TRUE—There is a station with a mismatched protection configuration on the ring.

FALSE—Otherwise.)

```

Boolean
MismatchedProtection()
{
    int ringlet, hops, protConfig;

    protConfig = myTopoInfo.protConfig;
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        if (myTopoInfo.spanEdge[ringlet])
            continue;
        for (hops = 1; hops <= MAX_STATIONS; hops += 1)
        {
            if (!topoEntry[ringlet][hops].valid ||
                topoEntry[ringlet][hops].macAddress == myTopoInfo.macAddress)
                break;
            if (topoEntry[ringlet][hops].protConfig != protConfig)
                return (TRUE);
        }
    }
    return (FALSE);
}

```

SdbPurgeUnreachable()

This function removes all entries from the SDB whose target address is unreachable. The *sdbTopologyPurges* counter is incremented by one for each invocation. See 14.4.3.

11.2.8 Variables and routines defined in other clauses

This clause references the following variables and routines defined in Clause 7:

Broadcast(macAddress)
currentTime
Dequeue(queue)
DequeueRi(ri, queue)
Enqueue(queue, frame)
EnqueueRi(rid, queue, frame)
EnqueueRi(ringlet, queue, frame)
EntryInQueue(queue)
Max(valueA, valueB)
NULL
Other(ri)
Q_RX_ATD
Q_RX_LRTT_REQ
Q_RX_LRTT_RSP
Q_RX_TC
Q_RX_TP
Q_TX_RS
sourceCheck[rid][hops]

This clause references the following variables defined in Clause 8:

DEGRADE_FAIL
LINK_STATUS[*ri*]
SIGNAL_DEGRADE
SIGNAL_FAIL

This clause references the following literals defined in Clause 9:

CLASS_A0
CT_LRTT_REQ
CT_LRTT_RSP
CT_STATION_ATD
CT_TOPO_CHKSUM
CT_TOPO_PROT
JUMBO_MAX
REGULAR_MAX

This clause references the following literals, variables, and routines defined in Clause 10:

advertisingInterval
LINK_RATE

11.2.9 Defect indications

duplicateSecMacAddressDefect

A critical severity defect indicating the local station has one or more duplicate secondary MAC addresses (see 11.5.2).

TRUE—A duplicate secondary MAC address defect is present.
FALSE—(Otherwise.)

excessReservedRateDefect

This defect indicates that the amount of reserved bandwidth on a ringlet exceeds the available LINK_RATE.

TRUE—An excess reserved rate defect is present.

FALSE—(Otherwise.)

lrttIncompleteDefect

A critical severity defect indicating the LRTT measurement for the ring has not been completed within the *lrttTimeout* specified time (see 11.6.14.4).

TRUE—An LRTT incomplete defect is present.

FALSE—(Otherwise.)

maxSecMacAddressDefect

A critical severity defect indicating more than MAX_SEC_MAC secondary MAC addresses (see 11.5.2) are present.

TRUE—More than MAX_SEC_MAC secondary MAC addresses are present.

FALSE—(Otherwise.)

maxStationsDefect

A critical severity defect indicating more than MAX_STATIONS stations are present.

TRUE—More than MAX_STATIONS stations are present.

FALSE—(Otherwise.)

miscablingDefect[ri]

The critical severity miscabling defect associated with the *ri*-selected span.

TRUE—A miscabling defect is present on the *ri*-selected span.

FALSE—(Otherwise.)

pircLbDefect

A critical severity defect indicating the PIRC protection group members have conflicting PIRC load balancing provisioning.

TRUE—A conflicting PIRC load balancing provisioning defect is present.

FALSE—(Otherwise.)

pircPrDefect

A critical severity defect indicating the PIRC protection group members have conflicting PIRC priority protection group provisioning.

TRUE—A conflicting PIRC priority protection group provisioning defect is present.

FALSE—(Otherwise.)

protMisconfigDefect

A critical severity defect that indicates the presence of mismatched-protection-configuration stations, based on the value returned by *MismatchedProtection()*.

TRUE—A protection configuration defect is present on the station.

FALSE—(Otherwise.)

topoEntryInvalidDefect

A critical severity defect indicating that an invalid entry has been found within the scope of the topology.

TRUE—A topology entry invalid defect is present.

FALSE—(Otherwise.)

topoInconsistencyDefect

The critical severity inconsistent topology defect (see 11.6.6.1).

TRUE—An inconsistent topology defect is present.

FALSE—(Otherwise.)

topoInstabilityDefect

The critical severity topology instability defect.

TRUE—A topology instability defect is present.

FALSE—(Otherwise.)

11.3 Frame formats

11.3.1 Topology and protection (TP) frame format

The fixed-length TP frame is identified by the *controlType* field value of CT_TOPO_PROT and is broadcast to minimize transmission delays. The TP frame contents include information for signaling span protection state, for discovery of the physical topology of the ring, and for reporting station preferences information, as specified in Figure 11.6.

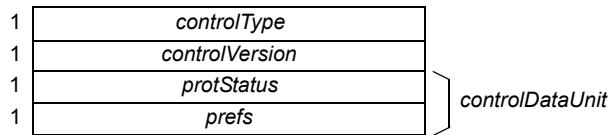


Figure 11.6—TP payload

The header fields (specified in 9.3, but not illustrated here) are additionally constrained as specified in Table 11.1.

Table 11.1—TP frame header field-value restrictions

Field	Subfield	Value	Description
<i>ttl</i>	—	MAX_STATIONS	Allows TP frames to propagate through all stations
<i>baseControl</i>	<i>sc</i>	CLASS_A0	Sent as subclassA0
	<i>fe</i>	FALSE	Not fairness eligible
	<i>we</i>	0	These frames are never wrapped
<i>da</i>	—	FF-FF-FF-FF-FF-FF	Allows TP frames to be received by any station

11.3.1.1 *protStatus*: An 8-bit field specified in 11.3.1.3.

11.3.1.2 *prefs*: An 8-bit field specified 11.3.1.4.

11.3.1.3 *protStatus*

The subfields of the 8-bit *protStatus* (protection status) field are shown in Figure 11.7 and described thereafter.

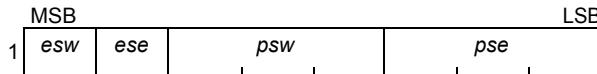


Figure 11.7—*protStatus* field format

11.3.1.3.1 *esw*: A (edge state, west) bit that indicates whether an edge is present on the west span of a station. A value of 0 indicates that there is no edge, whereas a value of 1 indicates that an edge is present.

11.3.1.3.2 *ese*: A (edge state, east) bit that indicates whether an edge is present on the east span of a station. A value of 0 indicates that there is no edge, whereas a value of 1 indicates that an edge is present.

11.3.1.3.3 *psw*: A 3-bit (protection state, west) field indicates the protection state on the west span of the station, as defined in the right columns of Table 11.2.

11.3.1.3.4 *pse*: A 3-bit (protection state, east) field indicates the protection state on the east span of the station, as defined in the right columns of Table 11.2.

Table 11.2—*psw* and *pse* values

Value	Name	Description
000_2	IDLE	No request
001_2	WTR	Wait-to-restore
010_2	MS	Manual switch
011_2	SD	Signal degrade
100_2	SF	Signal fail
101_2	FS	Forced switch
110_2	—	Reserved
111_2	—	Reserved

11.3.1.4 *prefs*

The subfields of the 8-bit *prefs* (preferences) field are illustrated in Figure 11.8 and described thereafter.

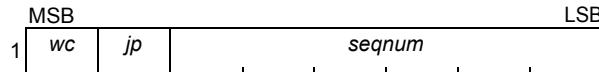


Figure 11.8—*prefs* field format

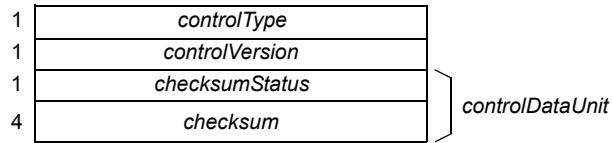
11.3.1.4.1 *wc*: A (wrap protection configured) bit that is set based on the protection configuration stored in *myTopoInfo.protConfig*, as defined in 11.2.5.

11.3.1.4.2 *jp*: A (jumbo frame preferred) bit that is set based on the jumbo-frame preference stored in *myTopoInfo.jumboPrefer*, as defined in 11.2.5.

11.3.1.4.3 *seqnum*: A 6-bit (sequence number) field that is incremented each time other portions of the *controlDataUnit* have changed, as specified in 11.6.7.

11.3.2 Topology checksum (TC) frame format

The fixed-length broadcast TC frames have *controlType* equal to CT_TOPO_CHKSUM. These TC frames communicate the topology checksum [defined by Equation (11.2)] to neighbor stations so that the stations can determine whether the relevant portions of their topology databases match and, therefore, whether they can exit context containment, as specified in Figure 11.9.

**Figure 11.9—TC payload**

The header fields (specified in 9.3, but not illustrated here) are additionally constrained as specified in Table 11.3. A *ttl* field value of 1 forces downstream stripping, so these frames never travel beyond the neighbor station.

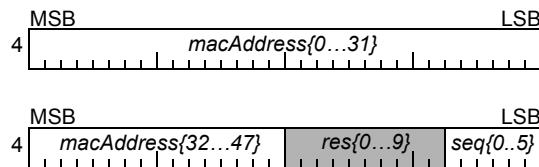
Table 11.3—TC frame header field-value restrictions

Field	Subfield	Value	Description
<i>ttl</i>	—	1	Ensures TC frames do not propagate beyond neighbor station
<i>baseControl</i>	<i>sc</i>	CLASS_A0	Sent as subclassA0
	<i>fe</i>	FALSE	Not fairness eligible
	<i>we</i>	0	These frames are never wrapped
<i>da</i>	—	FF-FF-FF-FF-FF-FF	Allows TC frames to be received by neighbor station

11.3.2.1 *checksumStatus*: An 8-bit field defined in 11.3.2.3.

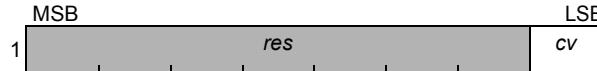
11.3.2.2 *checksum*: A 32-bit field shown in Figure 11.9. The value in this field is a checksum computed from a subset of fields in the topology database, as described in 11.2.

A checksum computation involves one step for each of the configured stations. Two 32-bit numbers are added in each of these checksum computation steps [see Equation (11.1)]. The first number consists of the most significant 4 bytes of the source MAC address for that entry. The second number (in most significant to least significant bit order) consists of the two least significant bytes of the source MAC address, a 10-bit reserved field, and the 6-bit sequence number, as illustrated in Figure 11.10.

**Figure 11.10—Alignment of the 32-bit fields used to compute topology checksum**

11.3.2.3 *checksumStatus*

The 8-bit *checksumStatus* (checksum status) field provides a bit that indicates when the associated checksum is valid, as illustrated in Figure 11.11.

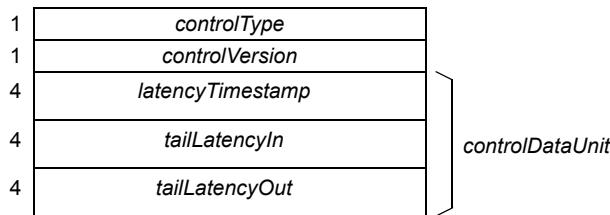
**Figure 11.11—checksumStatus field format**

11.3.2.4 res: A 7-bit (reserved) field.

11.3.2.5 cv: A (checksum valid) bit that indicates if the checksum field is valid. A value of 1 indicates the checksum is valid, and a value of 0 indicates the checksum is invalid as specified by Equation (11.12).

11.3.3 Loop round-trip time request frame format

The fixed-length loop round-trip time (LRTT) request control frames are identified by the *controlType field* value of CT_LRTT_REQ. Stations using conservative rate adjustment shall generate loop round-trip time request frames. All stations shall perform loop round-trip time request frame receipt processing. The loop round-trip time request frame has a 14-byte payload, as specified in Figure 11.12.

**Figure 11.12—LRTT request payload**

The header fields (specified in 9.3, but not illustrated here) are additionally constrained as specified in Table 11.4.

Table 11.4—LRTT request frame header field-value restrictions

Field	Subfield	Value	Description
baseControl	sc	CLASS_A0	Sent as subclass A0
	fe	FALSE	Not fairness eligible
	we	0	These frames are never wrapped
da	—	(depends)	Unicast address of the station whose LRTT is being measured

11.3.3.1 latencyTimestamp: A 32-bit field that contains a local timestamp generated by the requestor at the time of transmitting this frame. The interpretation of this field is local to the requestor and has no meaning to other stations.

11.3.3.2 tailLatencyIn: A 32-bit field provided for the responder's use when reflecting the frame as a response frame, initialized to zero.

11.3.3.3 tailLatencyOut: A 32-bit field provided for the responder's use when reflecting the frame as a response frame, initialized to zero.

11.3.4 Loop round-trip time response frame format

LRTT response control frames have *controlType* equal to CT_LRTT_RSP. Stations respond to each LRTT request by sending an LRTT response. An LRTT response frame has a 14-byte payload, as specified in Figure 11.13.

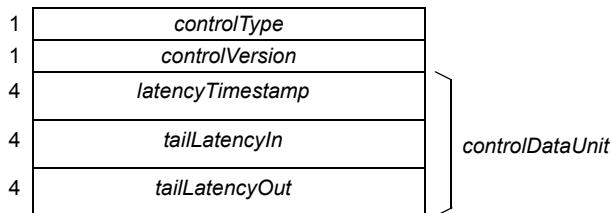


Figure 11.13—LRTT response payload

The header fields (specified in 9.3, but not illustrated here) are additionally constrained as specified in Table 11.5.

Table 11.5—LRTT response frame header field-value restrictions

Field	Subfield	Value	Description
<i>baseControl</i>	<i>sc</i>	CLASS_A0	Sent as subclassA0
	<i>fe</i>	FALSE	Not fairness eligible
	<i>we</i>	0	These frames are never wrapped
<i>da</i>	—	<i>sa</i> from request	The unicast address of the station measuring the LRTT

11.3.4.1 *latencyTimestamp*: A copy of the like-named 32-bit field from the last received loop round-trip time request frame.

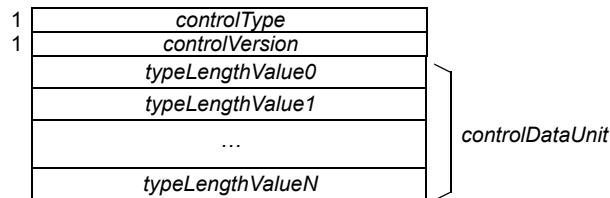
11.3.4.2 *tailLatencyIn*: A 32-bit field provided for a responding station to mark the time at which it received the corresponding LRTT request frame. If a timestamp is placed herein, that value is specified in units of microseconds. If a station is unable to timestamp the frame, this field value shall be cleared to zero.

11.3.4.3 *tailLatencyOut*: A 32-bit field provided for a responding station to mark the time at which it transmits the LRTT response frame. If a *tailLatencyIn* timestamp is provided, this value is specified in units of microseconds. Otherwise, this value shall be cleared to zero.

NOTE—Although the latency is best kept as close to zero as possible, and the measurement or estimate of the latency is best kept as accurate as possible, estimations that overestimate the latency yield better fairness response than estimations that underestimate the latency.

11.3.5 ATD frame format

The variable-length ATD control frames are identified by the *controlType* field value of CT_STATION_ATD. In comparison with TP frames, these ATD frames contain less time-critical information and are sent less often. The ATD frame's payload provides type-length-value (TLV) encoded parameters to other stations, as specified in Figure 11.14.

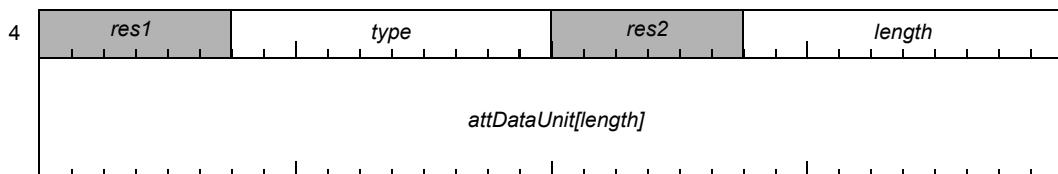
**Figure 11.14—Attribute discovery payload**

The header fields (specified in 9.3, but not illustrated here) are additionally constrained as specified in Table 11.6.

Table 11.6—ATD frame header field-value restrictions

Field	Subfield	Value	Description
<i>ttl</i>	—	MAX_STATIONS	Allows ATD frames to propagate through all stations
<i>baseControl</i>	<i>sc</i>	CLASS_A0	Sent as subclassA0
	<i>fe</i>	FALSE	Not fairness eligible
	<i>we</i>	0	These frames are never wrapped
<i>da</i>	—	FF-FF-FF-FF-FF-FF	Allows ATD frames to be received by any station

A type-length-value encoding scheme is used to encode additional station attributes (ATTs) in ATD frames, as specified in Figure 11.15. Within an ATD frame, the ATTs shown in Table 11.7 can appear in any order.

**Figure 11.15—Type-length-value format**

11.3.5.1 *res1*: A 6-bit reserved field.

11.3.5.2 *type*: A 10-bit field that identifies the ATT type, as specified in Table 11.7.

Table 11.7—type encoding for ATD frame

Value	Name	Size	Reference	Support	Description
0	—	—	—	—	Reserved
1	ATT_WEIGHT	2	11.4.1	C	Fairness weights of stations
2	ATT_STATION_BW	4	11.4.2	C	Reserved classA0 bandwidth

Table 11.7—type encoding for ATD frame (continued)

Value	Name	Size	Reference	Support	Description
3	ATT_STATION_SET	1	11.4.3	C	Station settings
4	ATT_STATION_NAME	0-127	11.4.4	C	Station name (an ASCII string)
5	ATT_MANAGE_ADDRESS	5 or 17	11.4.5	C	Management IP address
6	ATT_INTERFACE_INDEX	4	11.4.6	C	<i>ifIndex</i> of the ring interface
7	ATT_SECONDARY_MACS	12	11.4.7	C	Secondary MAC addresses
8-1022	—	—	—	—	Reserved
1023	ATT_ORG_SPECIFIC	4-1023	11.4.8	O	Nonstandard (organization-specific) values

Legend:

O—Optional

C—Conditional (mandatory if value is different from default value, otherwise optional)

11.3.5.3 res2: A 6-bit reserved field.**11.3.5.4 length:** The length, in bytes, of the following *typeDependent* value information. The value of *length* shall be less than 1024.**11.3.5.5 attDataUnit:** Data whose format and function is specified by the preceding *type* field. The length of this field is less than 1024 bytes.

If a station has a non-default ATT value, it shall advertise those values in the ATT component of each ATD frame. If a station has default ATT values, it may advertise those values in the ATT component of each ATD frame. If a conditionally supported ATT (see Table 11.7) is not included in the ATD frames, then the source station's default ATT component values shall be restored.

11.4 Defined ATT encodings

11.4.1 Weight ATT

The weight ATT is identified by its distinctive *type* field value of ATT_WEIGHT (see 11.3.5). The weight ATT encodes the station configurable fairness weight *myTopoInfo.weight[rid]* on each ringlet (see 11.2.5), as specified in Figure 11.16. The consumer of the information contained in the weight ATT is the fairness module, which needs this information to compute the sum of the weights of active stations in 10.3.2.

**Figure 11.16—Weight attDataUnit format****11.4.1.1 ringlet0Weight:** Specifies the fairness weight of the station on ringlet0; the default value is 1.**11.4.1.2 ringlet1Weight:** Specify the fairness weight of the station on ringlet1; the default value is 1.

When received, this information is copied to the *weight[rid]* field in the topology database (see 11.5).

11.4.2 Station bandwidth ATT

The station bandwidth ATT is identified by its distinctive *type* field value of ATT_STATION_BW (see 11.3.5). The station bandwidth ATT encodes the source station's reserved subclassA0 transmit bandwidth on each ringlet, as specified in Figure 11.17.



Figure 11.17—Station bandwidth *attDataUnit* format

11.4.2.1 *ringlet0ReservedBw*: A 16-bit field that specifies the station's reserved subclassA0 bandwidth allocation on ringlet0, normalized as per Equation 11.5. The default value is zero.

$$\text{ringlet0ReservedBw} = (\text{reservedRate}[ri] * 10^{**6} * \text{agingInterval}) / (\text{rateCoef} * 8 \text{ bits/byte}) \quad (11.5)$$

11.4.2.2 *ringlet1ReservedBw*: A 16-bit field that specifies the station's reserved subclassA0 bandwidth allocation on ringlet1, normalized as per Equation 11.6. The default value is zero.

$$\text{ringlet1ReservedBw} = (\text{reservedRate}[ri] * 10^{**6} * \text{agingInterval}) / (\text{rateCoef} * 8 \text{ bits/byte}) \quad (11.6)$$

When received, this information is copied to the *reservedRate[rid]* field in the topology database (see 11.5).

11.4.3 Station settings ATT

The station settings ATT is identified by its distinctive *type* field value of ATT_STATION_SET (see 11.3.5). The station settings ATT encodes several station settings, as specified in Figure 11.18. When received, this information is copied to the *pircGroupMember*, *sasUser*, *badFcsUser*, *conservativeMode*, and *multichokeUser* fields in the topology database (see 11.5).

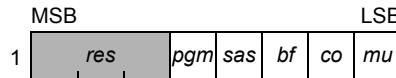


Figure 11.18—Station settings *attDataUnit* format

11.4.3.1 *res*: A 3-bit reserved field.

11.4.3.2 *pgm*: A (PIRC group member) bit that is set to 1 if the station is a member of a PIRC protection group. Otherwise, this bit has a default 0 value.

11.4.3.3 *sas*: A (spatially aware sublayer) bit that is set to 1 if the SAS sublayer is enabled in the station, based on the value of *myTopoInfo.sasUser* defined in 11.2.5. Otherwise, this bit has a default 0 value.

11.4.3.4 *bf*: A (bad frame) bit that is set to 1 if the station sends data frames with invalid *fcs* fields to the client, based on the value of *myTopoInfo.badFcsUser* defined in 11.2.5. Otherwise, this bit has a default 0 value.

The *ringInfo.badFcsUser* bit indicates if any station on the ring copies data frames with invalid *fcs* fields to the client and, therefore, that every station transmits data frames with invalid *fcs*.

11.4.3.5 *co*: A (conservative option) bit that is set to 1 if the station's *conservativeMode* bit is configured to use conservative mode fairness, based on the value of *myTopoInfo.conservativeMode* defined in 11.2.5. Otherwise, this bit has a default 0 value. (The tail station of a congestion domain sends FDD frames if the head station's *co* bit value is advertised as 1.)

11.4.3.6 *mu*: A (multi-choke update) bit that is set to 1 if the station expects to receive multi-choke fairness frames from all other stations on the ring, based on the value of *myTopoInfo.multichokeUser* defined in 11.2.5. Otherwise, this bit has a default 0 value. The *ringInfo.multichokeUser* bit indicates if any station on the ring uses multi-choke fairness frames and, therefore, that every station transmits multi-choke fairness frames.

11.4.4 Station name ATT

The variable-length station name ATT is identified by its distinctive *type* field value of ATT_STATION_NAME (see 11.3.5). The station name ATT encodes the first 127 characters of the operator assigned *stationName* value, as specified in Figure 11.19.

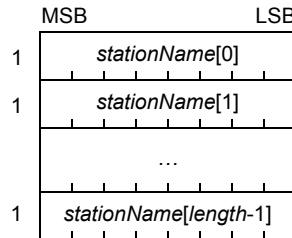


Figure 11.19—Station name *attDataUnit* format

stationName: A sequence of 8-bit fields that identify the first through last ASCII characters corresponding to the station's name (a NULL character termination is not provided). The default value is zero.

The *stationName* field follows the definition of the RFC 3418 sysName object.

When received, this information is copied to the *stationName* field in the topology database (see 11.5).

11.4.5 Management address ATT

The variable-length management address ATT is identified by its distinctive *type* field value of ATT_MANAGE_ADDRESS (see 11.3.5). The management address ATT encodes the internet protocol (IP) address of the management interface of the station, as specified in Figure 11.20.

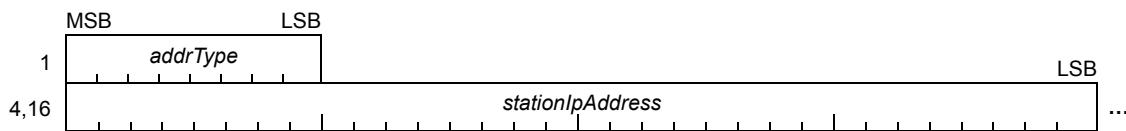


Figure 11.20—Management address ATT *attDataUnit* format

11.4.5.1 *addrType*: An 8-bit field that carries the Internet address type identifier of the following IP address, as defined in RFC 3291. The mapping of these values to supported address types is shown in Table 11.8.

Table 11.8—Mapping of address type values to supported address types

Value	Name	Address length	Description
1	<i>ipv4</i>	4	Internet Protocol version 4 address
2	<i>ipv6</i>	16	Internet Protocol version 6 address

11.4.5.2 *stationIpAddress*: The 32-bit or 128-bit *stationIpAddress* field carries the IP address of the management interface of the station. If the *managementAddressType* field indicates an IPv4 address, *stationIpAddress* is a 32-bit field. If the *managementAddressType* field indicates an IPv6 address, *stationIpAddress* is a 128-bit field. If a station management address ATT is not included in the ATD frames sent by a given source station, then the station management address is assumed to have a default value of all zeros by receiving stations.

The station management address ATT need not be unique for every station. A single managed entity may be attached to a ring multiple times.

When received, this information is copied to the *managementAddressType* and *managementIpAddr* fields in the topology database (see 11.5).

11.4.6 Station interface index ATT

The station interface index ATT is identified by its distinctive *type* field value of ATT_INTERFACE_INDEX (see 11.3.5). The station interface index ATT encodes the *rprIfIndex* attribute from the RPR MIB and is illustrated in Figure 11.21.

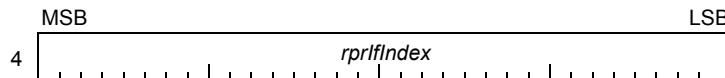


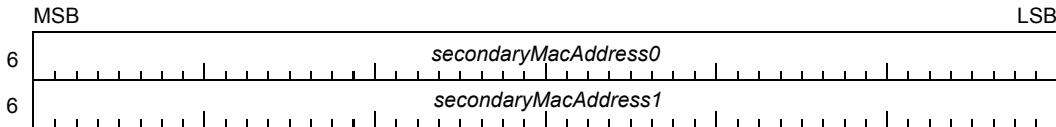
Figure 11.21—Station interface index ATT *attDataUnit* format

11.4.6.1 *rprIfIndex*: A 32-bit field that contains the RFC 2863 *ifIndex* value of the RPR interface of a station. The default value is zero.

When received, this information is copied to the *interfaceIndex* field in the topology database (see 11.5).

11.4.7 Secondary MAC ATT

The secondary MAC ATT is identified by its distinctive *type* field value of ATT_SECONDARY_MACS (see 11.22). This ATT supplies two fields, each of which contains either a 48-bit unicast secondary MAC address, or the NOT_USED (all ones) value, as specified in Figure 11.22.

**Figure 11.22—Secondary MAC ATT attDataUnit format**

11.4.7.1 *secondaryMacAddress0*: A 48-bit field that identifies the first secondary MAC address of a station. If no secondary MAC address is configured, *secondaryMacAddress0* shall default to the broadcast address.

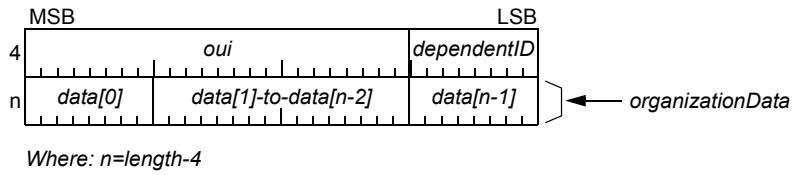
11.4.7.2 *secondaryMacAddress1*: A 48-bit field that identifies the final secondary MAC address of a station. Unless two secondary MAC address are configured, *secondaryMacAddress1* shall default to the broadcast address.

When received, this information is copied to the *secMac[2]* fields in the topology database (see 11.5).

11.4.8 Organization-specific ATT

The variable-length organization-specific ATT is identified by its distinctive *type* value of ATT_ORG_SPECIFIC (see 11.3.5). The organization-specific ATT may be included in the ATD frame, allowing this information to be reported to all other stations on the RPR ring. The actions taken as a result of receiving or not receiving the information contained in the organization-specific ATT are outside the scope of this standard.

The organization-specific ATT encodes a collection of organization-specific parameters, as specified in Figure 11.23.

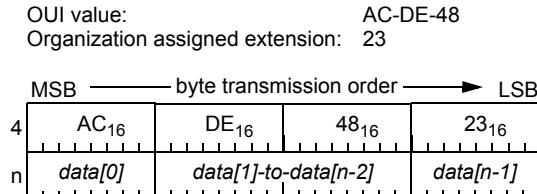
**Figure 11.23—Organization-specific attDataUnit format**

11.4.8.1 *oui*: A 24-bit organizationally unique identifier (OUI) field supplied by the IEEE/RAC for the purpose of identifying the organization supplying the (unique within the organization, for this specific context) 8-bit *dependentID*.

11.4.8.2 *dependentID*: An 8-bit field supplied by the *oui*-specified organization. The concatenation of the *oui* and *dependentID* provide a unique (within this context) identifier for each distinct form of the following *organizationalData*.

11.4.8.3 *data*: A sequence of bytes whose format and function is dependent on the preceding *oui*/*dependentID* identifiers. The content of this *data* is beyond of the scope of this standard.

To reduce the likelihood of error, the mapping of OUI values to the *oui/dependentID* fields are illustrated in Figure 11.24. For the purposes of illustration, specific OUI and *dependentID* values have been assumed.

**Figure 11.24—Mapping of organization-specific values**

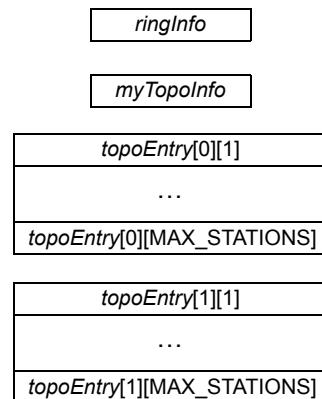
11.5 Topology database

11.5.1 Topology database structure

11.5.1.1 Logical representation of topology database

The topology database is partitioned into multiple components, as described in Figure 11.25 and listed below.

- a) Ring-level information (see Table 11.9): *ringInfo.fieldname*
- b) Local station information (see Table 11.10): *myTopoInfo.fieldname*
- c) Per station TP, TC, LRTT, and ATD information (see Table 11.11)
 - topoEntry[0][hops].fieldname*, for information received from ringlet0
 - topoEntry[1][hops].fieldname*, for information received from ringlet1.

**Figure 11.25—Topology database structure**

The correspondence between indexing per ringlet and the east/west span nomenclature used in the MIB is described in 11.2.3.

The information used to configure values in the topology database comes from a variety of sources:

- a) TP frames (see 11.3.1) are used to update:
 - 1) Per-ringlet station MAC addresses
 - 2) Per-ringlet hop-count information

- 3) Edge state
- 4) Protection state
- 5) Station capabilities
- b) TC frames (see 11.3.2) update the topology checksum.
- c) LRTT frames update the loop round-trip time.
- d) ATD frames update other elements in the topology database

The purpose of explicitly showing downstream data reachability in the topology database is to provide a clear indication of stations that are downstream data reachable from a given station, while enabling the complete physical connectivity of the ring topology to be shown. Data reachability is derived as described in 11.5.3.

The topology database contains ring-level information (see Table 11.9), station-local information (see Table 11.10), and information associated with each of the other stations (see Table 11.11).

Table 11.9—Ring-level topology database: *ringInfo*

Description	Variable	Example value
TP-derived information		
ring jumbo type	<i>jumboType</i>	REGULAR
MTU size	<i>mtuSize</i>	REGULAR_MAX
number of stations on ring	<i>numStations</i>	4
receive hops on ringlet0	<i>totalHopsRx[0]</i>	3
receive hops on ringlet1	<i>totalHopsRx[1]</i>	3
ring topology type	<i>topoType</i>	CLOSED_RING
transmit hops on ringlet0	<i>totalHopsTx[0]</i>	3
transmit hops on ringlet1	<i>totalHopsTx[1]</i>	3
ATD-derived information		
Station receiving frames with bad <i>fcs</i>	<i>badFcsUser</i>	FALSE
Station using multi-choke fairness	<i>multichokeUser</i>	FALSE
Available bandwidth on ringlet0	<i>unreservedRate[0]</i>	800
Available bandwidth on ringlet1	<i>unreservedRate[1]</i>	700

Table 11.10—Topology database for local station: *myTopoInfo*

Description	Variable	Example value
TP-derived information		
local station MAC address	<i>macAddress</i>	00-10-A4-97-B7-DE
station jumbo preference	<i>jumboPrefer</i>	REGULAR
station protection configuration	<i>protConfig</i>	WRAPPING
west span protection state	<i>spanProtState[0]</i>	IDLE
east span protection state	<i>spanProtState[1]</i>	IDLE
west span edge state	<i>spanEdge[0]</i>	FALSE
east span edge state	<i>spanEdge[1]</i>	FALSE
sequence number	<i>sequenceNumber</i>	3
west last known neighbor MAC address	<i>lastNeighborMac[0]</i>	00-10-A4-97-A8-DE
east last known neighbor MAC address	<i>lastNeighborMac[1]</i>	00-10-A4-97-A8-BD
topology checksum	<i>myChecksum.value</i>	AA-11-BB-22
topology checksum valid	<i>myChecksum.valid</i>	TRUE
TC-derived information		
west neighbor topology checksum	<i>neighborCheck[0].value</i>	AA-11-BB-22
west neighbor topology checksum valid	<i>neighborCheck[0].valid</i>	TRUE
east neighbor topology checksum	<i>neighborCheck[1].value</i>	AA-11-BB-22
east neighbor topology checksum valid	<i>neighborCheck[1].valid</i>	TRUE
ATD-derived information		
station name	<i>stationName</i>	San Francisco
1st secondary MAC address	<i>secMac[0].address</i>	0
1st secondary MAC address state	<i>secMac[0].state</i>	NOT_USED
2nd secondary MAC address	<i>secMac[1].address</i>	0
2nd secondary MAC address state	<i>secMac[1].state</i>	NOT_USED
weight on ringlet0	<i>weight[0]</i>	1
weight on ringlet1	<i>weight[1]</i>	5
reserved bandwidth on ringlet0	<i>reservedRate[0]</i>	200
reserved bandwidth on ringlet1	<i>reservedRate[1]</i>	160
receives frames with bad <i>fcs</i>	<i>badFcsUser</i>	FALSE
station is member of a PIRC protection group	<i>pircGroupMember</i>	TRUE

Table 11.10—Topology database for local station: *myTopoInfo* (continued)

Description	Variable	Example value
SAS is enabled	<i>sasUser</i>	TRUE
multi-choke fairness frames expected	<i>multichokeUser</i>	FALSE
uses conservative mode	<i>conservativeMode</i>	TRUE
management address type	<i>managementAddressType</i>	IPV4
management IP address	<i>managementIpAddr</i>	192.168.1.50
interface index	<i>interfaceIndex</i>	2

Table 11.11—Topology database for one ringlet: *topoEntry[ringlet][hops]*

Variable	Example values		
	hops = 1	hops = 2	hops = 3
TP-derived information			
<i>macAddress</i>	00-10-A4-97-A8-DE	00-10-A4-97-A8-EF	00-10-A4-97-A8-BD
<i>valid</i>	TRUE	TRUE	TRUE
<i>reachable</i>	TRUE	TRUE	TRUE
<i>jumboPrefer</i>	REGULAR	JUMBO	JUMBO
<i>protConfig</i>	WRAPPING	WRAPPING	WRAPPING
<i>spanProtState[0]</i>	IDLE	IDLE	IDLE
<i>spanProtState[1]</i>	IDLE	IDLE	IDLE
<i>spanEdge[0]</i>	FALSE	FALSE	FALSE
<i>spanEdge[1]</i>	FALSE	FALSE	FALSE
<i>sequenceNumber</i>	3	15	5
ATT-derived information			
<i>stationName</i>	Los Angeles	Denver	Portland
<i>secMac[0].address</i>	00-10-A4-97-B7-AB	00-10-A4-97-D4-DE	0
<i>secMac[0].state</i>	OK_IN_USE	OK_IN_USE	NOT_USED
<i>secMac[1].address</i>	00-10-A4-97-B7-CD	00-10-A4-97-B7-EF	0
<i>secMac[1].state</i>	OK_IN_USE	OK_IN_USE	NOT_USED
<i>weight[0]</i>	1	3	1
<i>weight[1]</i>	5	3	5
<i>reservedRate[0]</i>	50	100	200
<i>reservedRate[1]</i>	40	80	160
<i>badFcsUser</i>	FALSE	FALSE	FALSE

Table 11.11—Topology database for one ringlet: *topoEntry[ringlet][hops]* (continued)

Variable	Example values		
	hops = 1	hops = 2	hops = 3
<i>pircGroupMember</i>	TRUE	FALSE	TRUE
<i>sasUser</i>	TRUE	FALSE	TRUE
<i>multichokeUser</i>	FALSE	FALSE	FALSE
<i>conservativeMode</i>	TRUE	FALSE	TRUE
<i>managementAddressType</i>	IPV4	IPV4	IPV4
<i>managementIpAddr</i>	10.0.0.1	10.0.0.2	192.168.1.100
<i>interfaceIndex</i>	12	3	2
LRTT-derived information			
<i>lrtt</i>	10	20	30

11.5.2 Attribute updates

The topology database is modified upon receipt of TP, TC, LRTT, or ATD frames that result in a change in information contained in the database, by the protection state machine, or based on input from the protection state machine that results in further updates to the database. Changes in TC, LRTT, or ATD frame parameters, or by the protection state machine, modify fields within the database, but not the apparent position of stations in the topology.

The following rules apply to updates of the topology database due to receipt of ATTs other than organization-specific ATTs:

- a) The ATD-frame information is retained until replaced by new ATD frame information, or cleared by the absence of an ATT within the ATD.
- b) The ATD-frame information is cleared whenever
 - 1) The associated entry is marked invalid.
 - 2) The MAC address of an entry is overwritten with a new value (as happens due to passthrough).
- c) When a ATD frame is received, the information contained in the ATTs within this frame is used to update the topology database.
- d) All ATT information received from a station is included in that station's topology database entry.

11.5.3 Lower level representation of topology database

The MAC has topology database components illustrated in Figure 11.26. The index associated with this database represents the distance-to-source hops on ringlet0 or on ringlet1, derived from TP frames received on ringlet1 and ringlet0, respectively. The index value represents the hop-count distance to each station and is derived from the *frame.ttl* field value within received TP frames.

NOTE—Figure 11.26 is provided strictly for explanatory, illustrative, and representational purposes. It is not intended to depict a specific implementation or to constrain implementers. In particular, the widths of the fields in the figure are not shown to scale.

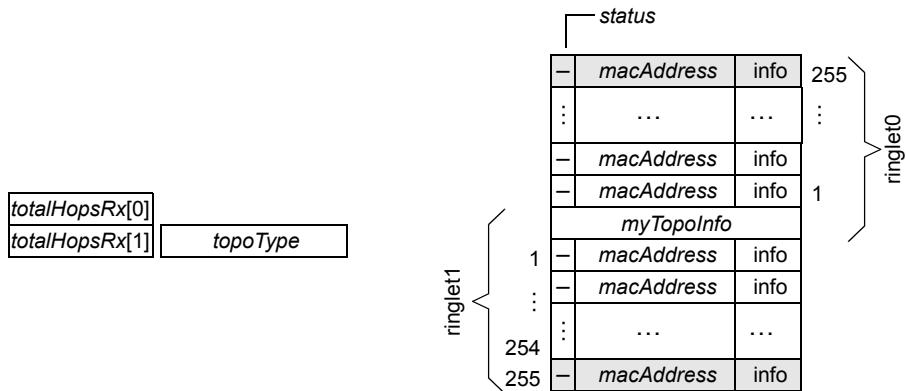


Figure 11.26—Topology database parameters

The *totalHopsRx[0]* and *totalHopsRx[1]* fields indicate the number of upstream stations on ringlet0 and ringlet1. The *topoType* field indicates whether the topology is a closed ring or an open ring.

Within each array, the *status* field indicates whether a topology database entry is considered valid and/or reachable and is composed of *valid* and *reachable* fields. The *valid* entry is set to invalid (mainly for all stations downstream of an edge) or valid (for neighbor stations across a non-SF edge). The *reachable* field is set to reachable (for valid stations that can be reached without crossing any edges), and to nonreachable otherwise.

The *macAddress* field is the source station MAC address within the received TP frame. The *info* column represents the remainder of the information contained in the topology database entry *topoEntry[rid][hops]*. The information for the local station is contained in *myTopoInfo*.

11.5.4 Topology change sequence

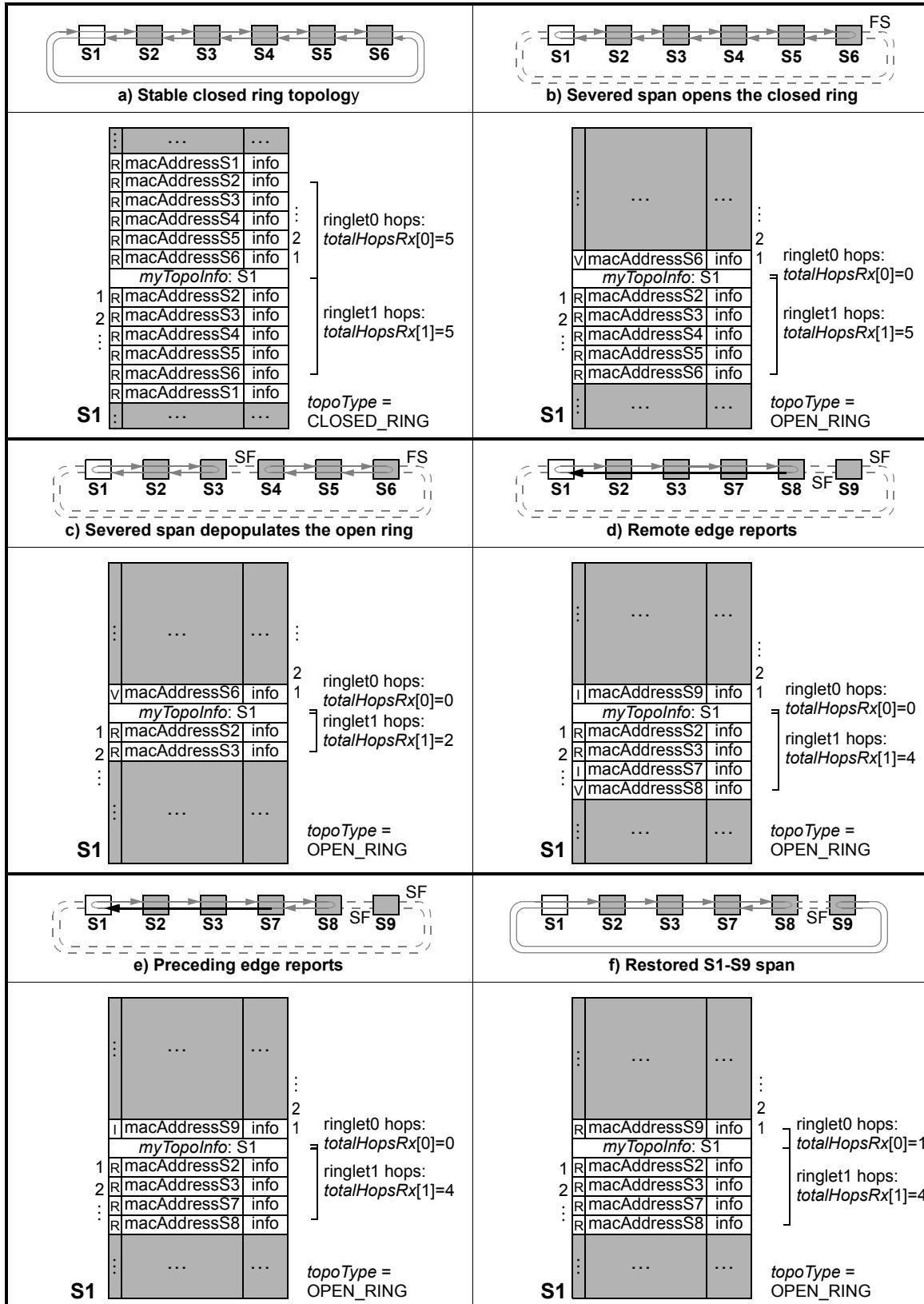
A valid, stable, closed-ring topology (see Figure 11.27-a) has no more than MAX_STATIONS, and redundant topology database entries are repeated on the ends of the closed ring. Within this and associated figures, the valid/reachable, valid/not-reachable, and not-valid entries are marked R, V, and I, respectively.

In an open-ring topology (see Figure 11.27-b), each side of the open ring terminates where an edge is detected. If the edge span remains operational (such as after a FS, MS, or SD), the edge-adjacent stations mark their neighbors valid (but not reachable). The intent is to facilitate across-edge communications, so that a non-operational span can be restored.

Two non-operational spans (see Figure 11.27-c) further restrict the database knowledge to a smaller open ring. For an SF edge, the edge-adjacent stations mark their neighbors invalid, but save the contents of communicated TP frames. The intent is to disable actions based on possibly corrupted SF-link communications, while saving the TP frame contents for diagnostic purposes.

When joining open-ring segments, distant stations can send TP frames and be marked valid (see Figure 11.27-d) before closer stations, yielding invalid intermediate entries. TP frame transmissions from closer stations eventually validate those inconsistent entries (see Figure 11.27-e).

Recovery of the S9-S1 span (see Figure 11.27-f) adds another station to the open-ring topology. Although S8 and S9 can retain entries for their across-edge neighbors, station S1 has no such knowledge and thus cannot differentiate between distant single-span and multiple-span ring failures.

**Figure 11.27—Topology change sequences**

11.6 State machines

11.6.1 State machine functions

A station's topology and protection state machines manage the transmission/reception of topology related frames, the reception of administrative requests, and the processing associated with these functions, as illustrated in Figure 11.28.

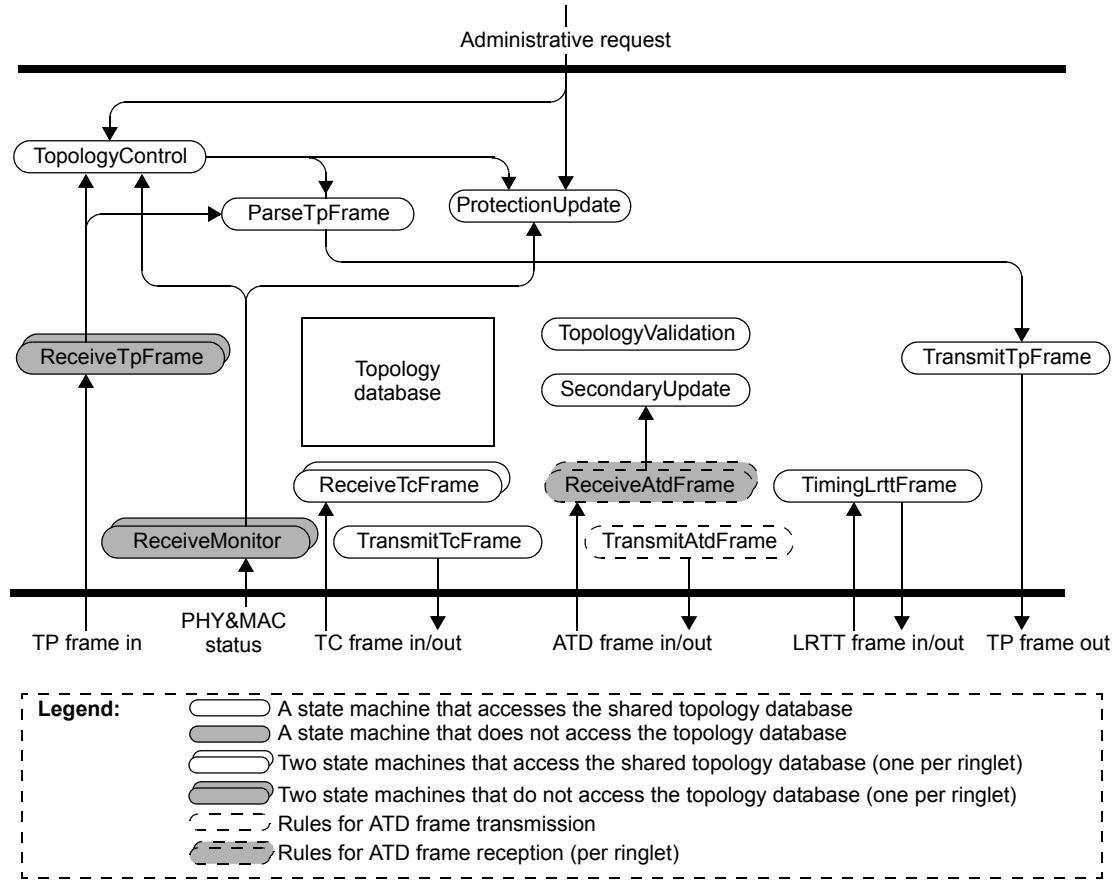


Figure 11.28—Topology and protection relationships

11.6.1.1 ParseTpFrame: The ParseTpFrame state machine (see 11.6.4) processes TP frames to update the topology database.

11.6.1.2 ProtectionUpdate: The ProtectionUpdate state machine (see 11.6.5) is called by TopologyControl and executes on a specified span. ProtectionUpdate uses information contained in the topology database to determine its span's protection state and edge status, before updating the topology database.

11.6.1.3 ReceiveAtdFrame: The ReceiveAtdFrame rules (see 11.6.12) describe the actions taken to receive and process ATD frames.

11.6.1.4 ReceiveMonitor: The pair of ReceiveMonitor state machines (see 11.6.2) monitor span status. A ReceiveMonitor state machine runs on each span of a station.

11.6.1.5 ReceiveTcFrame: The ReceiveTcFrame state machine (see 11.6.10) receives TC frames. A ReceiveTcFrame state machine runs on each span of a station.

11.6.1.6 ReceiveTpFrame: The ReceiveTpFrame state machines (see 11.6.8) receive and preprocess TP frames before passing them to the TopologyControl state machine. A ReceiveTpFrame state machine runs on each span of a station.

11.6.1.7 SecondaryUpdate: The SecondaryUpdate state machine (see 11.6.13) updates secondary MAC addresses, based on received ATT entries in ATD frames, or administrative request directives.

11.6.1.8 TimingLrttFrame: The TimingLrttFrame state machine (see 11.6.14) generates LRTT request frames and processes the returned response frames, for the purpose of calibrating the ring's LRTT.

11.6.1.9 TopologyControl: The TopologyControl state machine (see 11.6.3) processes administrative configuration requests and invokes the ParseTpFrame and ProtectionUpdate state machines.

11.6.1.10 TopologyValidation: The TopologyValidation state machine (see 11.6.6) determines when the topology is stable and valid.

11.6.1.11 TransmitAtdFrame: The TransmitAtdFrame rules (see 11.6.11) describe the actions taken to transmit ATD frames.

11.6.1.12 TransmitTcFrame: The TransmitTcFrame state machine (see 11.6.9) transmits TC frames.

11.6.1.13 TransmitTpFrame: The TransmitTpFrame state machine (see 11.6.4) transmits TP frames.

11.6.2 ReceiveMonitor state machine

The ReceiveMonitor state machine is responsible for monitoring the span status. An SF is set based on a PHY-sensed link failure, a loss of keepalives, or miscabling. An SD is set based on a PHY-sensed link degradation condition.

SCFFs are used as keepalives. The transmission of RPR keepalives should occur only if normal MAC operation is possible.

A configurable *holdoffTimeout* can suppress spurious responses to expected span status glitches, by extending the time between detection and reporting of a physical SF condition. The glitches could be due to protection switching of RPR traffic by underlying SONET infrastructure.

11.6.2.1 ReceiveMonitor state machine definitions

DEGRADE_FAIL
 SIGNAL_DEGRADE
 SIGNAL_FAIL
 See 11.2.8.

11.6.2.2 ReceiveMonitor state machine variables

currentTime
 See 11.2.8.
holdoffStartTime[ri]
 See 11.2.6.
holdoffTimeout[ri]
 The time at which the holdoff was started.
keepaliveTime[ri]
 See 11.2.3.

keepaliveTimeout[ri]

If no keepalive is received within this time, a signal fail condition is assumed. (The minimum 2-millisecond value corresponds to a loss of at least four keepalives. The extent of the configurable *keepaliveTimeout* range facilitates the delay of layer-2 failure detection, so that layer-1 equipment redundancy can take effect before protection is initiated.)

Range: [2 milliseconds, 200 milliseconds]

Resolution: 1 millisecond

Default: 3 milliseconds

LINK_STATUS[ri]

See 11.2.8.

miscablingDefect[ri]

See 11.2.9.

ri

See 11.2.3.

spanOperStatus[ri]

See 11.2.3.

11.6.2.3 ReceiveMonitor state tables

The ReceiveMonitor state machine generates the *spanOperStatus[ri]* indication (for the ProtectionUpdate state machine) based on receive-link status, as specified in Table 11.12. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.12—ReceiveMonitor state table

Current state		Row	Next state	
state	condition		action	state
START	LINK_STATUS[ri] != DEGRADE_FAIL && LINK_STATUS[ri] != SIGNAL_FAIL && LINK_STATUS[ri] != SIGNAL_DEGRADED	1	holdoffStartTime[ri]=currentTime;	FINAL
	—	2	—	
FINAL	currentTime - keepaliveTime[ri] >= keepaliveTimeout[ri]	3	spanOperStatus[ri] = SF;	START
	miscablingDefect[ri]	4	—	
	currentTime - holdoffStartTime[ri] >= holdoffTimeout[ri] && LINK_STATUS[ri] == DEGRADE_FAIL	5	—	
	currentTime - holdoffStartTime[ri] >= holdoffTimeout[ri] && LINK_STATUS[ri] == SIGNAL_FAIL	6	—	
	currentTime - holdoffStartTime[ri] >= holdoffTimeout[ri] && LINK_STATUS[ri] == SIGNAL_DEGRADE	7	spanOperStatus[ri] = SD;	
	—	8	spanOperStatus[ri] = IDLE;	

Row 11.12-1: In the absence of signal-degrade and signal-fail conditions reported by the PHY, the holdoff timer is continually cleared, to ensure the timer is properly initialized at the onset of an SD or SF condition.

Row 11.12-2: Otherwise, the holdoff timer is not cleared.

Row 11.12-3: If the keepalive timer has expired, an SF condition is asserted.

NOTE—The *holdoffTimeout* does not affect the assertion of SF due to loss of keepalives, because *keepaliveTimeout* refers to MAC-layer errors, whereas *holdoffTimeout* applies to PHY-layer errors. Both can be set independently.

Row 11.12-4: If a miscabling condition exists, an SF condition is asserted. Neither the *holdoffTimeout* nor the *keepAliveTimeout* delays the assertion of a miscabling error.

Row 11.12-5: When a combined signal-degrade/signal-fail condition persists for *holdoffTimeout*, an SF condition is reported.

Row 11.12-6: When a signal-fail condition persists for *holdoffTimeout*, an SF condition is reported.

Row 11.12-7: When a signal-degrade condition persists for *holdoffTimeout*, an SD condition is reported.

Row 11.12-8: In the absence of signal-degrade and signal-fail conditions, the span returns to IDLE.

NOTE—Receipt of a single keepalive clears the SF condition due to loss of keepalives. Receipt of a single TP frame on the correct ringlet clears the SF condition due to miscabling. Wait to restore is applied uniformly to all SF conditions.

11.6.3 TopologyControl state machine

11.6.3.1 TopologyControl state machine definitions

JUMBO

See 11.2.2.

JUMBO_MAX

NULL

See 11.2.8.

Q_RX_TP_PARSE

See 11.2.2.

REGULAR

See 11.2.2.

REGULAR_MAX

See 11.2.8.

11.6.3.2 TopologyControl state machine variables

adminReqJumboConfig

An administrative request value, serviced and cleared by the MAC, with the following values:

JUMBO—Indicates a station that prefers to support jumbo frames.

REGULAR—Indicates a station that does not support jumbo frames.

NULL—No jumbo administrative request is available.

adminReqProtectConfig

An administrative request value, serviced and cleared by the MAC, with the following values:

STEERING—Indicates a station that is configured to use steering protection.

WRAPPING—Indicates a station that is configured to use wrapping protection.

NULL—No administrative protection configuration request is available.

adminReqProtection[ri]

See 11.2.3.

currentTime

See 11.2.8.

frame

See 11.2.3.

myTopoInfo

See 11.2.5.

oldStatus[ri]

An old operational status value that facilitates the detection of operational status changes.

protectChanged

See 11.2.3.

protMisconfigDefect

See 11.2.9.

*revertive**ri*

See 11.2.3.

ringInfo

See 11.2.4.

*spanOperStatus[ri]**topoChanged**wtr[ri]*

See 11.2.3.

11.6.3.3 TopologyControl state machine routines

The routines below are used in the TopologyControl state machine.

EntryInQueue(queue)

See 11.2.8.

MismatchedProtection()

See 11.2.7.

Other(ri)

See 11.2.8.

ParseTpFrame()

A routine whose functionality is defined by the ParseTpFrame state machine (see 11.6.4).

ProtectionUpdate()

A routine whose functionality is defined by the ProtectionUpdate state machine (see 11.6.5).

RegularStationExists()

Determines if there is a regular (non-jumbo preferred) station on the ring, as specified by Equation (11.7).

TRUE—There is a regular (non-jumbo preferred) station on the ring.

FALSE—(Otherwise.)

```
Boolean
RegularStationExists()
{
    int ringlet, hops;

    if (myTopoInfo.jumboPrefer == REGULAR)
        return(TRUE);
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        if (myTopoInfo.spanEdge[ringlet])
            continue;
        for (hops = 1; hops <= MAX_STATIONS; hops += 1)
        {
            if (!topoEntry[ringlet][hops].valid ||
                topoEntry[ringlet][hops].macAddress == myTopoInfo.macAddress)
                break;
            if (topoEntry[ringlet][hops].jumboPrefer == REGULAR)
                return(TRUE);
        }
    }
    return(FALSE);
}
```

(11.7)

11.6.3.4 TopologyControl state table

The TopologyControl state machine updates the topology database based on receipt of TP frames from the ring. This state machine is also responsible for calling the ProtectionUpdate state machine, upon changes in local protection status or local administrative requests, as specified in Table 11.13. It also performs ring-level jumbo preference and protection configuration checking.

All stations within a ring are expected to be configured to use the same protection mechanism. Each station indicates its protection configuration in TP frames based on the *myTopoInfo.protConfig* setting. The indication in the TP frame is via the *wc* field (see 11.3.1.4.1). In the event that any station detects any other station that has a protection configuration that does not match its own, a defect is triggered. Each station uses the protection mechanism set by the local *protConfig* setting.

NOTE—The implication of having a ring composed partially of steering configured stations and partially of wrapping configured stations is that an edge span adjacent to one or more steering stations does not result in a wrap at those stations. As a result, traffic sourced at wrapping stations that is intended to traverse the edge span to reach its destination(s) is dropped at this span rather than being wrapped. This scenario results in traffic sourced at wrapping stations not being protected.

A jumbo frame preferred station defaults to a non-jumbo frame preferred behavior, if any non-jumbo frame station is reported through the *jp* (jumbo frame preferred) field in the TP frame (see 11.3.1.4.2).

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.13—TopologyControl state table

Current state		Row	Next state	
state	condition		action	state
START	adminReqJumboConfig != NULL	1	myTopoInfo.jumboPrefer = adminReqJumboConfig; adminReqJumboConfig = NULL;	JUMBO
	adminReqProtectConfig != NULL	2	myTopoInfo.protConfig = adminReqProtectConfig; adminReqProtectConfig = NULL;	PROTC
	EntryInQueue(Q_RX_TP_PARSE)	3	ParseTpFrame(); ri = 0;	MORE
	—	4	ri = 0;	LOOP
MORE	protectChanged == TRUE	5	protectChanged = FALSE;	EXEC
	topoChanged == TRUE	6	—	
	—	7	—	JUMBO

Table 11.13—TopologyControl state table (continued)

Current state		Row	Next state	
state	condition		action	state
LOOP	ri >= 2	8	—	START
	spanOperStatus[ri] != oldStatus[ri]	9	oldStatus[ri] = spanOperStatus[ri];	EXEC
	adminReqProtection[ri] != NULL	10	—	
	revertive && wtr[ri].enabled && myTopoInfo.spanProtState[ri]==WTR && currentTime-wtr[ri].time >= wtr[ri].timeout	11	—	
	—	12	ri += 1;	LOOP
EXEC	—	13	ProtectionUpdate(); ri = Other(ri); ProtectionUpdate();	JUMBO
JUMBO	RegularStationExists()	14	ringInfo.jumboType = REGULAR; ringInfo.mtuSize = REGULAR_MAX;	PROTC
	—	15	ringInfo.jumboType = JUMBO; ringInfo.mtuSize = JUMBO_MAX;	
PROTC	—	16	protMisconfigDefect = MismatchedProtection();	START

Row 11.13-1: The jumbo setting for the station is updated; changes (if any) are noted.

Row 11.13-2: The protection setting for the station is updated; changes (if any) are noted.

Row 11.13-3: Parse any TP frames that have been queued.

Row 11.13-4: Otherwise, check for protection condition changes.

Row 11.13-5: A change in another station's protection state invokes a local protection-state evaluation.

Row 11.13-6: A change in topology invokes a local protection-state evaluation. The relevant changes in topology are addition/removal of edges or of stations, including passthrough events.

Row 11.13-7: Otherwise, the local protection-state remains unchanged. Proceed to jumbo preference and protection configuration checking.

Row 11.13-8: Looping stops after each of the two ringlets have been checked.

Row 11.13-9: A span operational status change invokes a protection-state evaluation.

Row 11.13-10: A protection administrative request invokes a protection-state evaluation.

Row 11.13-11: A WTR completion invokes a protection-state evaluation.

Row 11.13-12: Check for applicable changes on both ringlets.

Row 11.13-13: The ProtectionUpdate state machine is executed on both spans.

Row 11.13-14: If a regular station is present on the ring, the ring jumbo preference type is set to non-jumbo. (This setting is done upon detection of the station, not after topology validation has completed.)

Row 11.13-15: If all stations are jumbo-preferred, then the ring jumbo preference type is set to jumbo.

Row 11.13-16: Update the protection misconfiguration defect.

11.6.4 ParseTpFrame state machine

The ParseTpFrame state machine is responsible for updating the topology database based on receipt of TP frames from the ring.

11.6.4.1 ParseTpFrame state machine definitions

MAX_STATIONS
 Q_RX_TP_PARSE
 SF
 STEERING
 See 11.2.2.

11.6.4.2 ParseTpFrame state machine variables

containmentActive
 See 11.2.3.
frame
hops
 See 11.2.3.
myTopoInfo
 See 11.2.5.
new
 A changed copy of the revised *topoEntry* values.
newNeighbor[rid]
 See 11.2.3.
old
 An unchanged copy of the revised *topoEntry* values.
protectChanged
rid
 See 11.2.3.
ringInfo
 See 11.2.4.
topoChanged
 See 11.2.3.
topoEntry[rid][hops]
 See 11.2.6.
transmitTpFrame
wtr[ri]
 See 11.2.3.

11.6.4.3 ParseTpFrame state machine routines

The routines below are used in the ParseTpFrame state machine.

ClearAtdInfo(rid, hops)
 See 11.2.7.
ComputeTc()
 See 11.2.7.

CopyTopoInfo(rid, hops)

Copies key information from the topology database, so that critical changes can be readily detected, as specified by Equation (11.8).

```
Entry
CopyTopoInfo(rid, hops)
{
    Entry entry;
    int ringlet;

    entry.valid = topoEntry[rid][hops].valid;
    entry.macAddress = topoEntry[rid][hops].macAddress;
    entry.sequenceNumber = topoEntry[rid][hops].sequenceNumber;
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        entry.spanProtState[ringlet] = topoEntry[rid][hops].spanProtState[ringlet];
        entry.spanEdge[ringlet] = topoEntry[rid][hops].spanEdge[ringlet];
    }
    return(entry);
}
```

(11.8)
Dequeue(queue)

See 11.2.8.

FindIndex(frame)

See 11.2.7.

Other(ri)

See 11.2.8.

UpdateTopoEntry(frame, rid, hops)

Copies information from the received TP frame into the topology database, as specified by Equation (11.9).

```
void
UpdateTopoEntry(frame, rid, hops)
{
    topoEntry[rid][hops].macAddress = frame.sa;
    topoEntry[rid][hops].protConfig = frame.wc;
    topoEntry[rid][hops].jumboPrefer = frame.jp;
    topoEntry[rid][hops].spanProtState[0] = frame.psw;
    topoEntry[rid][hops].spanProtState[1] = frame.pse;
    topoEntry[rid][hops].spanEdge[0] = frame.esw;
    topoEntry[rid][hops].spanEdge[1] = frame.ese;
    topoEntry[rid][hops].sequenceNumber = frame.seqnum;
}
```

(11.9)

11.6.4.4 ParseTpFrame state table

The ParseTpFrame state machine updates the topology database based on receipt of TP frames from the ring, as specified in Table 11.14. This state machine is also responsible for detecting when the ProtectionUpdate state machine should be called. It sets *topoChanged* for the TopologyControl and TopologyValidation state machines, *protectChanged* for the TopologyControl state machine, *newNeighbor[rid]* for the ProtectionUpdate state machine, and *transmitTpFrame* for the TransmitTpFrame state machine.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.14—ParseTpFrame state table

Current state		Row	Next state		
state	condition		action	state	
START	—	1	frame = Dequeue(Q_RX_TP_PARSE); rid = frame.ri; hops = FindIndex(frame); old = CopyTopoInfo(rid, hops); UpdateTopoEntry(frame, rid, hops); new = CopyTopoInfo(rid, hops);	NEXT	
NEXT	myTopoInfo.spanProtState[rid] == SF	2	—	RETURN	
	!old.valid	3	topoChanged = TRUE;	TEST	
	old.macAddress != new.macAddress	4			
	old.spanEdge[0] && !new.spanEdge[0]	5			
	old.spanEdge[1] && !new.spanEdge[1]	6			
	!old.spanEdge[0] && new.spanEdge[0]	7	topoChanged = TRUE;	DIFF	
	!old.spanEdge[1] && new.spanEdge[1]	8			
	old.spanProtState[0]!= new.spanProtState[0]	9	protectChanged = TRUE;		
	old.spanProtState[1]!= new.spanProtState[1]	10			
	old.sequenceNumber != new.sequenceNumber	11	—	EXEC	
	—	12	—	RETURN	
TEST	!myTopoInfo.spanEdge[rid]	13	transmitTpFrame = TRUE;	DIFF	
	—	14	—		
DIFF	hops == 1 && frame.sa != myTopoInfo.lastNeighborMac[rid]	15	newNeighbor[rid] = TRUE; myTopoInfo.lastNeighborMac[rid] = frame.sa;	EXEC	
	—	16	—		
EXEC	!myTopoInfo.spanEdge[rid] && old.valid && old.macAddress == new.macAddress && old.sequenceNumber != new.sequenceNumber	17	myTopoInfo.myChecksum.value = ComputeTc();	CC	
	—	18	—		

Table 11.14—ParseTpFrame state table (continued)

Current state		Row	Next state	
state	condition		action	state
CC	!old.valid	19	—	NEAR
	old.macAddress != new.macAddress	20	containmentActive = TRUE; transmitTpFrame = TRUE; ClearAtdInfo(rid, hops);	
	old.spanEdge[0] && !new.spanEdge[0]	21	containmentActive = TRUE;	
	old.spanEdge[1] && !new.spanEdge[1]	22		
	!old.spanEdge[0] && new.spanEdge[0] && myTopoInfo.protConfig == STEERING	23		
	!old.spanEdge[1] && new.spanEdge[1] && myTopoInfo.protConfig == STEERING	24		
	—	25	—	
NEAR	hops != 1 && new.spanEdge[Other(rid)]	26	—	FINAL
	hops == 1 && new.spanEdge[Other(rid)]	27	topoEntry[rid][hops].valid = TRUE; topoEntry[rid][hops].reachable=FALSE; hops += 1;	
	new.spanEdge[rid]	28	topoEntry[rid][hops].valid = TRUE; hops += 1;	
	!old.valid	29	topoEntry[rid][hops].valid = TRUE; transmitTpFrame = TRUE;	RETURN
	—	30	topoEntry[rid][hops].valid = TRUE;	
FINAL	hops > MAX_STATIONS	31	—	RETURN
	—	32	topoEntry[rid][hops].valid = FALSE; topoEntry[rid][hops].reachable=FALSE; ClearAtdInfo(rid, hops); hops += 1;	FINAL

Row 11.14-1: Save the received TP frame values in topoEntry. Also save a copy of the relevant old values, so that those are available for detection of changes.

Row 11.14-2: The information from across an SF span is not trusted. (It is used to update the topology database for diagnostic purposes, but it is not marked valid. This ensures that stale protection information from the neighbor station is not used to override WTR when the SF clears.)

NOTE—The clearing of validity of a neighbor entry across an SF span is done in the ProtectionUpdate state machine.

Row 11.14-3: If the previous entry was invalid, a major topology change is possible.

Row 11.14-4: If the MAC address changed, a major topology change is possible.

Row 11.14-5: If the ringlet0 edge was removed, a major topology change is possible.

Row 11.14-6: If the ringlet1 edge was removed, a major topology change is possible.
Row 11.14-7: If the ringlet0 edge was formed, a minor topology change is possible.
Row 11.14-8: If the ringlet1 edge was formed, a minor topology change is possible.
Row 11.14-9: If the ringlet0 protection-state changed, a minor topology change is possible.
Row 11.14-10: If the ringlet1 protection-state changed, a minor topology change is possible.
Row 11.14-11: If the sequence number has changed due to a jumbo preference or protection configuration change, continue with the topology database update.
Row 11.14-12: Otherwise, no topology database update is needed.

Row 11.14-13: Trigger TP frame transmission unless the major topology change indication comes across an edge.

Row 11.14-14: Otherwise, continue with the topology database update.

Row 11.14-15: Detection of a valid changed neighbor sets variables for the ProtectionUpdate state machine.

Row 11.14-16: Otherwise, the neighbor-changed condition variables are not set.

Row 11.14-17: If the received TP frame has a new sequence number (its content has changed without a topology change), a new topology checksum is computed.

NOTE—This ensures that context containment can be exited in corner case scenarios. The new checksum need not be computed if the topology has changed, because it will be done later.

Row 11.14-18: Otherwise, a new topology checksum is not computed.

Row 11.14-19: Upon the addition of a new station, containment is not required.

(This avoids containment when a local SF is removed, causing the cross-edge neighbor to become valid.)

Row 11.14-20: Upon the movement of a station, containment is required.

Row 11.14-21: Upon the removal of a ringlet0 edge, containment is required.

Row 11.14-22: Upon the removal of a ringlet1 edge, containment is required.

Row 11.14-23: Upon the addition of a ringlet0 edge, steering stations require containment.

Row 11.14-24: Upon the addition of a ringlet1 edge, steering stations require containment.

Row 11.14-25: Otherwise, containment is not invoked.

Row 11.14-26: A non-neighbor entry just beyond an edge is cleared, along with entries beyond this.

NOTE—This row can only be executed for the initial transmitted TP frame indicating an edge on *Other(rid)*. After the edge state is known, subsequent TP frames are stripped by the neighbor station directly across the edge.

Row 11.14-27: Mark an adjacent across-edge neighbor valid but not reachable. Clear entries beyond this.

Row 11.14-28: An entry just before an edge is marked valid. Clear entries beyond this.

Row 11.14-29: Otherwise, an invalid entry is marked valid. Trigger TP frame transmission.

Row 11.14-30: Otherwise, this entry is marked valid and no distant entries need be cleared.

Row 11.14-31: Invalidations stop after the last entry has been marked invalid.

Row 11.14-32: Otherwise, invalidate more-distant entries.

11.6.5 ProtectionUpdate state machine

11.6.5.1 ProtectionUpdate state machine overview

This subclause specifies the unified state machine for both steering and wrapping protection. This state machine runs on the span (east and west) of a station for which it is called.

These protection conditions affect the RPR topology, as illustrated in Figure 11.29. The left column specifies the $P1$ -span and $P2$ -span protection states that lead to the protected topology figure in the right column. More severe $P2$ locations are not illustrated; these can be derived by interchanging the $P1$ and $P2$ span labels. As shown, a span is an edge if there is a sufficiently severe protection condition on one or both links of the span.

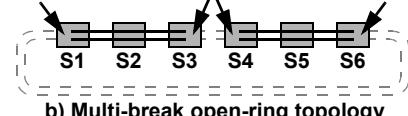
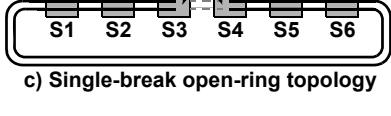
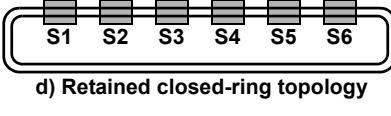
		a) Protection conditions	P2
P1 span state	P2 span state	Resulting protection topology	
FS	FS, SF		b) Multi-break open-ring topology
	SF		
FS	SD, MS, WTR, IDLE		c) Single-break open-ring topology
	SF		
SD	MS, WTR, IDLE		d) Retained closed-ring topology
	MS		
WTR	IDLE		
	WTR		
SD	SD		
	MS		
WTR	WTR		
	IDLE		

Figure 11.29—Protection topologies

An FS directive or an SF condition forces an edge, even though the ring can be severed elsewhere (see Figure 11.29-b).

The protection hierarchy determines which of non-IDLE span states has precedence (see Figure 11.29-c). An FS or SF overrides SD, MS, and WTR; other MS directives are rejected, and WTR conditions are cleared. An SD overrides MS and WTR; other MS directives are rejected, and a WTR condition is cleared. An MS directive overrides a WTR condition; other WTR conditions are cleared in the presence of an MS condition. A WTR condition has precedence over an IDLE condition.

A tie in the protection hierarchy causes duplicate non-fatal span states to be ignored (see Figure 11.29-d). An SD on two distinct spans causes both to be ignored, because the precedence is ambiguous, but an SD on both links of the same span still results in an edge. Concurrent MS protection states on two distinct spans causes both to be cleared, because the precedence is ambiguous. If one MS is known to be active on one span, an MS request for another span is rejected. Two MS requests on two links of the same span still results in an edge. A WTR on two distinct spans causes both to be cleared, because the precedence is ambiguous, but WTR on two links of a span still results in an edge. Lastly, the default IDLE directive with no degradation conditions allows the ring to operate in a closed-ring topology.

NOTE—FS directives or SF conditions simultaneously in effect on both $P1$ and $P2$ are visible to all stations other than S2, as TP frames from S4 across $P1$ are stripped by S3, and TP frames from S6 across $P2$ are stripped by S1. S2 sees idle edges reported by S1 and S3. (S5 does not see idle edges. Rather, it sees FS or SF reported by S4 and S6.) Any station that sees idle edges on both sides of an open ring considers the protection state associated with those edges to be FS, to ensure that non-fatal conditions elsewhere on the open ring are overridden. Any station that sees an idle edge on one side of an open ring and an inconsistent condition on the other side [see Equation (11.10)] also considers the associated protection state to be FS. Any station that sees an idle edge on one side of an open ring and an edge on the other side behaves according to the protection hierarchy. A ring with a single edge also appears to have an idle edge on one side and an edge on the other side; in this case, false overrides of this edge due to itself are prevented.

SF and SD (signal fail or signal degrade) are reported throughout the ring, even when a higher priority request or protection status is present on other spans. This span status knowledge is intended to be beneficial from a diagnostic perspective. All TP frames (whether sourced by neighbor stations, non-neighbor stations, edge-span stations, or non-edge-span stations) are used to determine if protection conditions such as MS or WTR are preempted by protection conditions reported elsewhere on the ring.

11.6.5.2 Station connectivity failures

A station can lose the internal connections between east-side and west-side components, as illustrated in Figure 11.30. An example is a center-wrap station with separate hardware modules (east and west) that are interconnected and the interconnecting cable or circuitry fails.

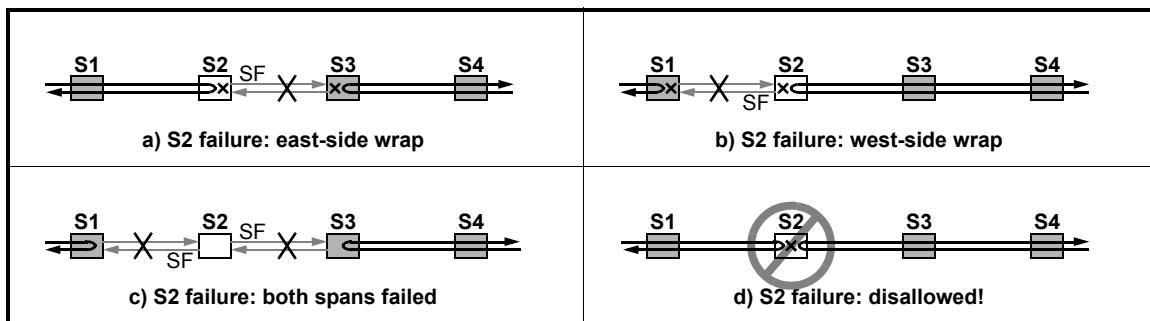


Figure 11.30—Edge condition updates

To accommodate such failures, a station can wrap either its west span (see Figure 11.30-a) or its east span (see Figure 11.30-b), and assert SF on its east span or its west span, respectively. Alternatively, a station can report a failure on both spans (see Figure 11.30-c), which results in the station disappearing from the ring. The SF can be communicated by removing physical link communications, suspending transmission of keepalives, or reporting SF via TP frames (in the case of a single span SF).

A station shall never wrap in both directions (see Figure 11.30-d), to avoid topology confusion (e.g., seeing a wrapped station as two stations with duplicate MAC addresses).

11.6.5.3 ProtectionUpdate state machine definitions

FS

IDLE

MAX_STATIONS

MS

See 11.2.2.

NULL

See 11.2.8.

SD

SF
STEERING
WRAPPING
WTR
See 11.2.2.

11.6.5.4 ProtectionUpdate state machine variables

adminReqProtection
containmentActive
See 11.2.3.
currentTime
See 11.2.9.
distantState
A saved result from an initial call to *DistantStateMax(ri)*.
hops
See 11.2.3.
myTopoInfo
See 11.2.5.
neighborState
A saved result from an initial call to *NeighborState(ri)*.
newNeighbor[ri]
See 11.2.3.
oldEdge
An old edge-state value that facilitates the detection of edge-state changes.
oldState
An old span-protection state value that facilitates the detection of span-protection-state changes.
revertive
ri
See 11.2.3.
spanOperStatus[ri]
spanProtAdmin[ri]
See 11.2.3.
topoChanged
See 11.2.3.
topoEntry
See 11.2.6.
transmitTpFrame
wtr
See 11.2.3.

11.6.5.5 ProtectionUpdate state machine routines

The routines below are used in the protection state machine or are called within routines in this subclause.

ClearAtdInfo(ri, hops)
See 11.2.7.
DistantStateMax(ri)
Reports the largest protection-state associated with any span, other than the span connecting the station to its neighbor, as specified by Equation (11.10).

```
(11.10)
```

```

int
DistantStateMax(ri)
{
    TopoEntry tp;
    int ringlet, hops, idleEdge[2] = {FALSE, FALSE}, awayState, check;
    int topoEnd[2] = {FALSE, FALSE}, idleEdgeNotFromNbr[2] = {FALSE, FALSE};

    awayState = myTopoInfo.spanProtState[Other(ri)];
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        // Protection state beyond edge not taken into account
        if (!myTopoInfo.spanEdge[ringlet])
        {
            for (hops = 1; hops <= MAX_STATIONS; hops += 1)
            {
                tp = topoEntry[ringlet][hops];

                // End of topology reached for non-edge
                if (!tp.valid || tp.macAddress == myTopoInfo.macAddress)
                {
                    topoEnd[ringlet] = TRUE;
                    break;
                }
                // Account for protection state of on far side of neighbor node
                if (ringlet != ri &&
                    tp.macAddress == topoEntry[Other(ringlet)][1].macAddress)
                {
                    awayState = Max(awayState, tp.spanProtState[ri]);
                    break;
                }

                // Idle edge reached. Determine if reported by neighbor
                if (tp.spanEdge[ringlet] && tp.spanProtState[ringlet] == IDLE)
                {
                    idleEdge[ringlet] = TRUE;
                    if (topoEntry[Other(ringlet)][1].valid &&
                        tp.macAddress != topoEntry[Other(ringlet)][1].macAddress)
                        idleEdgeNotFromNbr[ringlet] = TRUE;
                    break;
                }
                // End of topology reached due to MAX_STATIONS
                if (hops == MAX_STATIONS && tp.valid)
                {
                    topoEnd[ringlet] = TRUE;
                    break;
                }

                if ((hops == 1) && (ringlet == ri))
                    awayState = Max(awayState, tp.spanProtState[ri]);
                else
                    awayState = Max(awayState,
                        Max(tp.spanProtState[ri], tp.spanProtState[Other(ri)]));
                // Edge reached
                if (tp.spanEdge[ringlet])
                    break;
            }
        }
        else
    }
    // In 2-node ring with immediately encountered idle edge,
    // account for protection state of on other link of the idle edge span
    if (ringlet != ri && myTopoInfo.spanProtState[ringlet] == IDLE &&
        topoEntry[ringlet][1].valid && topoEntry[Other(ringlet)][1].valid)
        awayState = Max(awayState, topoEntry[ringlet][1].spanProtState[ri];
}

```

```

        }

    }

    check = (idleEdge[0] && idleEdge[1]) ||
        (idleEdge[0] && topoEnd[1]) || (topoEnd[0] && idleEdge[1]);
    awayState = check ? FS : awayState;

    // Return FS unless the highest protection state on the ring is WTR
    // This prevents an MS command from being overridden by WTR
    if (myTopoInfo.spanProtState[ri] < SF && idleEdge[Other(ri)] &&
        idleEdgeNotFromNbr[Other(ri)] && awayState != WTR)
        awayState = FS;

    return(awayState);
}

```

Max(valueA, valueB)

See 11.2.8.

NeighborState(ri)

Reports the protection state of the neighbor, as specified by Equation (11.11).

(topoEntry[ri][1].valid ? topoEntry[ri][1].spanProtState[Other(ri)] : IDLE) (11.11)

IndicateEdgeState()

Communicates the edge state to LME when the edge state changes. The LME can use this information to compute protection duration, last activation time of protection, and number of times protection has been initiated on that span.

Other(ri)

See 11.2.8.

11.6.5.6 ProtectionUpdate state table

The ProtectionUpdate state table is specified in Table 11.15. In addition to determination of the edge state of the directly connected span, it updates the topology database based on this. It sets *topoChanged* for TopologyValidation and *transmitTpFrame* for TransmitTpFrame.

Operational status (*spanOperStatus[ri]*) and administrative state (*spanProtAdmin[ri]*) are separate inputs so that upon clearing of FS, a pending SF or SD takes effect immediately.

Both the protection state of the neighbor (*neighborState*) and the highest protection state on the rest of the ring (*distantState*) can override/preempt a non-fatal local protection state. If the rest of the ring is IDLE, both the local protection state and the protection state of the neighbor must become IDLE to clear an edge.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.15—ProtectionUpdate state table

Current state		Row	Next state	
state	condition		action	state
START	—	1	oldEdge = myTopoInfo.spanEdge[ri]; oldState= myTopoInfo.spanProtState[ri]; neighborState = NeighborState(ri); distantState = DistantStateMax(ri);	ADMIN

Table 11.15—ProtectionUpdate state table (continued)

Current state		Row	Next state	
state	condition		action	state
ADMIN	adminReqProtection[ri] == FS	2	spanProtAdmin[ri] = FS; adminReqProtection[ri] = NULL;	MAIN
	adminReqProtection[ri] == MS && distantState >= MS	3	spanProtAdmin[ri] = IDLE; adminReqProtection[ri] = NULL;	
	adminReqProtection[ri] == MS && neighborState > MS	4		
	adminReqProtection[ri] == MS	5	spanProtAdmin[ri] = MS; adminReqProtection[ri] = NULL;	
	adminReqProtection[ri] == IDLE	6	spanProtAdmin[ri] = IDLE; adminReqProtection[ri] = NULL;	
	—	7	—	
MAIN	spanProtAdmin[ri] == FS	8	myTopoInfo.spanProtState[ri] = FS; myTopoInfo.spanEdge[ri] = TRUE; wtr[ri].enabled = FALSE;	MARK
	spanOperStatus[ri] == SF	9	myTopoInfo.spanProtState[ri] = SF; myTopoInfo.spanEdge[ri] = TRUE; spanProtAdmin[ri] = IDLE; wtr[ri].enabled = TRUE;	
	spanOperStatus[ri] == SD && neighborState > SD	10	myTopoInfo.spanProtState[ri] = SD; myTopoInfo.spanEdge[ri] = TRUE; spanProtAdmin[ri] = IDLE; wtr[ri].enabled = FALSE;	
	spanOperStatus[ri] == SD && distantState >= SD	11	myTopoInfo.spanProtState[ri] = SD; myTopoInfo.spanEdge[ri] = FALSE; spanProtAdmin[ri] = IDLE; wtr[ri].enabled = FALSE;	
	spanOperStatus[ri] == SD	12	myTopoInfo.spanProtState[ri] = SD; myTopoInfo.spanEdge[ri] = TRUE; spanProtAdmin[ri] = IDLE; wtr[ri].enabled = TRUE;	
	spanProtAdmin[ri] == MS && distantState >= MS	13	myTopoInfo.spanProtState[ri] = IDLE; spanProtAdmin[ri] = IDLE;	IDLE
	spanProtAdmin[ri] == MS && neighborState > MS	14		
	spanProtAdmin[ri] == MS	15	myTopoInfo.spanProtState[ri] = MS; myTopoInfo.spanEdge[ri] = TRUE; wtr[ri].enabled = FALSE;	MARK
	myTopoInfo.spanProtState[ri] == IDLE	16	—	IDLE
	myTopoInfo.spanProtState[ri] == WTR	17	—	WTRC
	—	18	wtr[ri].time = currentTime;	

Table 11.15—ProtectionUpdate state table (continued)

Current state		Row	Next state	
state	condition		action	state
WTRC	distantState >= WTR	19	myTopoInfo.spanProtState[ri] = IDLE;	IDLE
	neighborState > WTR	20		
	newNeighbor[ri]	21		
	!wtr[ri].enabled	22		
	revertive && currentTime - wtr[ri].time >= wtr[ri].timeout	23		
	—	24	myTopoInfo.spanProtState[ri] = WTR; myTopoInfo.spanEdge[ri] = TRUE;	MARK
IDLE	neighborState > distantState	25	myTopoInfo.spanEdge[ri] = TRUE; newNeighbor[ri] = FALSE;	MARK
	neighborState > SD	26		
	—	27	myTopoInfo.spanEdge[ri] = FALSE; newNeighbor[ri] = FALSE;	
MARK	myTopoInfo.spanProtState[ri] == SF	28	topoEntry[ri][1].valid = FALSE; topoEntry[ri][1].reachable = FALSE; ClearAtdInfo(ri, 1); hops = 2;	CLEAR
	myTopoInfo.spanEdge[ri]	29	topoEntry[ri][1].valid = TRUE; topoEntry[ri][1].reachable = FALSE; hops = 2;	
	—	30	—	CHECK
CLEAR	hops > MAX_STATIONS	31	—	CHECK
	—	32	topoEntry[ri][hops].valid = FALSE; topoEntry[ri][hops].reachable = FALSE; ClearAtdInfo(ri, hops); hops += 1;	CLEAR
CHECK	myTopoInfo.spanEdge[ri] != oldEdge && oldEdge	33	containmentActive = TRUE;	FINAL
	myTopoInfo.spanEdge[ri] != oldEdge && myTopoInfo.protConfig == STEERING	34		
	—	35	—	
FINAL	oldEdge != myTopoInfo.spanEdge[ri]	36	topoChanged = TRUE; transmitTpFrame = TRUE; IndicateEdgeState();	RETURN
	oldState != myTopoInfo.spanProtState[ri]	37	transmitTpFrame = TRUE;	
	—	38	—	

Row 11.15-1: Save state (to facilitate change detections). Determine protection state of the neighbor on this span, and the largest protection state associated with any other span.

Row 11.15-2: A FS administrative request is retained.

Row 11.15-3: An MS administrative request is preempted by *distantState*.

(The rejected MS has the behavior of an IDLE, in that a pending FS (if any) is canceled.)

Row 11.15-4: An MS administrative request is overridden by *neighborState*.

(The rejected MS has the behavior of an IDLE, in that a pending FS (if any) is canceled.)

Row 11.15-5: Otherwise, the MS administrative request takes effect.

Row 11.15-6: An IDLE administrative request is always accepted.

Row 11.15-7: No administrative request is available.

Row 11.15-8: An FS administrative request forces an edge and disables the WTR.

Row 11.15-9: An SF operational state forces an edge and enables WTR.

Row 11.15-10: An SD operational state overridden by *neighborState* forces an edge and disables WTR.

Row 11.15-11: An SD operational state preempted by *distantState* removes an edge and disables WTR.

Row 11.15-12: Otherwise, a SD operational state forces an edge and enables WTR.

Row 11.15-13: An MS administrative request is preempted by *distantState*. (The edge is removed in the IDLE state.)

Row 11.15-14: An MS administrative request is overridden by *neighborState*. (The edge is forced in the IDLE state.)

Row 11.15-15: An MS administrative request forces an edge and disables the WTR.

Row 11.15-16: The IDLE protection state is processed.

Row 11.15-17: The WTR protection state is processed.

Row 11.15-18: Otherwise, proceed to the WTRC state to check if WTR is entered.

Row 11.15-19: Another's preemptive protection terminates the WTR timer, changing to IDLE.

Row 11.15-20: A neighbor's override terminates the WTR timer, changing to IDLE.

Row 11.15-21: A new neighbor terminates the WTR timer, changing to IDLE.

Row 11.15-22: If the WTR was disabled, the IDLE-state transition is immediate (such as upon clearing of FS or MS).

Row 11.15-23: If revertive, transition to the IDLE protection state after the WTR timer expires.

Row 11.15-24: Otherwise, the WTR condition remains (or is entered) and an edge state is forced.

Row 11.15-25: A non-preempted overriding neighbor forces an IDLE-state edge. This ensures that edges are created as required in passthrough cases.

Row 11.15-26: A neighbor's SF or FS always forces an IDLE-state edge.

Row 11.15-27: Otherwise, any existing edge is cleared.

Row 11.15-28: A neighbor across an SF edge is marked invalid; distant entries are invalidated.

Row 11.15-29: A neighbor across a non-SF edge is marked valid; distant entries are invalidated.

Row 11.15-30: Otherwise, the neighbor state remains unchanged.

Row 11.15-31: Invalidations stop after the last entry has been marked invalid.

Row 11.15-32: Otherwise, invalidate more-distant entries.

Row 11.15-33: If a station's edge state has disappeared, context containment is activated.

Row 11.15-34: If a steering-protected station's edge state appears, context containment is activated.

Row 11.15-35: No action is performed if the edge has not changed.

Row 11.15-36: Set *topoChanged* and *transmitTpFrame* when the edge condition changes.

An edge change and value are also indicated to the LME, to facilitate protection-related statistics updates.

Row 11.15-37: Set *transmitTpFrame* when the protection-state changes.

Row 11.15-38: Otherwise, neither *topoChanged* nor *transmitTpFrame* is changed.

11.6.6 TopologyValidation state machine

The TopologyValidation state machine validates the topology database, sets *topologyStable* and *topologyValid* accordingly, and reports detected defects. The *topologyStable* value is set when the topology (station and edge positions) has not changed for *stabilityTimeout*, and when all stations in the topology are valid. A *topoInstabilityDefect* is reported if the topology does not stabilize within *instabilityTimeout*.

When topology is stable, *topologyValid* is set if topology consistency and station-count checks pass. If an invalid entry is found within the scope of the topology, a *topoEntryInvalidDefect* is reported. If the number of stations on the ring exceeds MAX_STATIONS, a *maxStationsDefect* is reported. If the topology is determined to be inconsistent, a *topoInconsistencyDefect* is reported. If the protection configuration of the stations on the ring is nonuniform, a *protMisconfigDefect* is reported.

Once set, these defects (see 11.2.9) can be cleared after a topology change, followed by restabilization.

11.6.6.1 Topology consistency

Topology consistency checking is done by checking valid station ordering on ringlet0 against the mirror-image ordering on ringlet1. This type of check can often identify duplicate MAC address errors, as illustrated in Figure 11.31.

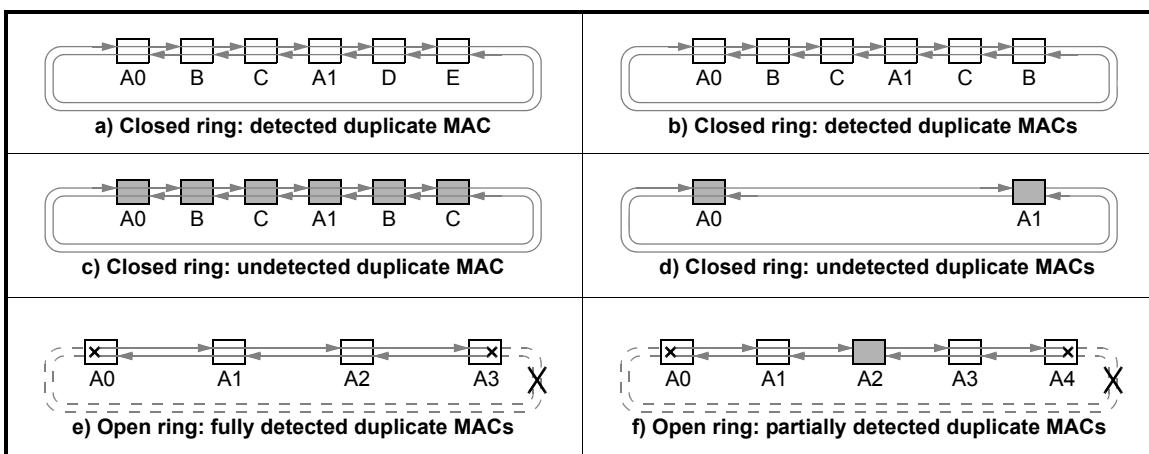


Figure 11.31—Duplicate MAC address scenarios

On closed rings, duplicate MACs can often be detected due to the irregular (see Figure 11.31-a) or asymmetric (see Figure 11.31-b) nature of the topology.

Symmetric topologies with duplicate MAC address errors cannot be detected on closed-ring topologies (see Figure 11.31-c). Similarly, a closed-ring topology with all MAC addresses duplicated (see Figure 11.31-d) cannot be distinguished from a station looped back to itself.

In open rings, duplicate MACs can sometimes be detected even when all MAC addresses in the topology are duplicated (see Figure 11.31-e and Figure 11.31-f). Stations that are connected to an edge span or that are a single hop removed from an edge span can identify that there is a duplicate. In Figure 11.31-f, station A2 cannot distinguish the topology from a station looped back to itself.

If an inconsistency is found and persists for 10 seconds, then an implementation may choose to stop adding or dropping all data frames from the ring.

11.6.6.2 TopologyValidation state machine definitions

CLOSED_RING

OPEN_RING

See 11.2.2.

11.6.6.3 TopologyValidation state machine variables

check

A variable that captures the value returned by the *InvalidTopoCheck()* routine.

containmentActive

See 11.2.3.

currentTime

See 11.2.9.

instabilityTime

The time at which the instability defect timer has started.

instabilityTimeout

The time after which the instability defect timer is due to expire.

Value: 10 seconds

lrrtRequest

See 11.2.3.

maxStationsDefect

See 11.2.9.

myTopoInfo

See 11.2.5.

needSecondaryMacValidation

See 11.2.3.

pircTopologyChange

See 11.2.3.

protMisconfigDefect

See 11.2.9.

ringInfo

See 11.2.4.

stabilityTime

The time at which the topology stability timer has started.

stabilityTimeout

The topology stabilization time.

Range: [10 milliseconds, 100 milliseconds]

Default: 40 milliseconds.

tempHopsRx[ringlet]

topoChanged

See 11.2.3.

topoEntry

See 11.2.6.

topoEntryInvalidDefect

topoInconsistencyDefect

topoInstabilityDefect

See 11.2.9.

topologyStable

topoType

topologyValid

transmitTcFrame

See 11.2.3.

11.6.6.4 TopologyValidation state machine routines

ChecksumMatch()

Compares the checksums received from non-edge neighbor(s), as specified by Equation (11.12).

TRUE—There are checksum matches with each of the valid neighbor station(s)

FALSE—(Otherwise.)

```
Boolean
ChecksumMatch()
{
    int ringlet, invalid;

    invalid = !myTopoInfo.myChecksum.valid;
    for (ringlet = 0; ringlet < 2; ringlet += 1)
        if (!myTopoInfo.spanEdge[ringlet] &&
            (!myTopoInfo.neighborCheck[ringlet].valid ||
             myTopoInfo.myChecksum.value != myTopoInfo.neighborCheck[ringlet].value));
            invalid = TRUE;
    return(!invalid);
}
```

(11.12)

ChecksumStep(checksum, macAddress, sequenceNumber)

See 11.2.7.

ComputeTc()

See 11.2.7.

InvalidTopoCheck()

Checks the topology database for inconsistencies, sets the hop count for each ringlet, and differentiate between open-ring and closed-ring topologies, as specified by Equation (11.13).

INVALID—An invalid entry is detected, typically because the topology has not stabilized.

EXCESS—An excessive number of stations are detected.

INCONSISTENT—An unexpected duplicate or asymmetric MAC address is detected.

CONSISTENT—(Otherwise.)

```
int
InvalidTopoCheck()
{
    int ringlet, hops, hops0, hops1, maxHops, topoType = CLOSED_RING;
    Boolean selfImage = FALSE;

    tempHopsRx[0] = 0;
    tempHopsRx[1] = 0;
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        if (myTopoInfo.spanEdge[ringlet])
        {
            topoType = OPEN_RING;
            continue;
        }
        for (hops = 1; hops <= MAX_STATIONS; hops += 1)
        {
            if (!topoEntry[ringlet][hops].valid)
                return(INVALID);
            if (topoEntry[ringlet][hops].spanEdge[ringlet])
            {
                topoType = OPEN_RING;
                tempHopsRx[ringlet] = hops;
                break;
            }
            if (myTopoInfo.macAddress == topoEntry[ringlet][hops].macAddress)
            {
```

(11.13)

```

        selfImage = TRUE;
        tempHopsRx[ringlet] = hops - 1;
        break;
    }
    if (hops == MAX_STATIONS)
        return(EXCESS);
    }
}
if (topoType == OPEN_RING)
    return(selfImage ? INCONSISTENT : CONSISTENT);
if (selfImage == FALSE || tempHopsRx[0] != tempHopsRx[1])
    return(INCONSISTENT);
maxHops = tempHopsRx[0];
for (hops0 = 1, hops1 = maxHops; hops0 <= maxHops; hops0 += 1, hops1 -= 1)
    if (topoEntry[0][hops0].macAddress != topoEntry[1][hops1].macAddress)
        return(INCONSISTENT);
return(CONSISTENT);
}

```

MarkReachableEntries()

Sets the reachability fields in the topology database after the database is determined to be stable and valid, as specified by Equation (11.14). Also clears validity bits for all stations beyond an edge in an open ring, or beyond the scope of the topology in a closed ring.

```

void
MarkReachableEntries() (11.14)
{
    int ringlet, hops, plus;

    ringInfo.totalHopsRx[0] = tempHopsRx[0];
    ringInfo.totalHopsRx[1] = tempHopsRx[1];
    ringInfo.totalHopsTx[0] = tempHopsRx[1];
    ringInfo.totalHopsTx[1] = tempHopsRx[0];
    ringInfo.numStations = 1 +
        topoType == CLOSED_RING ? tempHopsRx[0] : tempHopsRx[0] + tempHopsRx[1];
    plus = (ringInfo.topoType == CLOSED_RING);
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        for (hops = 1; hops <= ringInfo.totalHopsRx[ringlet] + plus; hops += 1)
            topoEntry[ringlet][hops].reachable = TRUE;
        for ( ; hops <= MAX_STATIONS; hops += 1)
            topoEntry[ringlet][hops].reachable = FALSE;
    }
}

```

MismatchedProtection()

See 11.2.7.

Other(ri)

See 11.2.8.

SdbPurgeUnreachable()

See 11.2.7.

SetSourceCheck()

Configures the *sourceCheck* database (see 7.2.2) after the topology is determined to be stable and valid, as specified by Equation (11.15).

```

void SetSourceCheck()
{
    int ringlet, hops, hopsWrap, othRi;

    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        for (hops = 1; hops <= MAX_STATIONS; hops += 1)
        {
            sourceCheck[ringlet][hops] = 0xFFFFFFFFFFFF;
            if (hops <= ringInfo.totalHopsRx[ringlet])
                sourceCheck[ringlet][hops] = topoEntry[ringlet][hops].macAddress;
        }
    }
    if (myTopoInfo.protConfig == WRAPPING && ringInfo.topoType == OPEN_RING)
    {
        for (ringlet = 0; ringlet < 2; ringlet += 1)
        {
            othRi = Other(ringlet);
            for (hops = ringInfo.totalHopsTx[ringlet]; hops >= 1; hops -= 1)
            {
                hopsWrap = ringInfo.numStations - hops;
                sourceCheck[ringlet][hopsWrap] = topoEntry[othRi][hops].macAddress;
            }
        }
    }
}

```

11.6.6.5 TopologyValidation state table

The TopologyValidation state machine validates the topology database, as specified in Table 11.16. The input variable *topoChanged* (from the ParseTpFrame state machine) indicates a stability-impacting source MAC address or edge state has changed. The TopologyValidation state machine determines whether any topology stability-impacting changes have occurred during the period of the stability timer. If validation checks fail, defects are reported. If all validation checks pass, the topology database is updated.

Table 11.16—TopologyValidation state table

Current state		Row	Next state	
state	condition		action	state
START	—	1	stabilityTime = currentTime; instabilityTime = currentTime; topologyValid = FALSE; topologyStable = FALSE; myTopoInfo.myChecksum.valid = FALSE; lrrtRequest = FALSE;	UN STABLE
UN STABLE	!topoChanged && currentTime - stabilityTime >= stabilityTimeout	2	topologyStable = TRUE; lrrtRequest = TRUE; check = InvalidTopoCheck();	STABLE
	topoChanged	3	stabilityTime = currentTime; topoChanged = FALSE;	UN STABLE
	currentTime - instabilityTime >= instabilityTimeout	4	topoInstabilityDefect = TRUE; instabilityTime = currentTime;	
	—	5	—	

Table 11.16—TopologyValidation state table (continued)

Current state		Row	Next state	
state	condition		action	state
STABLE	check == INVALID	6	topoEntryInvalidDefect = (check == INVALID); maxStationsDefect = (check == EXCESS); topoInconsistencyDefect = (check == INCONSISTENT);	INVAL
	check == EXCESS	7		
	check == INCONSISTENT	8		
	—	9	topoInstabilityDefect = FALSE; maxStationsDefect = FALSE; topoInconsistencyDefect = FALSE; ringInfo.topoType = topoType; MarkReachableEntries(); myTopoInfo.myChecksum.value = ComputeTc(); myTopoInfo.myChecksum.valid = TRUE; myTopoInfo.neighborCheck[0].valid = FALSE; myTopoInfo.neighborCheck[1].valid = FALSE; SetSourceCheck(); protMisconfigDefect = MismatchedProtection(); transmitTcFrame = TRUE; topologyValid = TRUE; topoEntryInvalidDefect = FALSE; needSecondaryMacValidation = TRUE;	VALID
	topoChanged	10	—	START
VALID	—	11	—	INVAL
	myTopoInfo.pircGroup-Member && topoChanged	12	pircTopologyChange = TRUE	START
	myTopoInfo.sasUser && topoChanged	13	SdbPurgeUnreachable();	
	topoChanged	14	—	
	ChecksumMatch()	15	containmentActive = FALSE;	VALID
	—	16	—	

The variable *lrittRequest* is set for the TimingLrttFrame state machine. The variable *transmitTcFrame* is set for the TransmitTcFrame state machine. The variable *needSecondaryMacValidation* is set for the SecondaryUpdate state machine. The *sourceCheck* database (see 7.2.2) is populated.

When topology is validated and the topology checksum matches valid neighbor checksums, then context containment is cleared (see 11.1.3).

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 11.16-1: At the start, timeouts are started and information is marked unstable or invalid.

Row 11.16-2: A topology that is unchanged for *stabilityTimeout* is marked stable and LRTTs are enabled.

Row 11.16-3: The *stabilityTimeout* timer is cleared whenever the topology changes.

Row 11.16-4: If the topology remains unstable, the *topoInstabilityDefect* timer is set.

Row 11.16-5: Otherwise, the topology remains unstable.

Row 11.16-6: If an invalid entry is within the scope of the topology, a *topoEntryInvalidDefect* is indicated.

Row 11.16-7: If excess stations are visible, a *maxStationsDefect* is indicated.

Row 11.16-8: If the apparent station locations are inconsistent, a *topoInconsistencyDefect* is indicated.

Row 11.16-9: Otherwise, the topology is marked valid.

Row 11.16-10: An invalid topology is rechecked in the presence of a topology change.

Row 11.16-11: Otherwise, the topology remains invalid.

Row 11.16-12: A topology change forces a rechecking of the topology. If the station is a PIRC protection group member, set an indication for the PircControl state machine.

Row 11.16-13: A topology change forces a rechecking of the topology. If SAS is active, call routine *SdbPurgeUnreachable()* to remove SDB entries where targetAddress is unreachable.

Row 11.16-14: A topology change forces a rechecking of the topology.

Row 11.16-15: The presence of matching checksums releases context containment.

Row 11.16-16: Otherwise, the topology remains stable.

11.6.7 TransmitTpFrame state machine

TP frames are transmitted on two distinct periodic timers, *txFastTimeout* and *txSlowTimeout* (see 11.2.3). A fixed number of frames are sent on the fast timer when *transmitTpFrame* is set to TRUE, before reverting to the *txSlowTimeout* rate.

NOTE—The *transmitTpFrame* variable is set to TRUE on the start of RPR topology discovery, when protection state changes, when a new station is detected at a given position in the ring, or when station positions have changed. It is not set to TRUE upon change of protection configuration or jumbo preference.

11.6.7.1 TransmitTpFrame state machine constants

FAST_TP_COUNT

The number of frames transmitted on the fast TP timer, before using the slow TP timer.

Value—8

MAX_SEQNUM

The maximum value of the 6-bit *seqnum* field.

Value—63

Q_TX_RS

See 11.2.8.

11.6.7.2 TransmitTpFrame state machine variables

currentTime

See 11.2.9.

fastTpCount

The number of TP frames remaining to be transmitted on the fast TP timer.

frame

See 11.2.3.

myTopoInfo

See 11.2.5.

oldPrefsJp

A previously transmitted jumbo preference value that is used to determine when TP frame content has changed.

oldPrefsWc

A previously transmitted protection configuration value that is used to determine when TP frame content has changed.

oldProtState

A previously transmitted protection state value that is used to determine when TP frame content has changed.

*transmitTpFrame**txFastTimeout**txSlowTimeout*

See 11.2.3.

txTpTime

The start of the current transmission-interval timeout period.

11.6.7.3 TransmitTpFrame state machine routines

EnqueueRi(rid, queue, frame)

See 11.2.8.

FormTpFrame()

Forms a copy of the TP frame in preparation for frame transmission.

11.6.7.4 TransmitTpFrame state table

The TransmitTpFrame state machine determines when TP frames are sent, as specified in Table 11.17. A change in the advertised protection state or a set *transmitTpFrame* value triggers fast timer-based TP frame transmission.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.17—TransmitTpFrame state table

Current state		Row	Next state	
state	condition		action	state
START	transmitTpFrame == TRUE && fastTpCount != 0	1	fastTpCount = FAST_TP_COUNT; transmitTpFrame = FALSE;	START
	transmitTpFrame == TRUE && fastTpCount == 0	2	fastTpCount = FAST_TP_COUNT - 1; transmitTpFrame = FALSE;	NEXT
	fastTpCount >= 1 && currentTime - txTpTime >= txFastTimeout	3	fastTpCount -= 1;	
	fastTpCount == 0 && currentTime - txTpTime >= txSlowTimeout	4	—	
	—	5	—	START
NEXT	—	6	frame = FormTpFrame();	NEAR
NEAR	frame.protStatus != oldProtState	7	myTopoInfo.sequenceNumber = (myTopoInfo.sequenceNumber +1) % (MAX_SEQNUM+1); myTopoInfo.myChecksum.value = ComputeTc();	FINAL
	frame.prefs.wc != oldPrefsWc	8	—	
	frame.prefs.jp != oldPrefsJp	9	—	
	—	10	—	

Table 11.17—TransmitTpFrame state table

Current state		Row	Next state	
state	condition		action	state
FINAL	—	11	oldProtState = frame.protStatus; oldPrefsWc = frame.wc; oldPrefsJp = frame.jp; txTpTime = currentTime; EnqueueRi(0, Q_TX_RS, frame); EnqueueRi(1, Q_TX_RS, frame);	START

Row 11.17-1: Always send at a fixed number of *txFastTimeout* interval frames after the most recent trigger.
(Do not trigger immediately, however, if a TP frame was transmitted less than *txFastTimeout* ago.)

Row 11.17-2: A trigger during a *txSlowTimeout* interval restarts *txFastTimeout* transmissions immediately.

Row 11.17-3: When initial fast timers expire, update the counter and timer, and then do fast rate transmission.

Row 11.17-4: After the fast timers complete, update counter and timer, and then do slow rate transmission.

Row 11.17-5: Otherwise, continue checking until a timer expires.

Row 11.17-6: Form the next frame, in preparation for transmission.

Row 11.17-7: Upon a change in *protStatus*, increment the sequence number and compute the checksum.

Row 11.17-8: Upon a change in *prefs.wc*, increment the sequence number and compute the checksum.

Row 11.17-9: Upon a change in *prefs.jp*, increment the sequence number and compute the checksum.

Row 11.17-10: If the field contents have not changed, the sequence number is not incremented.

Row 11.17-11: Copy the frame (so that changes can be detected), clear the transmission-interval timer, and send a TP frame on each ringlet.

11.6.8 ReceiveTpFrame state machines

The receipt of a TP frame containing new information causes the MAC control sublayer to update its topology database. When a TP frame indicates miscabling, the topology database is not updated, but a *miscablingDefect[ri]* variable is set for use in the ReceiveMonitor state machine.

TP frames provide a sequence number to indicate when the TP frame content has changed. This allows redundant frames to be ignored.

Stations shall be capable of completing topology discovery for any stable ring of MAX_STATIONS for a worst-case TP frame retransmission period. (See *txFastTimeout* in 11.6.7.)

NOTE—One of several possible techniques to meet the preceding requirement would be to use a hardware front end that discards frames with sequence numbers matching those in the topology database for that source station.

11.6.8.1 ReceiveTpFrame state machine definitions

MAX_STATIONS
See 11.2.2.

NULL
See 11.2.8.

Q_RX_TP_PARSE
See 11.2.2.

Q_RX_TP
See 11.2.8.

11.6.8.2 ReceiveTpFrame state machine variables

duplicateSecMacAddressDefect

See 11.2.9.

frame

hops

See 11.2.3.

index

Index of topology database entry that is updated by the received TP frame.

miscablingDefect[ri]

See 11.2.9.

ri

See 11.2.3.

topoEntry

See 11.2.6.

11.6.8.3 ReceiveTpFrame state machine routines

The routines below are used in the TP frame receive state machine.

DequeueRi(ri, queue)

See 11.2.8.

Enqueue(queue, frame)

See 11.2.8.

FindIndex(frame)

See 11.2.7.

11.6.8.4 ReceiveTpFrame state table

Upon receipt of a TP frame on a span *ri*, the ReceiveTpFrame state machine checks whether that frame passes the miscabling and sequence number checks, to properly set or clear miscabling defects on the two spans of the station.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.18—ReceiveTpFrame state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = DequeueRi(ri, Q_RX_TP)) != NULL;	1	hops = FindIndex(frame);	FIRST
	—	2	—	START

Table 11.18—ReceiveTpFrame state table

Current state		Row	Next state	
state	condition		action	state
FIRST	frame.ri != ri && frame.ttl != MAX_STATIONS	3	—	START
	frame.ri != ri && frame.ttl == MAX_STATIONS	4	miscablingDefect[ri] = TRUE;	
	frame.ri == ri && frame.ttl == MAX_STATIONS	5	miscablingDefect[ri] = FALSE;	FINAL
	miscablingDefect[ri]	6	—	START
	—	7	—	FINAL
FINAL	topoEntry[ri][hops].valid && topoEntry[ri][hops].macAddress == frame.sa && topoEntry[ri][hops].sequenceNumber == frame.seqnum	8	—	START
	—	9	Enqueue(Q_RX_TP_PARSE, frame);	

Row 11.18-1: Find the hop count for the topology database entry corresponding to the TP frame's source.

Row 11.18-2: Wait until a TP frame is received.

Row 11.18-3: If a non-neighbor is incorrectly cabled, the TP frame is ignored.

Row 11.18-4: If the neighbor is incorrectly cabled, set the miscabling defect.

Row 11.18-5: If the neighbor is correctly cabled, clear the miscabling defect.

Row 11.18-6: If the miscabling defect persists, the TP frame is ignored.

Row 11.18-7: Otherwise, the TP frame is processed.

Row 11.18-8: If the sequence number matches that in the topology database, the topology database entry is valid, and the source MAC matches that in the topology database, the redundant TP frame is discarded.

Row 11.18-9: The prechecked frame is passed to the ParseTpFrame state machine.

11.6.9 TransmitTcFrame state machine

This subclause specifies the state machine for determining when TC frames are sent on the ring from a station and related requirements.

11.6.9.1 TransmitTcFrame state machine constants

FAST_TC_COUNT

The number of frames transmitted on the fast TC timer, before using the slow TC timer.

Value—4

Q_TX_RS

See 11.2.8.

11.6.9.2 TransmitTcFrame state machine variables

currentTime

See 11.2.9.

fastTcCount

The number of TC frames remaining to be transmitted on the fast TC timer.

frame

See 11.2.3.

*txFastTimeout**txSlowTimeout*

See 11.2.3.

transmitTcFrame

See 11.2.3.

txTcTime

The time at which the next TC frame should be sent.

11.6.9.3 TransmitTcFrame state machine routines*EnqueueRi(ringlet, queue, frame)*

See 11.2.8.

11.6.9.4 TransmitTcFrame state table

The TransmitTcFrame state machine determines when TC frames are sent, as specified in Table 11.19. The input variable *transmitTcFrame* triggers fast-timer based TC frame transmissions.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.19—TransmitTcFrame state table

Current state		Row	Next state	
state	condition		action	state
START	transmitTcFrame && fastTcCount != 0	1	transmitTcFrame = FALSE; fastTcCount = FAST_TC_COUNT;	START
	transmitTcFrame && fastTcCount == 0	2	fastTcCount = FAST_TC_COUNT - 1; transmitTcFrame = FALSE;	FINAL
	fastTcCount >= 1 && currentTime - txTcTime >= txFastTimeout	3	fastTcCount -= 1;	
	fastTcCount == 0 && currentTime - txTcTime >= txSlowTimeout	4	—	
	—	5	—	START
FINAL	—	6	frame.cv = myTopoInfo.myChecksum.valid; frame.checksum = myTopoInfo.myChecksum.value; txTcTime = currentTime; EnqueueRi(0, Q_TX_RS, frame); EnqueueRi(1, Q_TX_RS, frame);	START

Row 11.19-1: Always send a fixed number of *txFastTimeout* interval frames after the most recent trigger.
(Do not trigger immediately, however, if a TC frame was transmitted less than *txFastTimeout* ago.)

Row 11.19-2: A trigger during a *txSlowTimeout* interval restarts *txFastTimeout* transmissions immediately.

Row 11.19-3: When initial fast timers expire, update the counter and timer, and then do fast rate transmissions.

Row 11.19-4: After the fast timers complete, update counter and timer, and then do slow rate transmissions

Row 11.19-5: Otherwise, continue checking until a timer expires.

Row 11.19-6: The transmission-interval timer is cleared and a TC frame is sent on each ringlet.

11.6.10 ReceiveTcFrame state machine

This subclause specifies the state machine for handling the receipt of TC frames at a station and related requirements.

The receipt of a TC frame on a ringlet causes the MAC control sublayer to update its topology database, provided an SF condition is not present on the receiving span.

11.6.10.1 ReceiveTcFrame state machine constants

NULL
Q_RX_TC
See 11.2.8.

11.6.10.2 ReceiveTcFrame state machine variables

frame
See 11.2.8.

misablingDefect[ri]
See 11.2.9.

myTopoInfo
See 11.2.5.

ri
topologyValid
transmitTcFrame
See 11.2.3.

11.6.10.3 ReceiveTcFrame state machine routines

DequeueRi(ri, queue)
See 11.2.8.

11.6.10.4 ReceiveTcFrame state table

Upon receipt of a TC frame, ReceiveTcFrame updates the topology database. If the topology is stable, changes in the received TC frames also trigger the transmission of additional TC frames, as specified in Table 11.20.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.20—ReceiveTcFrame state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = DequeueRi(ri, Q_RX_TC)) != NULL	1	—	TEST
	—	2	—	START
TEST	frame.ttlBase == 1 && frame.ttl == 1	3	—	NEXT
	—	4	—	START
NEXT	!topologyValid	5	—	FINAL
	myTopoInfo.myChecksum.valid && myTopoInfo.neighborCheck[ri].valid != frame.cv	6	transmitTcFrame = TRUE;	
	myTopoInfo.myChecksum.valid && myTopoInfo.neighborCheck[ri].value != frame.checksum	7		
	—	8	—	
FINAL	—	9	myTopoInfo.neighborCheck[ri].value= frame.checksum; myTopoInfo.neighborCheck[ri].valid = frame.cv;	START

Row 11.20-1: Processing starts after a TC frame is received.

Row 11.20-2: Otherwise, remain in START.

Row 11.20-3: A valid TC frame is received from the neighbor station; continue checking.

Row 11.20-4: Otherwise, return to START.

Row 11.20-5: For timeliness, TC frames are accepted before this station's topology is validated.

Row 11.20-6: If the neighbor's checksum validity changes, trigger fast-timer TC frame transmissions.

Row 11.20-7: If the neighbor's checksum changes, trigger fast-timer TC frame transmissions.

Row 11.20-8: Otherwise, additional TC frame transmissions are not triggered.

Row 11.20-9: The contents of the TC frame update the topology database; return to START.

11.6.11 Transmit rules for ATD frames

ATD frames shall be transmitted independently by each station, upon expiration of the periodic *atdTimeout* timer started at station initialization. ATD frames may also be sent upon any change of their content.

In order to allow other stations time to react to changes in advertised station attributes, stations shall advertise changed values for attributes that use resources (i.e., ATT_STATION_BW, ATT_STATION_NAME, ATT_STATION_SEC_MAC) as follows:

- a) Before adding a resource (e.g., more reserved bandwidth or secondary MAC address), advertise the resource usage, then start using it after at least RRTT + 10 milliseconds.

- b) After removing a resource usage (e.g., station name), wait at least RRTT + 10 milliseconds before removing advertisement of it.

For attributes for which there is a resource total that should not be exceeded (e.g., reserved bandwidth), there is no control in the MAC to prevent this condition from occurring.

NOTE 1—An implementation can issue an alarm on passing the threshold(s), if it so desires.

NOTE 2—The means of determining RRTT are implementation specific. Either of the following are known to be sufficient:

- a) Transmit an advertisement and wait for that advertisement frame to return to this source station.
- b) Transmit a flush and wait for that flush frame to return to this source station.

For attributes for which there are one or two items of each resource value (e.g., station name or secondary MAC address), no station shall claim ownership if another station already claims ownership. Race conditions that allow two stations to advertise the same value shall force both to remove their claim and issue an alarm.

11.6.12 Receive rules for ATD frames

The receipt of an ATD frame on either ringlet from the Q_RX_ATD queue causes the MAC control sublayer to update its topology database. ATD frames shall be ignored under the following conditions:

- a) If received from a station not in the topology database.
- b) If received from a station whose topology database entry is marked INVALID.
- c) If received from a station not at the same location in the topology database as indicated by the *ttl* value of the received ATD frame.

ATT entries in ATD frames are illegal and shall be discarded under the following conditions:

- a) A previous ATT entry of the same type is contained within the frame.
- b) The ATT has an illegal size.
- c) The ATT type is not supported.
- d) An ATT extends beyond the end of frame.

The offset of the next ATT is always based on the offset and reported length of the previous ATT, even if the type is not recognized or the length is different from the expected value.

The above ATT processing rules are based on the definition of the various ATT types (see 11.4).

Upon completion of the above ATD and ATT checks, for the secondary MAC ATT, a copy of the frame is passed through the Q_RX_ATD_SMAC identified queue, to the SecondaryUpdate state machine that interprets its content. For other ATT types, the information contained in the ATT is written into the topology database.

For the station bandwidth ATT, each station sums up the reserved bandwidth reported by each station to determine the total reserved subclassA0 bandwidth on each ringlet, for use by the fairness module (see 10.4), as specified by Equation (11.16).

```

void
ComputeTotalUnreservedRate()
{
    int ringlet, otherRi, hops, resRate[2];

    resRate[0] = myTopoInfo.reservedRate[0];
    resRate[1] = myTopoInfo.reservedRate[1];
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        otherRi = Other(ringlet);
        for (hops = 1; hops <= ringInfo.totalHopsRx[ringlet]; hops += 1)
        {
            resRate[otherRi] += topoEntry[ringlet][hops].reservedRate[otherRi];
            if (ringInfo.topoType == OPEN_RING)
                resRate[ringlet] += topoEntry[ringlet][hops].reservedRate[ringlet];
        }
    }
    ringInfo.unreservedRate[0] = LINK_RATE - resRate[0]) * ageCoef * rateCoef;
    ringInfo.unreservedRate[1] = LINK_RATE - resRate[1]) * ageCoef * rateCoef;
}

```

(11.16)

In the event that the total unreserved bandwidth is less than zero on either ringlet, an excess reserved rate defect shall be declared by setting *excessReservedRateDefect* to TRUE. This defect is cleared when the excess reserved bandwidth is removed.

11.6.13 SecondaryUpdate state machine

When the topology database is updated, the SecondaryUpdate state machine is activated to validate modified secondary MAC addresses. Validation is performed when *topologyValid* is set by the TopologyValidation state machine.

If any secondary MAC address in the reachable topology is duplicated by any primary or any other secondary MAC address on the ring, *duplicateSecMacAddressDefect* is set. If the number of non-duplicated secondary MAC addresses exceeds MAX_SEC_MAC, *maxSecMacAddressDefect* is set (see 11.2.9).

The SecondaryUpdate state machine may optionally be implemented. If this state machine is not implemented, then the topology database is not updated with any secondary MAC information.

11.6.13.1 SecondaryUpdate state machine constants

MAX_SEC_MAC

See 11.2.2.

NULL

See 11.2.8.

Q_RX_ATD_SMAC

Selects a queue of received ATD control frames with secondary MAC address ATT parameters.

11.6.13.2 SecondaryUpdate state machine variables

adminRequestSecondaryMac

An administrative request value, serviced and cleared by the MAC.

macAddress[2]—secondary MAC addresses

NULL—(Otherwise.)

count

Number of usable secondary MAC addresses.

duplicateSecMacAddressDefect

See 11.2.9.

extract

An data structure capable of holding the secondary MAC address ATT parameters.

macAddress[2]—secondary MAC addresses

NULL—(Otherwise.)

frame

The contents of an ATD frame received from the ring (see 11.3.5 and 11.4).

hops

Index of topology database entry that is updated by the received ATD frame.

i

Counter for looping through secondary MAC addresses.

info

An internal copy of two secondary MAC addresses.

maxSecMacAddressDefect

See 11.2.9.

myTopoInfo

See 11.2.5.

needSecondaryMacValidation

See 11.2.3.

rid

See 11.2.3.

ringInfo

See 11.2.4.

tempHopsRx[rid]

See 11.2.3.

topoEntry

See 11.2.6.

topologyValid

See 11.2.3.

11.6.13.3 SecondaryUpdate state machine routines

Broadcast(macAddress)

See 7.2.3.

ExtractSecondaryMac(frame)

Extracts the secondary MAC addresses from the ATT entry within an ATD frame.

FindIndex(frame)

See 11.2.7.

Max(valueA, valueB)

See 11.2.8.

SecondaryMacCount(test)

For nonzero and zero *test* values, indicates the number of matching secondary MAC addresses and non-null secondary MAC addresses in the topology database, as specified by Equation (11.17).

```
int SecondaryMacCount (test) {  
    int ringlet, ringCount, hops, sec, total = 0;  
  
    total = (myTopoInfo.macAddress == test);  
    for (sec = 0; sec < 2; sec += 1)  
        total += (myTopoInfo.secMac[sec].state != NOT_USED &&  
                  myTopoInfo.secMac[sec].address == test);  
    ringCount = (ringInfo.topoType == CLOSED_RING) ? 1 : 2;  
    for (ringlet = 0; ringlet < ringCount; ringlet += 1)  
        for (hops = 1; hops <= ringInfo.totalHopsRx[ringlet]; hops += 1)  
        {  
            total += (topoEntry[ringlet][hops].macAddress == test);  
            for (sec = 0; sec < 2; sec += 1)  
                total += (topoEntry[ringlet][hops].secMac[sec].state != NOT_USED  
                          && topoEntry[ringlet][hops].secMac[sec].address == test);  
        }  
    return(total);  
}
```

(11.17)

SecondaryMacMarkInUse()

Marks the confirmed secondary MAC state, as specified by Equation (11.18).

```
int SecondaryMacMarkInUse (amount) {  
    int sec, ringCount, ringlet, maxHops, hops0, hops1, total = 0;  
    int allow, excess, newState;  
  
    excess = Max(0, amount - MAX_SEC_MAC);  
    for (sec = 0; sec < 2; sec += 1)  
    {  
        if (myTopoInfo.secMac[sec].state==NOT_USED ||  
            myTopoInfo.secMac[sec].state == DUPLICATE)  
            continue;  
        total += 1;  
        if (amount != 0)  
            myTopoInfo.secMac[sec].state = OK_IN_USE;  
    }  
    ringCount = (ringInfo.topoType == CLOSED_RING) ? 1 : 2;  
    for (ringlet = 0; ringlet < ringCount; ringlet += 1)  
    {  
        maxHops = ringInfo.totalHopsRx[ringlet];  
        for (hops0 = 1, hops1 = maxHops; hops0 <= maxHops; hops0 += 1, hops1 -= 1)  
            for (sec = 0; sec < 2; sec += 1)  
            {  
                if (allow > 0)  
                    myTopoInfo.secMac[sec].state = OK_IN_USE;  
                if (allow < 0)  
                    myTopoInfo.secMac[sec].state = NOT_USED;  
                if (allow == 0)  
                    myTopoInfo.secMac[sec].state = DUPLICATE;  
                if (allow > excess)  
                    myTopoInfo.secMac[sec].state = OK_IN_USE;  
                if (allow < -excess)  
                    myTopoInfo.secMac[sec].state = NOT_USED;  
                if (allow == -excess)  
                    myTopoInfo.secMac[sec].state = DUPLICATE;  
                if (allow > excess && allow < -excess)  
                    myTopoInfo.secMac[sec].state = DUPLICATE;  
            }  
        }  
    }  
}
```

(11.18)

```

        if (topoEntry[ringlet][hops0].secMac[sec].address==NOT_USED ||
            topoEntry[ringlet][hops0].secMac[sec].address == DUPLICATE)
            continue;
        allow = (amount == 0 || (total < MAX_SEC_MAC &&
            (excess == 0 || !SecondaryMacReject(ringlet,hops,sec)));
        excess -= (excess && !allow);
        total += allow;
        if (amount == 0)
            continue;
        newState = (allow ? OK_IN_USE : OK_TO_USE);
        topoEntry[ringlet][hops0].secMac[sec].state = newState;
        if (ringInfo.topoType == CLOSED_RING)
            topoEntry[ringlet][hops1].secMac[sec].state = newState;
    }
}
return(total);
}

```

SecondaryMacValidation()

Marks the topology database's duplicate secondary MAC addresses, as specified by Equation (11.19).

```

int SecondaryMacValidation() (11.19)
{
    uint64_t macAddress;
    int ringlet, ringCount, hops, sec, total = 0, countEnabled;

    for (sec = 0; sec < 2; sec += 1)
    {
        if (myTopoInfo.secMac[sec].state == NOT_USED ||
            SecondaryMacCount(myTopoInfo.secMac[sec].address) == 1)
            continue;
        myTopoInfo.secMac[sec].state = DUPLICATE;
        total += 1;
    }
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        countEnabled = (ringlet == 0 || ringInfo.topoType == OPEN_RING);
        for (hops = 1; hops <= ringInfo.totalHopsRx[ringlet]; hops += 1)
            for (sec = 0, total = 0; sec < 2; sec += 1)
            {
                macAddress = topoEntry[ringlet][hops].secMac[sec].address;
                if (topoEntry[ringlet][hops].secMac[sec].state == NOT_USED ||
                    SecondaryMacCount(macAddress) == 1)
                    continue;
                topoEntry[ringlet][hops].secMac[sec].state = DUPLICATE;
                total += countEnabled;
            }
    }
    return(total);
}

```

SecondaryMacReject(ringlet, hops, sec)

Returns an indication of whether a given secondary MAC address in the topology database is not used. The indication value is implementation dependent and beyond the scope of this standard.
(One of many possible implementations always returns FALSE.)

11.6.13.4 SecondaryUpdate state table

The SecondaryUpdate state machine is specified in Table 11.21. This state machine handles the topology database update resulting from a received ATD frame or secondary MAC address configuration at the local station.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.21—SecondaryUpdate state table

Current state		Row	Next state	
state	condition		action	state
START	(info = adminRequestSecondaryMac) != NULL	1	i = 0;	ADMIN
	(frame = Dequeue(Q_RX_ATD_SMAC)) != NULL	2	rid = frame.ri; hops = FindIndex(frame.ttl); extract = ExtractSecondaryMac(frame); i = 0;	ATD
	—	3	—	CHECK
ADMIN	i >= 2	4	info = NULL;	CHECK
	!Broadcast(info.secondaryMac[i]) && myTopoInfo.secMac[i].address != info.secondaryMac[i]	5	myTopoInfo.secMac[i].address = info.macAddress[i]; myTopoInfo.secMac[i].state = OK_TO_USE; needSecondaryMacValidation = TRUE; i += 1;	ADMIN
	Broadcast(info.secondaryMac[i]) && myTopoInfo.secMac[i].address != 0	6	myTopoInfo.secMac[i].address = 0xFFFFFFFFFFFF; myTopoInfo.secMac[i].state = NOT_USED; needSecondaryMacValidation = TRUE; i += 1;	
	—	7	i += 1;	
	extract != NULL	8	info = extract;	ATT
ATT	—	9	info.macAddress[0] = 0xFFFFFFFFFFFF; info.macAddress[1] = 0xFFFFFFFFFFFF;	
	i >= 2	10	—	CHECK
	topoEntry[rid][hops].valid && !Broadcast(info.macAddress[i]) && topoEntry[rid][hops].secMac[i].address != info.macAddress[i]	11	topoEntry[rid][hops].secMac[i].address = info.macAddress[i]; topoEntry[rid][hops].secMac[i].state = OK_TO_USE; needSecondaryMacValidation = TRUE; i += 1;	ATT
	topoEntry[rid][hops].valid && Broadcast(info.macAddress[i]) && topoEntry[rid][hops].secMac[i].address != 0	12	topoEntry[rid][hops].secMac[i].address = 0xFFFFFFFFFFFF; topoEntry[rid][hops].secMac[i].state = NOT_USED; needSecondaryMacValidation = TRUE; i += 1;	
	—	13	i += 1;	

Table 11.21—SecondaryUpdate state table (continued)

Current state		Row	Next state	
state	condition		action	state
CHECK	topologyValid && needSecondaryMacValidation	14	needSecondaryMacValidation = FALSE; duplicateSecMacAddressDefect = SecondaryMacValidation(); count = SecondaryMacMarkInUse(0); maxSecMacAddressDefect = (count > MAX_SEC_MAC); SecondaryMacMarkInUse(count);	START
	—	15	—	

Row 11.21-1: The receipt of an administrative request causes processing of secondary MAC addresses.

Row 11.21-2: The receipt of an ATD frame causes processing of secondary MAC addresses.

Row 11.21-3: Continue checking for new activation inputs.

Row 11.21-4: Stop after all secondary MAC addresses have been processed.

Row 11.21-5: Process revised secondary MAC addresses.

Row 11.21-6: Process nullified MAC addresses.

Row 11.21-7: Process another secondary MAC address.

Row 11.21-8: If an ATT entry was provided, use its *secondaryMacAddress* field values.

Row 11.21-9: Otherwise, the broadcast-valued default *secondaryMacAddress* field values are assumed.

Row 11.21-10: Stop after all secondary MAC addresses have been processed.

Row 11.21-11: Process revised secondary MAC addresses.

Row 11.21-12: Process nullified MAC addresses.

Row 11.21-13: Process another secondary MAC address.

Row 11.21-14: Secondary MAC validation is performed when the topology becomes valid.

Row 11.21-15: Secondary MAC validation is deferred until the topology has become valid.

11.6.14 TimingLrttFrame state machine

The LRTT value is used for computing the FRTT (see Clause 10). Whenever a new topology is validated, a station deploying conservative-mode fairness transmits unicast loop round-trip time (LRTT) request frames to all other stations on each ringlet. This may optionally be done thereafter, with at least 10 seconds between successive measurement cycles. If the topology becomes unstable while the LRTT measurement is taking place, the LRTT measurement is aborted.

Stations shall respond to an LRTT request by sending an LRTT response via the opposing ringlet. On receiving its LRTT response, a station computes the LRTT, as specified by Equation (11.20), and maintains this value in its topology database.

$$\text{lastRcvdLrtt} = (\text{currentTime} - \text{latencyTimestamp}) + (11.20) \\ (\text{hopsToResponder} * \text{advertisingInterval}) - (\text{tailLatencyOut} - \text{tailLatencyIn})$$

A station can measure or estimate the amount of latency added between reception of the LRTT request frame and transmission of the LRTT response frame, and it returns that latency amount with the response frame.

11.6.14.1 TimingLrttFrame state machine definitions

COMPLETION_TIME

Time allowed for completion of LRTT measurement.

Value—100 milliseconds.

NULL

Q_TX_RS

See 11.2.8.

11.6.14.2 TimingLrttFrame state machine variables

advertisingInterval

currentTime

See 11.2.8.

frame

See 11.2.3.

lrttComplete

Indication if LRTT measurement is completed.

lrttRequest

See 11.2.3.

lrttRetxTime

The time at which the last retransmitted LRTT frame was sent.

lrttRetxTimeout

Retransmit timer for LRTT before LRTT measurement is completed.

Value—200 milliseconds

lrttTime

The time at which the *lrttTimeout*-specified delay is due to expire.

lrttTimeout

The delay time to claim LRTT measurement defect.

Value—1 second

lrttIncompleteDefect

See 11.2.9.

lrttRequest

See 11.2.3.

myTopoInfo

See 11.2.5.

ringInfo

See 11.2.4.

rxFrame

Contents of an LRTT frame received from the ring.

tempHopsRx[rid]

See 11.2.3.

topoEntry

See 11.2.6.

topologyStable

topologyValid

See 11.2.3.

txFrame

Contents of an LRTT frame transmitted to the ring.

11.6.14.3 TimingLrttFrame state machine routines

Dequeue(queue)

See 11.2.8.

EnqueueRi(rid, queue, frame)

See 11.2.8.

LrttFrameReceived()

This routine provides a newly received LRTT frame.

LrttRequestTransmit()

Transmit LRTT request frame to start LRTT measurement, as specified by Equation (11.21).

```
void
LrttRequestTransmit(newRequest)
{
    int ringlet, txRinglet, hops;
    Frame frame;

    for(ringlet = 0, ringlet < 2, ringlet += 1)
    {
        for (hops = 1; hops <= ringInfo.totalHopsRx[ringlet]; hops += 1)
        {
            if(topologyStable == FALSE ||
               topoEntry[ringlet][hops].valid == FALSE)
                return;
            if((newRequest || topoEntry[ringlet][hops].lrtt == NULL))
            {
                frame.controlType = CT_LRTT_REQ;
                frame.da = topoEntry[ringlet][hops].macAddress;
                frame.latencyTimestamp = currentTime;
                frame.tailLatencyIn = 0;
                frame.tailLatencyOut = 0;
                if(newRequest)
                    topoEntry[ringlet][hops].lrtt = NULL;
                EnqueueRi(Other(ringlet), Q_TX_RS, frame);
            }
        }
    }
}
```

(11.21)
*LrttResponseHandle(frame)*Handle LRTT response frame, as specified by Equation (11.22). The LRTT measurement to a destination station from a source station on ringlet *rid* (returning on ringlet *Other(rid)*) is stored in the entry of the topology database corresponding to the ringlet from which the LRTT response frame is received (*Other(rid)*), and to the number of hops separating the destination station from the source station.

```
void
LrttResponseHandle(frame)
{
    int hops, index, ringlet, latency;

    latency = currentTime - frame.latencyTimestamp;
    if (latency > COMPLETION_TIME || topologyStable == FALSE)
        return;
    index = (frame.ttlBase - frame.ttl) + 1;
    topoEntry[frame.ri][index].lrtt = latency +
        (index*advertisingInterval) - (frame.tailLatencyOut-frame.tailLatencyIn);
    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hops = 1; hops <= ringInfo.totalHopsRx[ringlet]; hops += 1)
            if(topoEntry[ringlet][hops].valid == FALSE ||
               topoEntry[ringlet][hops].lrtt == NULL)
                return;
    lrttComplete = TRUE;
}
```

(11.22)

StartLrttReTxTimer()
Start *lrttReTxTimeout* timer.

11.6.14.4 TimingLrttFrame state table

The TimingLrttFrame state machine specified in Table 11.22 handles transmission and reception of LRTT frames.

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 11.22—TimingLrttFrame state table

Current state		Row	Next state	
state	condition		action	state
START	—	1	lrttTime = currentTime; lrttReTxTime = currentTime; lrttRequest = FALSE;	FIRST
FIRST	!topologyValid	2	—	START
	ringInfo.totalHopsRx[0] == 0 && ringInfo.totalHopsRx[1] == 0 && lrttRequest && myTopoInfo.conservativeMode	3	lrttComplete = TRUE; lrttIncompleteDefect = FALSE;	
	lrttRequest && myTopoInfo.conservativeMode	4	lrttComplete = FALSE; LrttRequestTransmit(TRUE);	
	(rxFrame = Dequeue(Q_RX_LRTT_REQ)) !=NULL	5	—	
	(rxFrame = Dequeue(Q_RX_LRTT_RSP)) !=NULL	6	—	
	!lrttComplete && currentTime - lrttReTxTime >= lrttReTxTimeout	7	LrttRequestTransmit(FALSE); lrttReTxTime = currentTime;	
	!lrttComplete && currentTime - lrttTime >= lrttTimeout	8	lrttIncompleteDefect = TRUE;	
	—	9	—	FIRST
NEAR	rxFrame.controlType == CT_LRTT_REQ	10	txFrame.tailLatencyIn = currentTime; txFrame.controlType = CT_LRTT_RSP; txFrame.da = rxFrame.sa; txFrame.latencyTimestamp = rxFrame.latencyTimestamp; txFrame.tailLatencyOut = currentTime; EnqueueRi(Other(rxFrame.ri), Q_TX_RS, txFrame);	FIRST
	—	11	LrttResponseHandle(rxFrame);	FINAL
FINAL	lrttComplete	12	lrttIncompleteDefect = FALSE;	START
	—	13	—	FIRST

Row 11.22-1: Reset the timers in preparation for LRTT measurements.

Row 11.22-2: Wait until the topology has been validated.

Row 11.22-3: The special case of a single-station ring completes immediately.

Row 11.22-4: A conservative-mode fairness station starts LRTT measurement when *lrttRequest* is TRUE.

Row 11.22-5: If an LRTT request frame is received, proceed to NEAR state to process it.

Row 11.22-6: If an LRTT response frame is received, proceed to NEAR state to process it.

Row 11.22-7: If *lrttReTxTimeout* is expired and LRTT measurement is not complete, an LRTT request is retransmitted. The parameter of *LrttRequestTransmit* is set to FALSE because this is not a new request.

Row 11.22-8: If the LRTT timeout expires before the measurement complete, mark an LRTT incomplete defect.

Row 11.22-9: Otherwise do nothing.

Row 11.22-10: If an LRTT request frame is received, the current time stamp is placed into *tailLatencyIn* and processed in the *lrttRequestHandle()* routine. A station shall respond within 100 milliseconds.

Row 11.22-11: If an LRTT response frame is received, the frame is processed in *lrttResponseHandle()*. If the latency of the response frame is greater than 100 milliseconds, the response frame shall be ignored.

Row 11.22-12: If the LRTT measurement is complete, the LRTT timers are cleared.

Row 11.22-13: Otherwise, continue LRTT processing.

11.7 Protocol Implementation Conformance Statement (PICS) proforma for Clause 11²³

11.7.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 11, Topology discovery and protection, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

11.7.2 Identification

11.7.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

11.7.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, Topology discovery and protection
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

²³Copyright release for PICS proformas: Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

11.7.3 PICS tables for Clause 11**11.7.3.1 State machines**

Item	Feature	Subclause	Value/Comment	Status	Support
SM1	ReceiveMonitor	11.6.2	ReceiveMonitor state machine supported	M	Yes []
SM2	TopologyControl	11.6.3	TopologyControl state machine supported	M	Yes []
SM3	ParseTpFrame	11.6.4	ParseTpFrame state machine supported	M	Yes []
SM4	ProtectionUpdate	11.6.5	Protection update state machine supported	M	Yes []
SM5	TopologyValidation	11.6.6	Topology validation state machine supported	M	Yes []
SM6	TransmitTpFrame	11.6.7	Transmit TP frame state machine supported	M	Yes []
SM7	ReceiveTpFrame	11.6.8	Receive TP frame state machine supported	M	Yes []
SM8	TransmitTcFrame	11.6.9	Transmit TC frame state machine supported	M	Yes []
SM9	ReceiveTcFrame	11.6.10	Receive TC frame state machine supported	M	Yes []
SM10	TransmitAtdFrame	11.6.11	Transmit ATD frame state machine supported	M	Yes []
SM11	ReceiveAtdFrame	11.6.12	Receive ATD frame state machine supported	M	Yes []
SM12	SecondaryUpdate	11.6.13	Secondary MAC update state machine supported	O	Yes [] No []
SM13	TimingLrttFrame	11.6.14	Timing of LRTT frames state machine supported	M	Yes []

11.7.3.2 Frame formats

Item	Feature	Subclause	Value/Comment	Status	Support
FF1	TP frame	11.3.1	TP frames are transmitted by the MAC in the format defined	M	Yes []
FF2	TC frame	11.3.2	TC frames are transmitted by the MAC in the format defined	M	Yes []
FF3*	Conservative rate computation	10.4.4	Does the MAC deploy the conservative rate computation method?	O	Yes [] No []

Item	Feature	Subclause	Value/Comment	Status	Support
FF4	LRTT request frame	11.3.3	LRTT request frames are transmitted by the MAC in the format defined	FF3:M	Yes []
FF5	LRTT response frame	11.3.4	LRTT response frames are transmitted by the MAC in the format defined	M	Yes []
FF6	Tail latency in field setting	11.3.4	Responder clears this field to zero	M	Yes []
FF7	Tail latency out field setting	11.3.4	Responder clears this field to zero	M	Yes []
FF8	ATD frame	11.3.5	ATD frames are transmitted by the MAC in the format defined	M	Yes []
FF9	ATT advertising, non-default	11.3.5	Non-default values advertised	M	Yes []
FF10	ATT advertising, default	11.3.5	Default values advertised	O	Yes [] No []
FF11	ATT not included	11.3.5	Default values for conditional ATTs restored	M	Yes []
FF12	Type dependent length	11.3.5	Length of type dependent value information less than 1024 bytes	M	Yes []

11.7.3.3 ATT entries

Item	Feature	Subclause	Value/Comment	Status	Support
ATT1	Weight ATT	11.4.1	Weight ATT supported	O	Yes [] No []
ATT2*	Station bandwidth ATT	11.4.2	Station bandwidth ATT supported	O	Yes [] No []
ATT3	Reserved bandwidth within station bandwidth ATT	11.4.2	Uses same normalization as in fairness frames	ATT2:M	Yes []
ATT4	Excess reserved rate defect	11.4.2	Unreserved bandwidth less than zero triggers defect	M	Yes []
ATT5	Station settings ATT	11.4.3	Station settings ATT supported	O	Yes [] No []
ATT6	Station name ATT	11.4.4	Station name ATT supported	O	Yes [] No []
ATT7*	Station management ATT	11.4.5	Station management ATT supported	O	Yes [] No []
ATT8	Managed entity	11.4.5	Single managed entity attached to ring multiple times	ATT7:O	Yes [] No []
ATT9	Station interface index ATT	11.4.6	Station interface index ATT supported	O	Yes [] No []

Item	Feature	Subclause	Value/Comment	Status	Support
ATT10*	Secondary MAC ATT	11.4.7	Secondary MAC ATT supported	O	Yes [] No []
ATT11	First secondary MAC address setting	11.4.7	Set to broadcast address if no secondary MAC addresses configured	ATT10:M	Yes []
ATT12	Second secondary MAC address setting	11.4.7	Set to broadcast address unless two secondary MAC addresses configured	ATT10:M	Yes []
ATT13	Organization-specific ATT	11.4.8	Organization-specific ATT supported	O	Yes [] No []

11.7.3.4 Protection

Item	Feature	Subclause	Value/Comment	Status	Support
PP1	Protection for relaxed data frames	11.1.6.3	Within combination of detection and restoration times	M	Yes []
PP2	Protection for strict data frames	11.1.6.3	Within combination of detection, restoration, and topology stabilization times	M	Yes []
PP3	Steering protection	11.1.6.1	Steering protection	M	Yes []
PP4*	Wrapping protection	11.1.6.2	Wrapping protection	O	Yes [] No []
PP5	Revertive operation	11.2.3	Revertive operation by default	M	Yes []
PP6	Non-revertive operation	11.2.3	Nonrevertive operation supported	O	Yes [] No []
PP7	Failure of transit path	11.6.5.2	Station does not wrap on both spans	PP4:M	Yes []

11.7.3.5 Topology discovery

Item	Feature	Subclause	Value/Comment	Status	Support
TD1	Topology discovery in maximum size ring	11.6.8	Station can complete topology discovery for stable ring of MAX_STATIONS	M	Yes []
TD2	Persistent topology inconsistency	11.6.6.1	Adding/dropping data to/from the ring may be impacted after 10 seconds of topology inconsistency	O	Yes [] No []

11.7.3.6 ATD frame handling

Item	Feature	Subclause	Value/Comment	Status	Support
ATD1	ATD timer	11.2.3	Range, resolution, and default supported	M	Yes []
ATD2	ATD transmission	11.6.11	Transmitted on initialization and periodically	M	Yes []
ATD3	ATD transmission	11.6.11	Transmitted immediately upon change of content	O	Yes [] No []
ATD4	Claiming ownership of resources through ATTs	11.6.11	No station claims ownership of resources if another station has claimed ownership	M	Yes []
ATD5	Handling of race conditions	11.6.11	Stations all remove their claims and generate defect	M	Yes []
ATD6	ATD reception	11.6.12	ATD frames ignored under listed conditions	M	Yes []
ATD7	Illegal ATD discards	11.6.12	ATD frames discarded under listed conditions	M	Yes []

11.7.3.7 LRTT measurements

Item	Feature	Subclause	Value/Comment	Status	Support
LRTT1	LRTT request processing	11.6.14	All stations process LRTT requests	M	Yes []
LRTT2	LRTT response time	11.6.14.4	Response to LRTT request within 100 ms	M	Yes []
LRTT3	No late LRTT response	11.6.14.4	If response not received within 100 ms, then response ignored	M	Yes []
LRTT4	LRTT re-measurement	11.6.14.4	Remeasurement done after at least 10 seconds	O	Yes [] No []

12. Operations, administration, and maintenance (OAM)

12.1 Overview

This clause provides operations, administration, and maintenance (OAM) functions supported by RPR stations. The methods defined are intended to be scalable, to cause insignificant overhead for ring traffic, and to cause insignificant overhead on software and hardware devices. The protocol resides in the MAC control sublayer, as shown in the shaded OAM region of Figure 12.1.

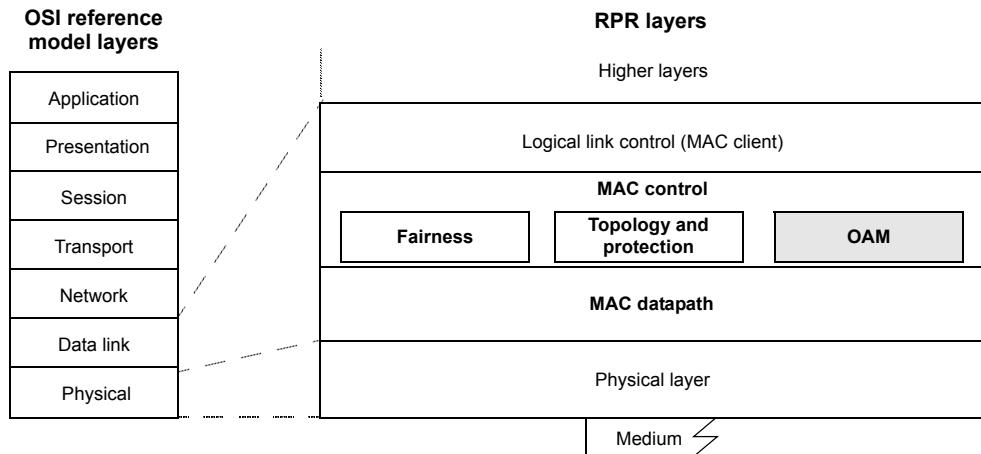


Figure 12.1—OAM control entity relationship to the ISO/IEC OSI reference model

The services and features provided are as follows:

- a) Determine/validate connectivity between any two stations on the ring.
- b) Determine/validate transit path operation for any service class.
- c) Operate without relying on a master station.
- d) Provide a mechanism to help in misorder prevention.
- e) Provide a mechanism to notify other stations on the ring that some of the SDB entries associated with this station are invalid.
- f) Provide a mechanism to interconnect rings with fast failure recovery.

It is not within the scope of the OAM functions to detect intermittent failures.

12.1.1 Protocol overview

Management functional areas pertinent to RPR are as follows:

- a) configuration management
- b) fault management
- c) performance management

Configuration management exercises control over, identifies, collects data from, and provides data to stations and the connections between stations. Configuration management is responsible for the installation of stations, their interconnection into a network, and provisioning.

Fault (or maintenance) management enables the detection, isolation, and correction of abnormal operation of the stations and the network. It is responsible for detecting and processing any faults, as well as to report them to the management system.

Performance management evaluates and reports upon the behavior and effectiveness of stations and the network. Performance management provides mechanisms to measure service quality, by monitoring the system performance. It also reports statistics information to the management system.

In order to improve fault and performance management capability, e.g., to allow fault detection, some in-band OAM functions are provided.

OAM functions in a network are performed on hierarchical levels. Different types of interfaces support different levels of OAM functionality. A physical layer based on SONET/SDH includes extensive OAM functions, whereas a physical layer based on PacketPHYs has a lower level of OAM support.

This standard reuses existing in-band OAM mechanisms provided by the PHY layers. In-band OAM functionality used in upper layers is outside the scope of this standard.

12.1.2 OAM functions supported by RPR

The OAM functions are based on special frames, sent between stations on a ring, that test the operational status of a path between stations.

The OAM frame types supported are as follows:

- a) Echo—For on-demand connectivity monitoring and fault localization on the path between stations, or for client determination of LRTT.
- b) Flush—For client-controlled misorder prevention when changing preferred ring direction of a given conversation, or for client determination of RRTT.
- c) Organization-specific—For encoding organization-specific OAM information.
- d) SAS notify—For client-controlled notification to other stations on the ring that some of the SDB entries associated with this station are invalid.
- e) PIRC management—For status exchange and notification between the mate stations.

All stations support echo request/response frames and flush frames.

12.1.3 Fault management

Fault management includes alarm surveillance, fault localization, fault correction, and testing. Alarm surveillance provides the capability to monitor failures detected in stations. In support of alarm surveillance, stations perform checks on hardware and software in order to detect failures, and generate alarms for such failures.

Fault localization determines the root cause of a failure. In addition to the initial failure information, it can use failure information from other entities in order to correlate and localize the fault.

Fault correction is responsible for the repair of a fault and for the control of procedures that use redundant resources to replace equipment or facilities that have failed. In the cases of fiber cuts or station failures, protection switching restores service.

Testing performs repair functions using testing and diagnostic routines. Testing is characterized as the application of signals/messages and their measurement. Echo request/response is one example of a testing routine.

The fault management mechanisms are useful to check the reachability at the MAC layer between two stations on the ring, especially when there are some failures (e.g., one station on the ring steals frames addressed to other stations) that are not detected at layer 1.

This clause defines one fault management mechanism. It is an on-demand in-service echo request/response mechanism (see 12.1.4) used for troubleshooting the RPR network (reactive mechanism).

In addition, Annex K describes a method for using echo frames for fault detection (proactive mechanism) by continuous connectivity verification between any pair of stations.

12.1.4 Echo operations

This standard allows a station to request an echo request/response operation to a specified destination in order to check the reachability of a station. An echo operation allows a frame to be inserted at one station in the ring, and an echo response returned by another station through the same or opposite ringlet, as illustrated in Figure 12.2. Service primitive parameters allow the client to specify the absolute, reverse, or default ringlet for the response (see Table 12.2).

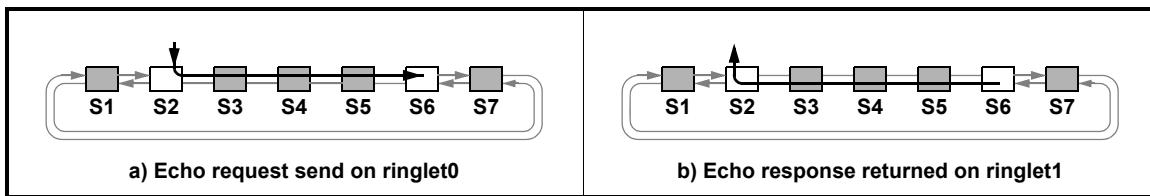


Figure 12.2—Echo request/response operations

Echo request/response frames can be assigned to any service class, and contain user-specified bytes up to the maximum permitted frame size. The echo request *userData* is copied into the response frame.

12.1.5 Flush operations

A flush has the effect of clearing the selected ringlet of previously sourced traffic. A flush is expected to be used when changing the ringlet selection algorithm, when revised ringlet selection protocols are necessary to access all stations (for steering protection), or to improve bandwidth utilization (for wrap protection).

A flush is a special control frame that is sent from a station to itself, as illustrated in Figure 12.3-a. Although distinct flushes (one for ringlet0 and one for ringlet1) are necessary to ensure complete delivery of all previously sourced traffic, one flush is often sufficient to flush the relevant traffic. The flush frame can also flush all previously sourced traffic from a wrapped ring, as illustrated in Figure 12.3-b.

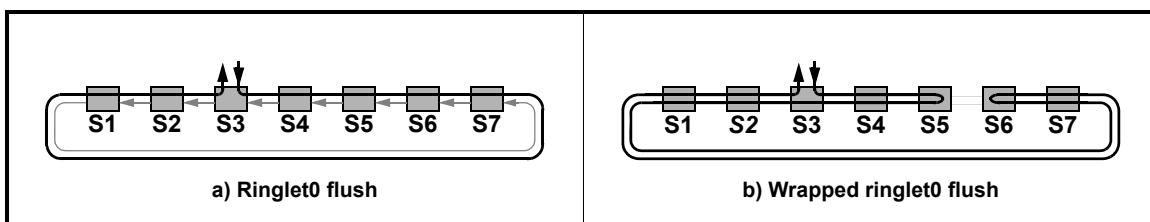


Figure 12.3—Flushing on an unprotected and wrapped rings

The flush frames can be assigned to any service class, with distinct flush action results, as follows:

- a) ClassA. Previously sourced primary transit queue (PTQ) traffic is flushed.
(For a single transit queue station, classB and classC traffic are also flushed).
- b) ClassB or classC. Previously sourced PTQ and secondary transit queue (STQ) traffic is flushed.

Flushing is not possible through a non-wrapping edge station, because traffic is discarded (rather than wrapped) at the endpoints. However, an endpoint-addressed echo can be used in a similar fashion. Annex K illustrates the use of echo frames for flushing transit queues.

12.1.6 Organization-specific operations

An RPR station may use the optional organization-specific OAM capability to implement additional OAM functions not specified by this standard.

If implemented, the organization-specific MAC client service request generates and transmits an organization-specific frame. Similarly, the receipt of an organization-specific frame generates a MAC client service indication.

12.1.7 SAS notify operations

A SAS notify frame allows a station to notify other stations on the ring that some of the SDB entries associated with this station are invalid. The SAS notify frame is broadcast over the ring with a service class of subclassA0.

SAS notify frames are received by all stations, processed by stations that support SAS, and are not passed to their clients. Entries in the SDB that are associated with the originator of the SAS notify frame are removed from the SDB.

12.1.8 PIRC management operations

A PIRC frame allows one interconnected station to notify its status and characteristic to its mate station in the same protection group. The PIRC frame is broadcast over the ring with a service class of subclassA0. PIRC frames are received by all stations and processed by the inter-ring stations.

12.2 Terminology and variables

12.2.1 Common state machine definitions

The following state-machine definitions are used multiple times within this clause:

RR_DEFAULT
RR_REVERSE
RR_RINGLET0
RR_RINGLET1

The enumerated *responseRinglet* values (see 12.3.1.3).

12.2.2 Common variables

The following state machine inputs are used multiple times within this clause.

sesThreshold

Configured. Number of *scfErrors* (see 12.6.1.2) to declare a second as a severely errored second.

Table 12.1 shows the *sesThreshold* values for the PHYs described in this standard.

Range: [1, 512], in integer increments

Default: the next higher integer of $0.000001 * (\text{lineRate} * \text{advertisementRatio})$

Table 12.1—*sesThreshold* example values

PHY	<i>sesThreshold</i>
1 Gb/s PacketPHY	2
OC-48	3
OC-192	13
10 Gb/s PacketPHY	

12.2.3 Common routines

The following state machine routines are used within this clause.

SdbPurge(frame)

Purge selected dynamic entries found in the SDB associated with the *frame.sa* and the *notificationID type* and *identifier* fields (see 12.3.4).

If the value of the type field is:

TYPE_ALL—Remove all entries that have a target address equal to the *sa* of the frame.

TYPE_VID—Remove entries that have a target address equal to the *sa* of the frame and the FDB identifier (FID) mapped from the VID in the *notificationID identifier* field.

TYPE_MSTI—Remove entries that have a target address equal to the *sa* of the frame and a Multiple Spanning Tree Instance (MSTI) value matching the *notificationID identifier* field.

TYPE_VID_MAP—Remove entries that have a target address equal to the *sa* of the frame and FID(s) mapped from the VID values denoted by the *notificationID identifier* field.

TYPE_MSTI_MAP—Remove entries that have a target address equal to the *sa* of the frame and MSTI values denoted by the *notificationID identifier* field.

The *sdbNotifyPurges* counter is incremented by one for each invocation.

NOTE—For SAS to purge entries based on MSTI, it needs access to the client's MSTI-to-FID mapping table in order to acquire the corresponding FID(s) for purging. The means of access is not specified.

See 14.4.3.

12.2.4 Literals and routines defined in other clauses

This clause references the following literals and variables defined in Clause 6:

mac_protection
OAM_ECHO_REQ
OAM_FLUSH_REQ
OAM_ORG_REQ
OAM_SAS_NOTIFY
 Control request *opcode* values.
RI_0

RI_1
 RI_DEFAULT
 ringlet_id

This clause references the following literals, variables, and routines defined in Clause 7:

Dequeue(queue)
Enqueue(queue,frame)
lineRate
myMacAddress
NULL
Q_RX_ECHO_REQ
Q_RX_ECHO_RSP
Q_RX_FLUSH
Q_RX_ORG
Q_RX_SAS_NOTIFY
Q_TX_OAM
Q_TX_RS
Q_TX_SAS

This clause references the following literal defined in Clause 9:

CT_OAM_ECHO_RSP
CT_OAM_SAS_NOTIFY

This clause references the following variable defined in Clause 10:

advertisementRatio

12.3 OAM frame formats

OAM functions use frames in the control frame format (see 9.3) with *controlType* set to one of the assigned OAM codes. Like most other frames, OAM frames are not transmitted over failed spans, even if the span failure is unidirectional.

12.3.1 Echo request/response payload

Figure 12.4 illustrates the echo request/response frame payload, with the *controlDataUnit* portion of the frame indicated.

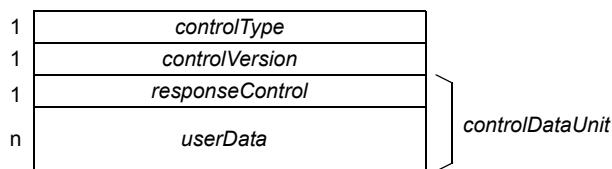


Figure 12.4—Echo request/response frame payload format

The *responseControl* field is shown in Figure 12.5. The *responseControl* field carries the same values in the response as were provided in the request, although no processing is done with them on receipt of the response.

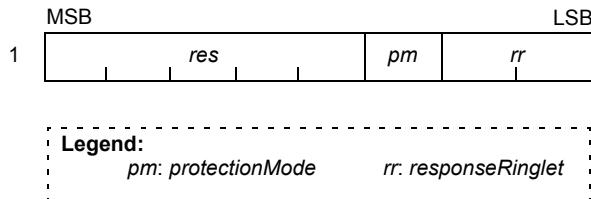


Figure 12.5—*responseControl* field format

12.3.1.1 *res*: A 5-bit reserved field.

12.3.1.2 *protectionMode*: A bit that indicates to the echo request receiving station whether to protect the echo response frame in case of a ring failure. A value of 0 indicates that the echo response is not protected, whereas a value of 1 indicates that the echo response is protected by the MAC. It uses the protection configuration method of the receiving station, and sets *mac_protection* to this value if the configured protection method is wrapping.

12.3.1.3 *responseRinglet*: The 2-bit *responseRinglet* field is interpreted by the receiving station to decide through which ringlet the echo response frame will be transmitted back to the source station. The *responseRinglet* field is encoded as shown in Table 12.2.

Table 12.2—*responseRinglet* values

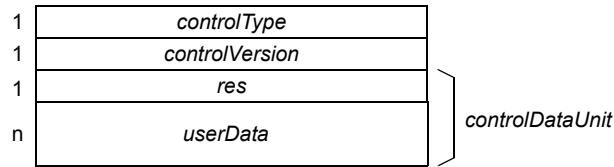
Value	Name	Description
0	RR_RINGLET0	Reply on ringlet0
1	RR_RINGLET1	Reply on ringlet1
2	RR_REVERSE	Reply on reverse ringlet
3	RR_DEFAULT	Reply on default ringlet

The RR_DEFAULT option allows the echo responder to determine the echo response frame path, by setting *ringlet_id* to RR_DEFAULT before invoking ringlet selection. The RR_RINGLET0, RR_RINGLET1, and RR_REVERSE options allow the echo request source station to specify the echo response path.

12.3.1.4 *userData*: A n-byte optional field whose content is determined by the client. When an echo request is received, the receiving station copies the *userData* field from the echo request frame to a generated echo response frame.

12.3.2 Flush frame

Figure 12.6 illustrates the flush frame payload with the *controlDataUnit* portion of the frame indicated.

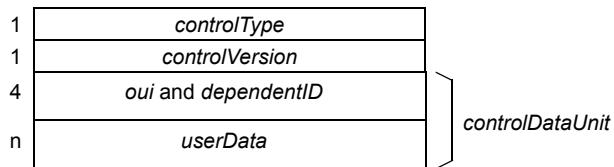
**Figure 12.6—Flush frame payload format**

12.3.2.1 *res*: A reserved byte.

12.3.2.2 *userData*: An n-byte optional field whose content is determined by the client.

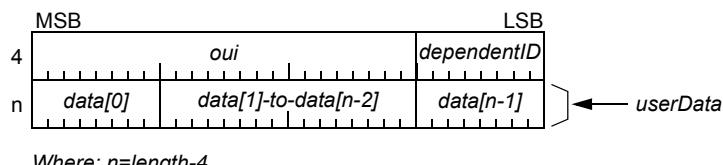
12.3.3 Organization-specific frame

Figure 12.7 illustrates the organization-specific frame payload, with the *controlDataUnit* portion of the frame indicated.

**Figure 12.7—Organization-specific frame payload format**

The organization-specific OAM frame encodes the organization-specific OAM information of the station, and is optional. If a destination station does not support the organization-specific OAM frame, it is allowed to discard the frame without triggering any alarm or indication.

The *controlDataUnit* field for organization-specific frames is illustrated in Figure 12.8.

**Figure 12.8—Organization-specific *controlDataUnit* field format**

12.3.3.1 *oui*: A 24-bit organizationally unique identifier (OUI) field supplied by the IEEE/RAC for the purpose of identifying the organization supplying the (unique within the organization, for this specific context) 8-bit *dependentID*.

12.3.3.2 *dependentID*: An 8-bit field supplied by the *oui*-specified organization. The concatenation of the *oui* and *dependentID* provide a unique (within this context) identifier for each distinct form of the following *userData*.

12.3.3.3 *userData*: An *n*-byte optional field whose content is implied by the preceding *oui/dependentID* value. The content in this field is beyond the scope of this standard.

To reduce the likelihood of error, the mapping of OUI values to the *oui/dependentID* fields are illustrated in Figure 12.9. For the purposes of illustration, specific OUI and *dependentID* example values have been assumed.

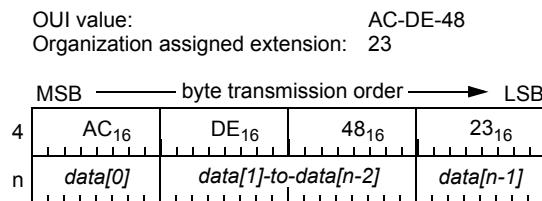


Figure 12.9—Mapping of organization-specific values

12.3.4 SAS notify frame

The variable-length SAS notify frames are identified by the *controlType* field value of CT_OAM_SAS_NOTIFY and a *controlVersion* value of 0. The SAS notify frame contents include information about which sending station SDB entries are invalid. Figure 12.10 illustrates the SAS notify frame payload with the *controlDataUnit* portion of the frame indicated.

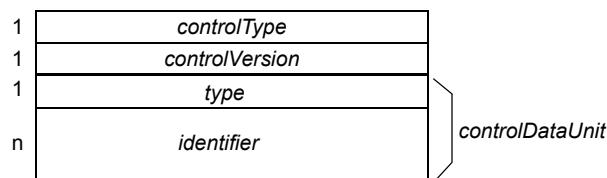


Figure 12.10—SAS notify frame payload format

The header fields (specified in 9.3, but not illustrated here) are additionally constrained as specified in Table 12.3.

Table 12.3—SAS notify frame header field-value restrictions

Field	Subfield	Value	Description
<i>ttl</i>	—	MAX_STATIONS	Allows SAS notify frames to propagate through all stations.
<i>baseControl</i>	<i>sc</i>	CLASS_A0	Sent as subclassA0.
	<i>fe</i>	FALSE	Not fairness eligible.
<i>da</i>	—	FF-FF-FF-FF-FF-FF	Allows SAS notify frame to be received by any station.

12.3.4.1 *type*: A one-byte field that indicates the type and length (in bytes) of the *identifier*. The *type* field is encoded as shown in Table 12.4.

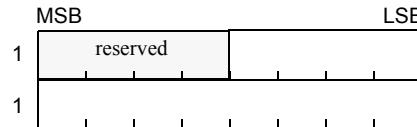
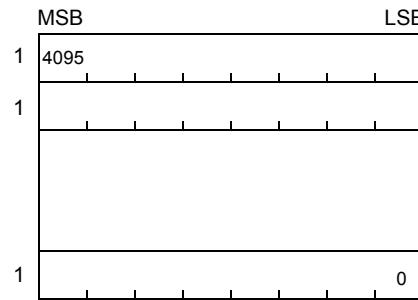
A SAS-enabled station shall be capable of performing, as a minimum, a complete purge of all entries in its SDB. An implementation with only the minimum capability shall transform any requested type to a complete purge of its SDB.

Table 12.4—*type* values

Value	Name	Length	Description
0	TYPE_ALL	0	Indicates this notify applies to all entries from the station.
1	TYPE_VID	2	Indicates this notify applies to VID entries from the station.
2	TYPE_MSTI	2	Indicates this notify applies to multiple spanning tree instance (MSTI) entries from the station.
3	TYPE_VID_MAP	512	Indicates this notify contains a 4096-bit map with bits set for the VIDs affected.
4	TYPE_MSTI_MAP	512	Indicates this notify contains a 4096-bit map with bits set for the MSTIs affected.
5-255			Reserved.

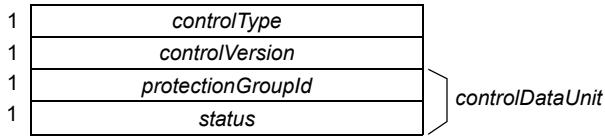
A compliant implementation shall be capable of requesting TYPE_ALL.

12.3.4.2 identifier: A field containing the value as indicated by *type*. This value is ignored for TYPE_ALL. Both TYPE_VID and TYPE_MSTI notifications use the most significant 12 bits to send a VLAN or MSTI identifier, as illustrated in Figure 12.11. The TYPE_VID_MAP and TYPE_MSTI_MAP notifications have a format illustrated in Figure 12.12.

**Figure 12.11—Identifier value format****Figure 12.12—Identifier bit map format**

12.3.5 PIRC frame

The PIRC frames are identified by the *controlType* field value of CT_OAM_PIRC_STATUS and a *controlVersion* value of 0. The PIRC frame is sent only by the inter-ring station. The PIRC frame contents include information about the sending station's status and characteristics. Figure 12.13 illustrates the PIRC frame payload with the *controlDataUnit* portion of the frame indicated.

**Figure 12.13—PIRC frame payload format**

The header fields (specified in 9.3, but not illustrated here) are additionally constrained as shown in Table 12.5.

Table 12.5—PIRC frame header field-value restrictions

Field	Subfield	Value	Description
<i>ttl</i>	—	MAX_STATIONS	Allows PIRC frames to propagate through all stations.
<i>baseControl</i>	<i>sc</i>	CLASS_A0	Sent as subclassA0.
	<i>fe</i>	FALSE	Not fairness eligible.
<i>da</i>	—	FF-FF-FF-FF-FF-FF	Allows PIRC frames to be received by any station.

12.3.5.1 Protection Group ID: A one byte field identifying a protection group.

12.3.5.1.1 Status: The status field consists of multiple subfields, as illustrated in Figure 12.14:

**Figure 12.14—PIRC frame status field**

12.3.5.1.2 *res*: A 4-bit reserved field.

12.3.5.1.3 *lb*: A (load balancing) bit that is set to 1 if the PIRC station works in load balancing mode. Otherwise, this bit has a default 0 value.

12.3.5.1.4 *pr*: A (primary protection) bit that is set to 1 if the PIRC station is provisioned as the primary protection group member. Otherwise, this bit has a default 0 value.

12.3.5.1.5 *ms*: A (manual switch) bit that is set to 1 if the PIRC station is in the PIRC protection state of PIRC_MANUAL_SWITCH. Otherwise, this bit has a default 0 value.

12.3.5.1.6 *ok*: An (interconnection capability) bit field that is set to 1 if the PIRC station is capable of interconnecting functionality. Otherwise, this bit has a default 0 value.

12.4 OAM service primitives

12.4.1 Echo MA_CONTROL.request

The client uses the MA_CONTROL.request function (see 6.4.3) to request the generation of an echo request frame. The semantics for the MA_CONTROL.request of an echo request frame are as follows:

```
MA_CONTROL.request
(
    OAM_ECHO_REQ,
    destination_address,
    service_class,
    ringlet_id,          // optional
    mac_protection,     // optional
    response_ringlet,   // optional
    user_data           // optional
)
```

The parameters of the MA_CONTROL.request of an echo request frame are described below.

OAM_ECHO_REQ

Opcode that indicates an echo request frame generation operation request.

destination_address

Specifies an individual MAC address of any station in the ring including the local station.

Sets the *da* (destination MAC address) of the transmitted echo request frame (see 12.3.1).

service_class

The class of service requested by the MAC client (see Table 6.1).

The MAC entity uses this value to

- a) Select the value of the *sc* field (see 9.6.4) of the transmitted echo request frame.
- b) Specify the MAC treatment of the transmitted frame (see 7.7.4).

ringlet_id

Indicates the ringlet choice of the client (see Table 6.2 and 7.7.1).

This value selects the *ri* (ringlet identifier) field value of the transmitted echo request frame (before any protection-based change) (see 9.6.1). If the *ringlet_id* parameter is omitted, the MAC uses its default algorithm to determine which ringlet to use.

mac_protection

Indicates a choice of whether the MAC provides protection for the echo request frame (see 7.7.1). The *mac_protection* field also sets the value of the *protectionMode* bit in the *responseControl* field (see 12.3.1.2).

TRUE—The MAC provides protection and *protectionMode* is set to 1.

FALSE—The MAC does not provide protection and *protectionMode* is cleared to 0.

(null)—The default value is TRUE.

response_ringlet

Sets the value of the *responseRinglet* field in the *responseControl* field (see Table 12.2 and 12.3.1.3).

(null)—The default value is RR_DEFAULT.

user_data

If present, *user_data* sets the value of the *userData* field in the echo request frame (see 12.3.1.4).

12.4.2 Echo MA_CONTROL.indication

The MAC uses the MA_CONTROL.indication function (see 5.4.4) to indicate to the client the reception of an echo response frame. The semantics of the MA_CONTROL.indication of an echo response frame are as follows:

```
MA_CONTROL.indication
(
    OAM_ECHO_IND,
    source_address,
    service_class,
    ringlet_id,
    protection_mode,
    response_ringlet,
    user_data
)
```

The parameters of the MA_CONTROL.indication of an echo response frame are as follows:

OAM_ECHO_IND	An <i>opcode</i> value that indicates an echo response frame arrival to the MAC sublayer.
source_address	The value of the <i>sa</i> field in the incoming echo response frame, as specified in 9.3.2.4.
service_class	The service class at which the echo response frame was sent, as specified in Table 6.1 (see also Table 7.1 of 7.3).
ringlet_id	The setting of the <i>ri</i> bit in the echo response frame header, as specified in Table 6.2 (see also 9.6.1).
protection_mode	The setting of the <i>protectionMode</i> bit in the <i>responseControl</i> field of the echo response frame (see 12.3.1.2).
response_ringlet	The setting of the <i>responseRinglet</i> field in the <i>responseControl</i> field of the echo response frame (see 12.3.1.3).
user_data	Data from the <i>userData</i> field of the incoming echo response frame (see 12.3.1.4).

12.4.3 Flush MA_CONTROL.request

The client uses the MA_CONTROL.request function (see 6.4.3) to request the generation of a flush frame. The semantics for the MA_CONTROL.request of a flush frame are as follows:

```
MA_CONTROL.request
(
    OAM_FLUSH_REQ,
    service_class,
    ringlet_id,          // optional
    mac_protection,     // optional
    user_data           // optional
)
```

The parameters of the MA_CONTROL.request of a flush frame are as follows:

OAM_FLUSH_REQ

Opcode that indicates a flush frame generation operation request.

service_class

Indicates the class of service requested by the MAC client, as specified in Table 6.1.

The service_class value selects the value of the *sc* field (see 9.6.4) of the transmitted flush frame, and to indicate the requested MAC treatment of the transmitted frame (see 7.7.4).

ringlet_id

Indicates the ringlet choice of the client, as specified in Table 6.2.

The ringlet_id value selects the value of the *ri* (ringlet identifier) field (see 9.6.1) of the transmitted flush frame, before any protection-based change (see 7.7.1). If the ringlet_id parameter is omitted, the MAC uses its default algorithm to determine which ringlet to use.

mac_protection

Indicates a choice of whether the MAC provides protection for the flush frame (see 7.7.1).

TRUE—The MAC shall provide protection for the flush frame.

FALSE—The MAC shall not provide protection for the flush frame.

(null)—The default value is TRUE.

user_data

Data for the *userData* field of the flush frame (see 12.3.2.2).

12.4.4 Flush MA_CONTROL.indication

The MAC uses the MA_CONTROL.indication function (see 5.4.4) to indicate to the client the reception of a flush frame. The semantics of the MA_CONTROL.indication of an flush frame are as follows:

MA_CONTROL.indication

(

OAM_FLUSH_IND,

service_class,

ringlet_id,

user_data

)

The parameters of the MA_CONTROL.indication of a flush frame are described below.

OAM_FLUSH_IND

An *opcode* value that indicates a flush frame arrival to the MAC sublayer.

service_class

Indicates the service class at which the flush frame was sent, as specified in Table 6.1 (see also Table 7.2 of 7.3).

ringlet_id

Indicates the setting of the *ri* bit (see 9.6.1) in the flush frame header, as specified in Table 6.2.

user_data

Data from the *userData* field of the incoming flush frame, as specified in 12.3.2.2.

12.4.5 Organization-specific MA_CONTROL.request

The client uses the MA_CONTROL.request function (see 6.4.3) to request the generation of an organization-specific frame. The semantics for the MA_CONTROL.request of an organization-specific frame are as follows:

```
MA_CONTROL.request
(
    OAM_ORG_REQ,
    destination_address,
    service_class,
    ringlet_id,          // optional
    mac_protection,     // optional
    organization,
    user_data           // optional
)
```

The parameters of the MA_CONTROL.request of an organization-specific frame are described below.

OAM_ORG_REQ

An *opcode* value that indicates an organization-specific frame generation operation request.

destination_address

Specifies an individual MAC address of any station in the ring including the local station.

Sets the *da* (destination MAC address) of the transmitted organization-specific frame (see 12.3.1).

service_class

The class of service requested by the MAC client (see Table 6.1).

The MAC entity uses this value to

a) Select the *sc* field (see 9.6.4) value in the transmitted organization-specific frame.

b) Specify the MAC treatment of the transmitted frame (see 7.7.4).

ringlet_id

Indicates the ringlet choice of the client (see Table 6.2 and 7.7.1).

This value selects the *ri* (ringlet identifier) field value of the transmitted organization-specific frame, before any protection-based change (see 9.6.1). If the ringlet_id parameter is omitted, the MAC uses its default algorithm to determine which ringlet to use.

mac_protection

Indicates whether the MAC provides protection for the organization-specific frame (see 7.7.1).

TRUE—The MAC shall provide protection for the organization-specific frame.

FALSE—The MAC shall not provide protection for the organization-specific frame.

(null)—The default value is TRUE.

organization

Sets the value of the *oui* and *dependentID* fields in the organization-specific frame (see 12.3.3.1 and 12.3.3.2).

user_data

If present, sets the value of the *userData* field in the organization-specific frame (see 12.3.3.3).

12.4.6 Organization-specific MA_CONTROL.indication

The MAC uses the MA_CONTROL.indication function (see 5.4.4) to indicate to the client the reception of an organization-specific frame. The semantics of the MA_CONTROL.indication of an organization-specific frame are as follows:

```
MA_CONTROL.indication
(
    OAM_ORG_IND,
    source_address,
    service_class,
    ringlet_id,
    organization,
    user_data
)
```

The parameters of the MA_CONTROL.indication of an organization-specific frame are described below.

OAM_ORG_IND

An *opcode* value that indicates an organization-specific frame arrival to the MAC sublayer.

source_address

The *sa* filed in the incoming organization-specific frame (see 9.3.2.4).

service_class

The service class at which the organization-specific frame was sent (see Table 6.1 and Table 7.1).

ringlet_id

The setting of the *ri* bit in the organization-specific frame header (see Table 6.2 and 9.6.1).

organization

Indicates the setting of the *oui* and *dependentID* fields in the organization-specific frame (see 12.3.3.1 and 12.3.3.2).

user_data

Data from the *userData* field of the incoming organization-specific frame (see 12.3.3.3).

12.4.7 SAS notify MA_CONTROL.request

The client uses the MA_CONTROL.request function (see 6.4.3) to request the generation of a SAS notify frame. The semantics for the MA_CONTROL.request of a SAS notify frame are as follows:

```
MA_CONTROL.request
(
    OAM_SAS_NOTIFY,
    type,           // optional
    identifier      // optional
)
```

The parameters of the MA_CONTROL.request of a SAS notify frame are described below.

OAM_SAS_NOTIFY

Opcode that indicates a SAS notify frame generation operation request.

type

Indicates the type of identifier found in the identifier parameter for the SAS notify frame.

TYPE_ALL—All SDB entries associated with the station sourcing the request are invalid.

TYPE_VID—All SDB entries associated with the station sourcing the request and the VID value contained in the *identifier* are invalid.

TYPE_MSTI—All SDB entries associated with the station sourcing the request and the MSTI value contained in the *identifier* are invalid.

TYPE_VID_MAP—All SDB entries associated with the station sourcing the request and the VIDs indicated in the *identifier* bit map are invalid.

TYPE_MSTI_MAP—All SDB entries associated with the station sourcing the request and the MSTIs indicated in the *identifier* bit map are invalid.

(null)—The default value is TYPE_ALL.

identifier

Provides the value of the *identifier* field (see 12.1.2).

(null)—The default value is 0.

12.5 OAM state machines

12.5.1 OamFrameTransmit state machine

12.5.1.1 OamFrameTransmit state machine definitions

NULL

See 12.2.4.

OAM_ECHO_REQ

OAM_FLUSH_REQ

OAM_ORG_REQ

OAM_PIRC_STATUS

OAM_SAS_NOTIFY

See 12.2.4.

Q_TX_OAM

See 12.2.4.

12.5.1.2 OamFrameTransmit state machine variables

frame

A control frame being prepared for transmissions.

12.5.1.3 OamFrameTransmit state machine routines

Dequeue(queue)

See 12.2.4.

EchoRequestFrame(frame)

Formats the remaining frame field values for an echo request frame (see 12.3.1).

Enqueue(queue,frame)

See 12.2.4.

FlushRequestFrame(frame)

Formats the remaining frame field values for an flush request frame (see 12.3.2).

OrgRequestFrame(frame)

Formats the remaining frame field values for an organization-specific request frame (see 12.3.3).

PircStatusFrame(frame)

Formats the remaining frame field values for a PIRC status frame (see 12.3.5).

SasNotifyFrame(frame)

Formats the remaining frame field values for a SAS notify frame (see 12.3.4).

12.5.1.4 OamFrameTransmit state table

The OamFrameTransmit state machine specified in Table 12.6 implements the functions necessary for OAM frame request processing by the source station. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 12.6—OamFrameTransmit state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_TX_OAM)) != NULL	1	—	FORM
	—	2	—	START
FORM	frame.controlType == OAM_ECHO_REQ	3	frame = EchoRequestFrame(frame);	SEND
	frame.controlType == OAM_FLUSH_REQ	4	frame = FlushRequestFrame(frame);	
	frame.controlType == OAM_ORG_REQ	5	frame = OrgRequestFrame(frame);	
	frame.controlType == OAM_SAS_NOTIFY	6	frame = SasNotifyFrame(frame);	
	frame.controlType == OAM_PIRC_STATUS	7	frame = PircStatusFrame(frame);	
	—	8	—	START
SEND	—	9	Enqueue(Q_TX_RS, frame);	START

Row 12.6-1: An OAM frame generation operation is requested by the MAC client.

Row 12.6-2: Wait for OAM frame generation request.

Row 12.6-3: If the client opcode indicates echo request, generate an echo request frame.

When the MAC services an echo request destined to itself, the echo request is still transmitted onto the ring.

Row 12.6-4: If client opcode indicates flush, generate a flush request frame.

Row 12.6-5: If the client opcode indicates organization-specific, generate an organization-specific frame.

Row 12.6-6: If the client opcode indicates SAS notify, generate a SAS notify frame.

Row 12.6-7: If the client opcode indicates PIRC status, generate a PIRC status frame.

Row 12.6-8: Discard unknown OAM request.

Row 12.6-9: Forward frame to ringlet selection.

12.5.2 OamFrameReceive state machine

The OamReceiving state machine implements the functions necessary for OAM frame reception, including echo response formation/transmission at the target station.

12.5.2.1 OamFrameReceive state machine definitions

CT_OAM_ECHO_RSP
Q_RX_ECHO_REQ
Q_RX_ECHO_RSP
Q_RX_FLUSH
Q_RX_ORG
Q_RX_SAS_NOTIFY
Q_TX_RS
 See 12.2.4.
RINGLET_0
RINGLET_1
 See 12.2.4.
RR_RINGLET0
RR_RINGLET1
RR_DEFAULT
 Enumerated values in an echo request, used to specify the echo response ringlet (see Table 12.2).
NULL
 See 7.2.1.

12.5.2.2 OamFrameReceive state machine variables

mac_protection
myMacAddress
 See 12.2.4.
ringlet_id
 See 12.2.4.
rxFrame
 A received control frame.
txFrame
 A control frame being prepared for transmissions.

12.5.2.3 OamFrameReceive state machine routines

Dequeue(queue)
Enqueue(queue,frame)
 See 12.2.4.
OamEchoIndication()
 Sends an indication to the client, with a specified MA_CONTROL.indication opcode value of OAM_ECHO_IND and the corresponding operands (see 6.4.4).
OamFlushIndication()
 Sends an indication to the client, with a specified MA_CONTROL.indication opcode value of OAM_FLUSH_IND and the corresponding operands (see 6.4.4).
OamOrganizationIndication()
 Sends an indication to the client, with a specified MA_CONTROL.indication opcode value of OAM_ORG_IND and the corresponding operands (see 6.4.4).
PircMateMsgReceive()
 Processes received PIRC status message from the PIRC mate (see 15.4.1).

12.5.2.4 OamFrameReceive state table

The OamFrameReceive state table is specified in Table 12.7. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 12.7—OamFrameReceive state table

Current state		Row	Next state	
state	condition		action	state
START	(rxFrame = Dequeue(Q_RX_ECHO_REQ)) != NULL	1	—	FIRST
	(rxFrame = Dequeue(Q_RX_ECHO_RSP)) != NULL	2	OamEchoIndication(rxFrame);	START
	(rxFrame = Dequeue(Q_RX_FLUSH)) != NULL	3	OamFlushIndication(rxFrame);	
	(rxFrame = Dequeue(Q_RX_ORG)) != NULL	4	OamOrganizationIndication(rxFrame);	
	(rxFrame = Dequeue(Q_RX_SAS_NOTIFY)) != NULL	5	SdbPurge(rxFrame);	
	(rxFrame = Dequeue(Q_RX_PIRC_STATUS)) != NULL	6	PircMateMsgReceive(rxFrame);	
	—	7	—	
FIRST	rxFrame.responseRinglet==RR_RINGLET0	8	ringlet_id = RI_0;	MODE
	rxFrame.responseRinglet==RR_REVERSE && rxFrame.ri == RINGLET_1	9		
	rxFrame.responseRinglet == RR_RINGLET1	10	ringlet_id = RI_1;	
	rxFrame.responseRinglet==RR_REVERSE && rxFrame.ri == RINGLET_0	11		
	—	12	ringlet_id = RI_DEFAULT;	
MODE	rxFrame.protectionMode != 0	13	mac_protection = TRUE;	FINAL
	—	14	mac_protection = FALSE;	
FINAL	—	15	txFrame = rxFrame; txFrame.controlType = CT_OAM_ECHO_RSP; txFrame.da = rxFrame.sa; txFrame.sa = myMacAddress; Enqueue(Q_TX_RS, txFrame);	START

Row 12.7-1: An echo request frame is received.

Row 12.7-2: An echo response frame sets the MA_CONTROL.indication opcode and operands.

Row 12.7-3: A flush frame sets the MA_CONTROL.indication opcode and operands.

Row 12.7-4: An organization-specific frame sets MA_CONTROL.indication opcode and operands.

Row 12.7-5: Purge selected entries found in the SDB associated with the source address of the frame.

Row 12.7-6: Process the received PIRC status frame.

Row 12.7-7: Wait for the next OAM frame.

Row 12.7-8: If the echo request indicates RR_RINGLET0, place the echo response on ringlet0.

Row 12.7-9: If a ringlet1 echo request indicates RR_REVERSE, place the echo response on ringlet0.

Row 12.7-10: If the echo request indicates RR_RINGLET1, place the echo response on ringlet1.

Row 12.7-11: If a ringlet0 echo request indicates RR_REVERSE, place the echo response on ringlet1.

Row 12.7-12: If an echo request indicates RR_DEFAULT, place the echo response on the default ringlet.

Row 12.7-13: If the echo request indicates protection, select MAC protected mode transmission.

Row 12.7-14: If the echo request indicates no protection, select MAC unprotected mode transmission.

Row 12.7-15: Generate an echo response frame, copy echo request fields, and forward to ringlet selection. When the MAC receives an echo request from itself, the echo response is still transmitted onto the ring.

The expected time between an echo request and the echo response is a function of the ring size, distance to the responding station, processing time of the responding station, and the service class chosen for the echo frame. However, the time elapsed between an echo request frame reception (a frame arrives in the queue used by Row 12.7-1) and the generation of the corresponding echo response (the frame leaves the queue used by Row 12.7-15) shall be no more than 10 milliseconds, when echo requests are received no more frequently than once every 100 milliseconds.

12.6 Performance monitoring

Stations may accumulate performance parameters related to RPR to enable the detection of developing failures before a total outage is detected.

The following subclauses define the RPR performance parameters related to error monitoring and availability. Stations may support some, all, or none of these parameters.

12.6.1 Performance monitoring counters

The performance monitoring counters, described in the following subclauses, are based on the concept of available and unavailable seconds, as adapted from ANSI T1.231-1997 6.5.4.2.8 (see 12.6.2).

12.6.1.1 *unavailableSeconds*

The *unavailableSeconds* counter is incremented once during each second for which the RPR service is unavailable (see 12.6.2).

12.6.1.2 *scffErrors*

The *scffErrors* counter (see Table 7.20) is incremented during an available second, once for each observed SCFF with either of the following properties:

- The SCFF's observed *parity* bit differs from its locally computed value.
- The SCFF's observed *fcs* field differs from its locally computed value.

12.6.1.3 *erroredSeconds*

The *erroredSeconds* counter is incremented once during an available second in which any of the following occurs:

- An SCFF's observed *parity* bit differs from its locally computed value.
- An SCFF's observed *fcs* field differs from its locally computed value.
- A miscabling defect (see 11.6.8) was present.
- A keepalive timeout (see 11.6.2) was present.
- A PHY failure (see 8.2) was present.

12.6.1.4 severelyErroredSeconds

The *severelyErroredSeconds* counter is incremented once during an available second in which any of the following occurs:

- The value of *scfErrors* increased by at least *sesThreshold* counts.
- A miscabling defect (see 11.6.8) was present.
- A keepalive timeout (see 11.6.2) was present.
- A PHY failure (see 8.2) was present.

12.6.2 Available and unavailable seconds

The update of the preceding performance monitoring counts depends on the available/unavailable classification of the seconds being measured. This classification is based on the presence or absence of severely errored second (SES) intervals, as illustrated in Figure 12.15.

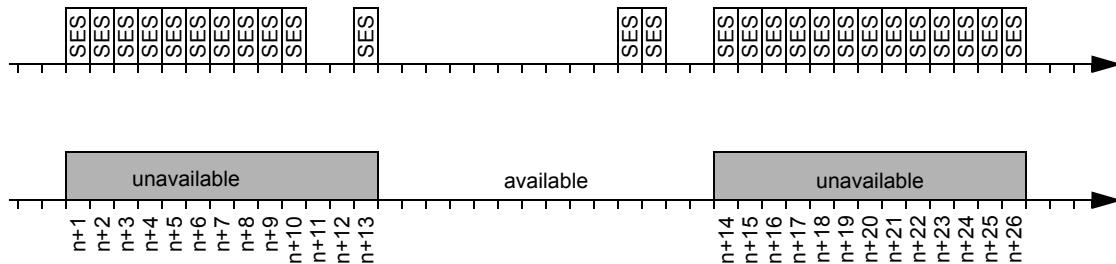


Figure 12.15—Available second classification

Once available, the RPR service transitions to unavailable at the onset of ten contiguous SESs, where the ten initial SESs are included in unavailable time. Once unavailable, the RPR service transitions to available at the onset of ten contiguous seconds, none of which are classified as an SES. The ten seconds with no SESs are included in the available time.

In Figure 12.15, for example, the value of $n+i$ represents the *unavailableSeconds* counter.

12.7 Protocol Implementation Conformance Statement (PICS) proforma for Clause 12²⁴

12.7.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 12, Operations, administration, and maintenance (OAM), shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005 as amended.

12.7.2 Identification

12.7.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

12.7.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, Operations, administration, and maintenance (OAM)
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

²⁴Copyright release for PICS proformas: Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

12.7.3 PICS tables for Clause 12

12.7.3.1 Major options

Item	Feature	Subclause	Value/Comment	Status	Support
ORG*	Organization specific frames	12.3.3	Adds organization-specific functionality.	O	Yes [] No []
PM*	Performance monitoring	12.6	accumulate performance parameters	O	Yes [] No []

12.7.3.2 State machines

Item	Feature	Subclause	Value/Comment	Status	Support
SM1	OamFrameTransmit state machine	12.5.1	OAM request frames generated in response to a MAC client service request.	M	Yes []
SM2	OamFrameReceive state machine	12.5.2	OAM response frames generate a MAC client service indication; an echo request generates an echo response.	M	Yes []

12.7.3.3 Frame formats

Item	Feature	Subclause	Value/Comment	Status	Support
FF1	Echo frame formats	12.3.1	Echo request/response formats	M	Yes []
FF2	Flush frame format	12.3.2	Flush frame format	M	Yes []
FF3	Organization-specific frame format	12.3.2	Organization-specific frame format	ORG:M	Yes []

12.7.3.4 Service primitives

Item	Feature	Subclause	Value/Comment	Status	Support
SP1	MA_CONTROL.request with an opcode value of OAM_ECHO_REQ	12.4.1	Echo request/response formats	M	Yes []
SP2	MA_CONTROL.indication with an opcode value of OAM_ECHO_IND	12.4.2	Echo request/response formats	M	Yes []
SP3	MA_CONTROL.request with an opcode value of OAM_FLUSH_REQ	12.4.3	Flush formats	M	Yes []

SP4	MA_CONTROL.indication with an opcode value of OAM_FLUSH_IND	12.4.4	Flush formats	M	Yes []
SP5	MA_CONTROL.request with an opcode value of OAM_ORG_REQ	12.4.5	Organization-specific formats	ORG:M	Yes []
SP6	MA_CONTROL.indication with an opcode value of OAM_ORG_IND	12.5.2	Organization-specific formats	ORG:M	Yes []

12.7.3.5 Performance monitoring

Item	Feature	Subclause	Value/Comment	Status	Support
PM1	<i>scffErrors</i>	12.6.1.2	counts errors	PM:O	Yes [] No []
PM2	<i>erroredSeconds</i>	12.6.1.3	counts seconds	PM:O	Yes [] No []
PM3	<i>severelyErroredSeconds</i>	12.6.1.4	counts seconds	PM:O	Yes [] No []
PM4	<i>unavailableTime</i>	12.6.1.1	counts 1 second intervals	PM:O	Yes [] No []
PM5	<i>scffErrors,</i> <i>erroredSeconds,</i> <i>severelyErroredSeconds</i>	12.6.1.2 12.6.1.3 12.6.1.4	accumulated during available time only	PM:M	Yes []

12.7.3.6 SAS notify

Item	Feature	Subclause	Value/Comment	Status	Support
SS1*	SAS support	14.1	Spatially aware sublayer supported	O	Yes [] No []
SS2	Capable of requesting SAS notify frame type of value TYPE_ALL	12.3.4.1	SAS notify frame type TYPE_ALL	SS1:M	Yes []
SS3	Capable of requesting SAS notify frame type of value TYPE_VID	12.3.4.1	SAS notify frame type TYPE_VID	SS1:O	Yes [] No []
SS4	Capable of requesting SAS notify frame type of value TYPE_MSTI	12.3.4.1	SAS notify frame type TYPE_MSTI	SS1:O	Yes [] No []
SS5	Capable of requesting SAS notify frame type of value TYPE_VID_MAP	12.3.4.1	SAS notify frame type TYPE_VID_MAP	SS1:O	Yes [] No []

SS6	Capable of requesting SAS notify frame type of value TYPE_MSTI_MAP	12.3.4.1	SAS notify frame type TYPE_MSTI_MAP	SS1:O	Yes [] No []
SS7	A station with minimal capabilities is able to transform any requested type to a complete purge	12.3.4.1	Transformation of any requested type to a complete purge	SS1:M	Yes []

12.7.3.7 PIRC

Item	Feature	Subclause	Value/Comment	Status	Support
PSI	PIRC support	15.1	Protected inter-ring connection sublayer supported	O	Yes [] No []

13. Layer management entity interface

13.1 Overview of the management model

The RPR MAC conceptually includes a management entity, called the MAC layer management entity (MLME). This entity provides the layer management service interfaces through which layer management functions may be invoked.

In order to provide correct RPR MAC operation, a station management entity (SME) shall be present. The SME is a layer-independent entity that may be viewed as residing in a separate management plane. The exact functions of the SME are not specified in this standard, but in general this entity may be viewed as being responsible for such functions as the gathering of layer-dependent status from the various layer management entities, and similarly setting the value of layer-specific parameters. The SME typically performs such functions on behalf of general system management entities and implements standard management protocols. Figure 13.1 depicts the relationship among management entities.

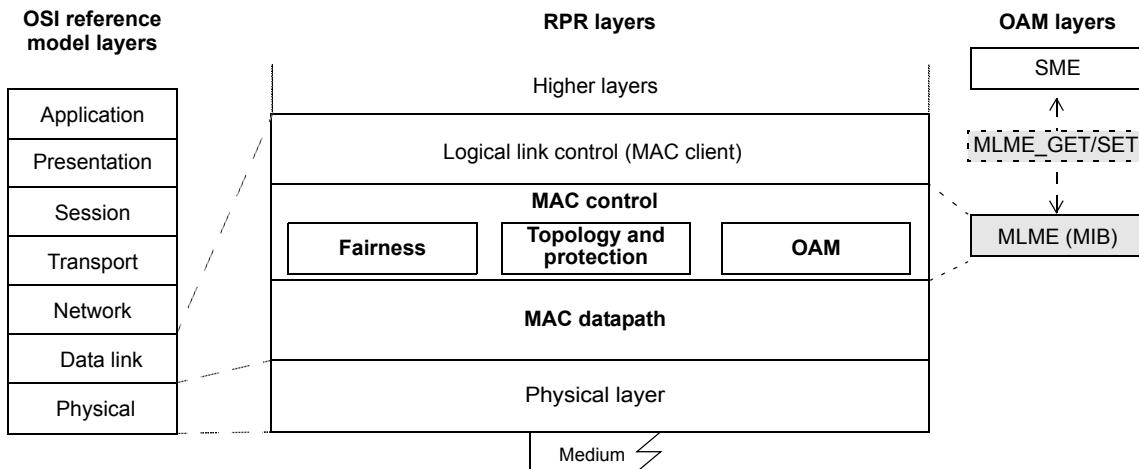


Figure 13.1—Management entities relationship to the ISO/IEC OSI reference model

The management service interface within this model is the SME-MLME service interface.

The protection management mechanisms defined in this standard are based on information known to the MAC entity. PHY layer protection mechanisms are independent and are addressed in the appropriate PHY specifications.

The interfaces of the SME with the different PHYs are not part of this standard and are specified in the respective standards documents that specify the management primitives and MIBs for the different PHYs.

13.2 MLME service interface

The management information specific to each layer is represented as a management information base (MIB) for that layer. The MAC and PHY layer management entities are viewed as each implementing the MIB for that layer. The generic model of MIB-related management primitives exchanged across the management service interfaces is to allow the service interface user-entity to either GET the value of a MIB attribute, to SET the value of a MIB attribute, or to be notified about asynchronous event.

The following five primitives are defined for the MAC layer management:

- MLME_GET.request
- MLME_SET.request
- MLME_EVENT.indication
- MLME_RESET.request

The primitives MLME_GET.request, MLME_SET.request, and MLME_EVENT.indication described in this subclause shall be supported.

The primitive MLME_RESET.request may be supported. This service is used to initialize the management entities, the MIBs, and the datapath entities. It may include a list of attributes for items to be initialized to default values.

13.2.1 MLME_GET.request

13.2.1.1 Function

MLME_GET.request is used to request the value of a MIB attribute.

13.2.1.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
MLME_GET.request
(
  mib_attribute,
  mib_attributevalue,
  status
)
```

The parameters of the MLME_GET.request primitive are as follows:

mib_attribute	A readable attribute in the RPR MIB as defined in Annex D.
mib_attributevalue	The value of the mib_attribute returned by the MLME_GET.request.
status	The status of the request returned as a result of the MLME_GET.request.
	SUCCESS—The request was successful.
	FAIL—(Otherwise.)

13.2.1.3 When generated

This primitive is generated by the SME whenever it wishes to retrieve attributes from the RPR MIB.

13.2.1.4 Effect of receipt

This primitive returns the appropriate MIB attribute value if the status equals “success”; otherwise, it returns an error indication.

13.2.2 MLME_SET.request

13.2.2.1 Function

MLME_SET.request is used to set a new value to a MIB attribute.

13.2.2.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
MLME_SET.request
(
    mib_attribute,
    mib_attributevalue,
    status
)
```

The parameters of the MLME_SET.request primitive are as follows:

```
mib_attribute
    A writable attribute in the RPR MIB as defined in Annex D.
mib_attributevalue
    The value of the mib_attribute.
status
    The status of the request returned as a result of the MLME_GET.request.
        SUCCESS—The request was successful.
        FAIL—(Otherwise.)
```

13.2.2.3 When generated

This primitive is generated by the SME whenever it wishes to set attributes in the RPR MIB.

13.2.2.4 Effect of receipt

This primitive returns a status equal to “success” if the set succeeds; otherwise, it returns an error indication.

13.2.3 MLME_EVENT.indication

13.2.3.1 Function

MLME_EVENT.indication is used by the MLME to asynchronously notify the SME of state changes or events.

13.2.3.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
MLME_EVENT.indication
(
    event
)
```

The parameters of the MLME_EVENT.indication are as follows:

```
event
    An RPR event as described in 13.2.3.3 and Table 13.1.
```

Table 13.1—Event indications

Meaning	Critical severity	Specified in
Miscabling	Yes	11.2.9
Topology instability	Yes	11.2.9
Topology inconsistency	Yes	11.2.9
Maximum number of stations exceeded	Yes	11.2.9
Protection configuration mismatch	Yes	11.2.9
LRTT measurement incomplete	Yes	11.2.9
Invalid topology entry	Yes	11.2.9
Duplicate MAC address	Yes	11.2.9
Excess reserved bandwidth	No	11.2.9
Change of RPR interface operational state	No	13.3.1.2
RPR keepalive timeout	No	11.6
Reception of echo reply message	No	12.4.1
Reception of returned flush frame	No	12.1.5

13.2.3.3 When generated

This primitive is generated by the MLME whenever it wishes to notify the SME about an asynchronous event, such as the following:

- a) Any item in 11.2.9.
- b) Change of RPR interface operational state (13.3.1.2).
- c) RPR keepalive timeout (11.6).
- d) Reception of echo reply message (12.4.1).
- e) Reception of returned flush frame (12.1.5).

13.2.3.4 Effect of receipt

This primitive indicates the event that caused the asynchronous MLME notification.

13.2.4 MLME_RESET.request**13.2.4.1 Function**

MLME-RESET.request is used to initialize the management entities, the MIB attributes, and the datapath entities.

13.2.4.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
MLME_RESET.request
(
    list of mib_attributes,          // optional
    status
)
```

The parameters of the MLME_RESET.request primitive are as follows:

list of mib_attributes

A set of writable attributes in the RPR MIB as defined in Annex D. The list of *mib_attributes* may be specified. If specified, the attributes in the list are initialized to default values. If not specified, all MIB attributes are initialized to default values.

status

The status of the request returned as a result of the MLME_RESET.request.

SUCCESS—The request was successful.

FAIL—(Otherwise.)

13.2.4.3 When generated

This primitive is generated by the SME whenever it wishes to reset MIB attributes to their default values.

13.2.4.4 Effect of receipt

This primitive returns a status equal to “success” if the reset succeeds; otherwise, it returns an error indication.

13.3 MLME services

The services provided by the MLME to the SME are specified in this section. These services are described in an abstract way and do not imply any particular implementation or exposed interface. MLME service interface primitives are of the general form ACTION.request. The SME uses the services provided by the MLME through the MLME service interface.

13.3.1 RPR interface configuration

13.3.1.1 Administrative status configuration

The RPR interface has its own administrative state that specifies the desired state of the interface. The administrative state allows or forbids the upper and lower layers to send frames to and from the ring.

When the RPR interface administrative status (ifAdminStatus) is set to down (as defined in RFC 2863), the station shall implement this using at least one of the following means:

- a) Station goes into passthrough.
- b) Station disables transit paths and add/drop client interfaces, and stops generating or responding to any topology/protection/fairness messages.
- c) Station disables both span interfaces.

13.3.1.2 Operational status reporting

The RPR interface operational state (RFC 2863) is determined as follows:

- a) Down if either of the following conditions is true:
 - 1) RPR interface administrative state (RFC 2863) is down.
 - 2) Both spans are edges.
- b) Otherwise up.

13.3.2 Topology discovery monitoring

The configuration management shall allow monitoring the state of the autoconfiguration and topology discovery protocols for maintenance purposes.

The detailed configuration requirements for the autoconfiguration and topology discovery protocols depend upon the actual mechanism that will be used and are now for further study.

NOTE—If some misconfiguration condition is detected, a notification is sent to the management system for maintenance purposes. For example, the topology discovery mechanism can discover that two stations on the ring have the same MAC address.

13.3.3 Performance and accounting measurements

Performance and accounting measurements shall be supported involving counting selected frame flows through client to MAC, MAC to client, PHY to MAC, or MAC to PHY locations, illustrated by the ellipses within Figure 13.2. Counter names are based on the measuring location and other frame-content-dependent information.

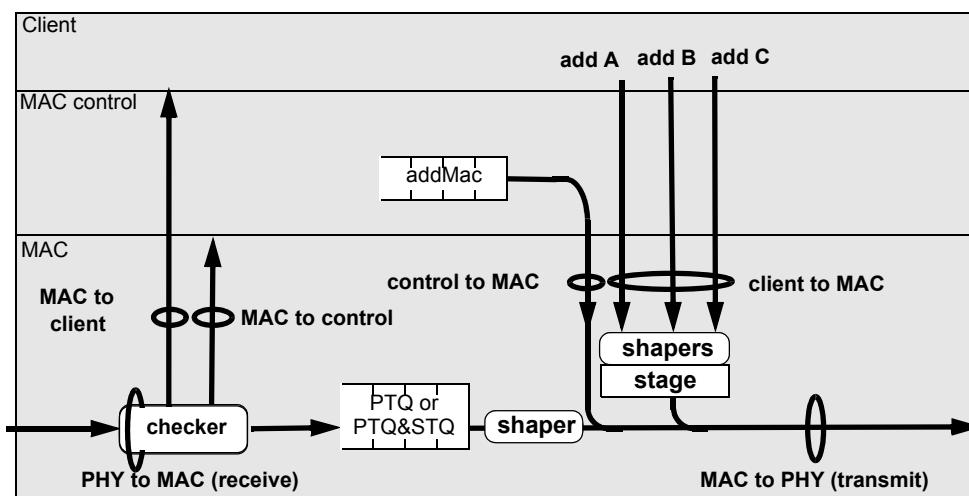


Figure 13.2—Counter-measurement points

The RPR statistics that are kept depend on the properties of the processed frame. The general format of a counter name is <place><period><dir><addr><class><units>, allowing the collection of names to be summarized in Table 13.2 and Table 13.3.

Table 13.2—Statistics definitions

Type	Variable	Values	Description
Location	<place>	Span, Client, MAC Control	Places where statistics are collected
Reporting period	<period>	Current interval, Past intervals, Total intervals, Running counters	Time periods for which statistics are reported.
Address type	<addr>	Unicast, Multicast+Broadcast, Broadcast	Frame types sent/received
Class	<class>	ClassA, ClassB-CIR, ClassB-EIR, ClassC	Service class
Units	<units>	Octets, Frames	What unit is being counted
Direction	<dir>	In, Out	Which direction from the point of view of the “higher layer”
Error type	<err>	TooLong, TooShort, BadFcs, BadHec, TtlExpired, SelfSourced, PmdAborted	Error type

Table 13.3—MAC statistics

Group	MAC counter instances	MAC counter names
Mac client frames	Reporting period Direction Address type Class / subclass	rprClient<period><dir><addr><class>Frames e.g., rprClientCurrentInUcastClassAFrames rprClientIntervalOutMcastClassCFrames
Mac client bytes	Reporting period Direction Address type Class / subclass	rprClient<period><dir><addr><class>Octets e.g., rprClientCurrentInUcastClassAOctets rprClientIntervalOutMcastClassCOctets
Span frames	Reporting period Direction Address type Class / subclass	rprSpan<period><dir><addr><class>Frames e.g., rprSpanCurrentInUcastClassAFrames rprSpanIntervalOutMcastClassCFrames
Span bytes	Reporting period Direction Address type Class / subclass	rprSpan<period><dir><addr><class>Octets e.g., rprSpanCurrentInUcastClassAOctets rprSpanIntervalOutMcastClassCOctets
Span errors	Reporting period Error type	rprSpan<period>Error<err> e.g., rprSpanCurrentErrorTtlExpFrames rprSpanIntervalErrorBadHecFrames

13.3.3.1 Interface speed

The interface speed (represented by the MIB attribute ifSpeed) is defined as the add/drop data rate of the RPR interface, which is the sum of the span interface speeds.

13.3.3.2 Intervals measurement

All performance parameters shall be accumulated in 15-minute intervals, and up to 96 intervals (24 hours worth) are kept by an agent. Fewer than 96 intervals of data will be available if the agent has been restarted within the last 24 hours. In addition, there is a rolling 24-hour total of each performance parameter. Performance parameters continue to be collected when the interface is down.

The agent implements the requirements of GR-820 and aligns with the wall clock. The start of every four 15-minute periods is aligned with the hour boundary, i.e., the 15-minute periods start on the hours, 15 minutes past the hour, 30 minutes past the hour, and 45 minutes past the hour.

Performance parameters are of types HCPerfCurrentCount, HCPerfIntervalCount, and HCPerfTotalCount as defined in RFC 3705. These textual conventions are based on Counter64 and are used to permit the resetting of the values as the intervals roll over.

13.4 Protocol Implementation Conformance Statement (PICS) proforma for Clause 13²⁵

13.4.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 13, Layer management entity interface, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

13.4.2 Identification

13.4.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

13.4.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, Layer management entity interface
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

²⁵Copyright release for PICS proformas: Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

13.4.3 PICS tables for Clause 13

13.4.3.1 SME functions

Item	Feature	Subclause	Value/Comment	Status	Support
SME1	SME	13.1	present	M	Yes []

13.4.3.2 LME functions

Item	Feature	Subclause	Value/Comment	Status	Support
LME1	LME primitives	13.2	MLME_GET.request, MLME_SET.request and MLME_EVENT.indication	M	Yes []
LME2*	LME initialization primitives	13.2	MLME_RESET.request	O	Yes [] No []
LME3	LME initialization attributes	13.2 13.2.4.2	initialize default values for attributes	LME2 :O	Yes [] No []
LME4	Administrative status down	13.3.1.1	station in passthrough	O.1	Yes [] No []
LME5	Administrative status down	13.3.1.1	station disables transit paths	O.1	Yes [] No []
LME6	Administrative status down	13.3.1.1	station disables spans	O.1	Yes [] No []
LME7	Topology discovery	13.3.2	monitor state in configuration management	M	Yes []
LME8	Performance and accounting measurements	13.3.3	counters supported as in Table 13.2 and Table 13.3	M	Yes []
LME9	Intervals	13.3.3.1	parameters accumulated in 15-min intervals and up to 96 intervals	M	Yes []

14. Spatially aware sublayer

14.1 Spatially aware sublayer overview

The spatially aware sublayer (SAS) is an optional sublayer of the MAC that may be implemented. SAS increases unicast frame spatial reuse in the ring by using directed transmissions for frames to/from remote addresses (i.e., to/from stations connected indirectly via bridges on the ring). SAS also increases multicast spatial reuse by using multicast scoping to limit multicast transmissions to less than the full ring. Without SAS, multicast and non-ring-local frame transmissions treat the ring as a broadcast medium and are flooded onto the ring, precluding spatial reuse. As illustrated in Figure 14.1, frame transmissions from station S10 to station S0 will be flooded over the ring (via an undirected transmission).

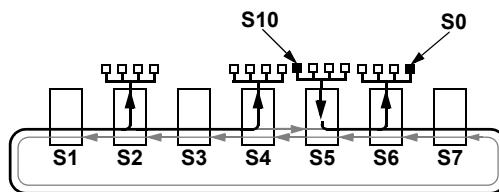


Figure 14.1—Undirected transmission

With SAS, remote unicast frames use directed transmission to achieve spatial reuse. As illustrated in Figure 14.2, once SAS database entries are learned, frame transmissions from station S10 to station S0 are sent as unicast RPR frames (via directed transmission).

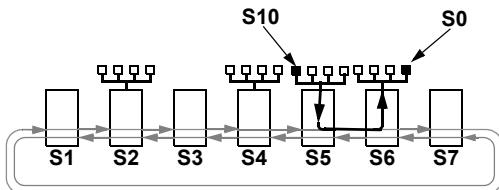


Figure 14.2—Directed transmission

SAS for unicast transmissions associates a remote address and optional accompanying VLAN identifier (VID) with a ring local address that provides the current path toward the remote address. Having a SAS association allows frames being sent to a remote address to use directed transmissions instead of being flooded. This also supports a VLAN-only entry, where all traffic with the same VID (including multicast) is forwarded to the ring local address.

SAS optionally provides spatial reuse for multicast frames by multicast scoping. This allows for ringlet and hop count to be configured per group address (and optionally VLAN identifier) to limit the flooding scope of these transmissions.

SAS comprises:

- An association database, which is referred to as the SAS association database (SDB).
- Updating the SDB as a result of changes in the topology of the ring or bridged network.
- Modification of client frames to be transmitted on the ring.
- Optional support of scoping of multicast transmissions.

- e) Interworking with RPR stations not using SAS.

14.2 Terminology and variables

14.2.1 Common definitions

This clause defines the following terms, which are used within this clause:

FID

As defined by IEEE Std 802.1Q-2005, a FID is a filtering identifier that denotes a grouping of VLAN identifiers.

MSTI

An MSTI is one of a number of spanning trees calculated by Multiple Spanning Tree Algorithm and Protocol (MSTP) within a Multiple Spanning Tree (MST) region (see IEEE 802.1Q, 3.19).

NULL VID

A VID of value 0.

target address

The ring local address associated with an end station MAC address and VID (if present).

VID

As defined by IEEE Std 802.1Q-2005, a VID is a VLAN identifier. When SAS is VLAN aware, it checks the first protocol type (contained in the protocolType field) to see whether it contains the expected type value to determine whether the next field is a VLAN identifier. If so, the VID from that tag is used by SAS.

14.2.2 Common state machine variables

optionSasTestReachability

A boolean variable determining whether reachability of the target station is tested before sending a directed transmission onto the ring. If the reachability test fails, the frame is sent using undirected transmission. The default value is TRUE. The method of configuring *optionSasTestReachability* is not standardized.

TRUE—Indicates that the target station reachability is tested before sending directed transmissions onto the ring.

FALSE—(Otherwise.)

optionSasVlanAware

A boolean variable determining whether SAS will consider VLAN identifiers from frames when making forwarding decisions. The default value is TRUE. The method of configuring *optionSasVlanAware* is not standardized.

TRUE—Indicates that SAS will consider VLAN identifiers when making forwarding decisions. If a VLAN identifier is not present in the frame, then a default VLAN identifier is used for forwarding decisions.

FALSE—Indicates that SAS will not consider VLAN identifiers when making forwarding decisions. If a VLAN identifier is present in the frame, then the VLAN identifier is ignored.

purgeScope

A variable denoting the scope used to purge the SDB.

ALL—Indicates that all entries in the SDB should be purged.

STATION—Indicates that all entries associated with the station address conveyed by the value of the *purgeValue* should be purged.

VID—Indicates that all entries associated with the VID conveyed by the value of the *purgeValue* should be purged.

MSTI—Indicates that all entries associated with the MSTI conveyed by the value of the *purgeValue* should be purged.

VID_MAP—Indicates that all entries associated with the FID(s) mapped from the VID(s) conveyed by the value of the *purgeValue* should be purged.

MSTI_MAP—Indicates that all entries associated with the MSTI(s) conveyed by the value of the *purgeValue* should be purged.

purgeValue

A variable denoting the value used to match entries in the SDB that should be purged (if applicable).

14.2.3 Common state machine routines

No common routines are defined for use by the state machines in this clause.

14.2.4 Variables and literals defined in other clauses

This clause references the following parameters, literals, variables, and routines defined in Clause 6:

remapRxAddrs

request_sas

This clause references the following parameters, literals, variables, and routines defined in Clause 7:

Dequeue(queue)
destination_address
destination_address_extended
enableSas
Enqueue(queue, frame)
flooding_form
FLOOD_NONE
Multicast(macAddress)
myMacAddress
NULL
Provided(parameter)
Q_RX_SAS
Q_TX_RS
Q_TX_SAS
Reachable(address, ringlet)
RI_0
RI_1
SAS_GROUP_ADDRESS
source_address
source_address_extended
Unicast(address)

This clause references the following parameters, literals, variables, and routines defined in Clause 11:

SdbPurgeUnreachable()

This clause references the following parameters, literals, variables, and routines defined in Clause 12:

SdbPurge(frame)

14.2.5 SAS database fields

The SAS database (see 14.5) is divided into *sdbStationInfo*, *sdbUcastEntry*, *sdbMcastEntry*, *sdbStaticUcast*, and *sdbStaticMcast* portions.

When the SAS database is configured to support VID only switching, the entry found in the *sdbUcastEntry* table applies to both unicast and multicast client traffic.

14.2.5.1 sdbStationInfo fields

The *sdbStationInfo* portion of the SAS database (see 14.5) includes the following field values:

numDynamicEntries

The number of dynamic (i.e., learned) entries in the station's SDB.

numStaticUcastEntries

The number of static unicast entries in the station's SDB.

numStaticMcastEntries

The number of static multicast entries in the station's SDB.

sdbAgingTimer

The aging duration (10 seconds–1 000 000 seconds) of dynamic entries. The default value is 300 seconds.

14.2.5.2 sdbUcastEntry fields

The *sdbUcastEntry* portion of the SAS database (see 14.5) includes the following field values:

sdbEntryAge

The age for this entry.

sdbEntryFid

The filtering database identifier (0–4094) for this entry.

sdbEntryDestAddr

The destination address used for this entry. If *sdbEntryType* is MGMT, and the *sdbStaticUcastType* is VID, then the value of this field is FF-FF-FF-FF-FF-FF.

sdbEntryTargetAddr

The target address (i.e., a ring local address) for this FID and destination address.

sdbEntryType

The type of this entry.

LEARNED—Dynamically learned.

MGMT—Represents one of the entries in *sdbStaticUcast*.

14.2.5.3 sdbMcastEntry fields

The *sdbMcastEntry* portion of the SAS database (see 14.5) includes the following field values:

sdbEntryFid

The filtering database identifier (0–4094) for this entry.

sdbEntryDestAddr

The destination address used for this entry.

sdbStaticMcastHopsRinglet0

The hop count to use when sending frames on ringlet0 to this VID/address. A value of 0 means that the frame is not sent on ringlet0.

sdbStaticMcastHopsRinglet1

The hop count to use when sending frames on ringlet1 to this VID/address. A value of 0 means that the frame is not sent on ringlet1.

sdbStaticMcastExtend

Determines whether multicast frames being sent to this VID/address are always transmitted as extended frames, with the outer destination address set to SAS_GROUP_ADDRESS, causing them to be learned by SDB during reception.

TRUE—Multicast frames are always sent as extended frames, with SAS_GROUP_ADDRESS as the destination address.

FALSE—(otherwise).

14.2.5.4 sdbStaticUcast fields

The *sdbStaticUcast* portion of the SAS database (see 14.5) includes the following field values:

sdbStaticUcastVid

The VLAN identifier (0–4095) for this entry. Untagged or priority tagged frames are matched by an entry with a value of 0.

sdbStaticUcastDestAddr

The destination address used for this entry. If the value of *sdbStaticUcastType* is VID, then this value is set to the broadcast address.

sdbStaticUcastTargetAddr

The target address (i.e., ring local address) for this VID and destination address.

sdbStaticUcastType

The type of this entry.

VID—Only match on the VID value. This applies to both unicast and multicast frames.

VID_ADDR—Match on *sdbStaticUcastVid* and *staticUcastDestAddr*.

sdbStaticUcastState

The state of this entry.

ENABLED—This entry is active, and there is a corresponding entry in the SDB unicast lookup table.

DISABLED—(otherwise).

14.2.5.5 sdbStaticMcast fields

The optional *sdbStaticMcast* portion of that database includes the following field values:

sdbStaticMcastVid

The VLAN identifier (0–4095) for this entry. Untagged or priority tagged frames are matched by an entry with a value of 0.

sdbStaticMcastAddr

The destination group address used for this entry.

sdbStaticMcastHopsRinglet0

The hop count to use when sending frames on ringlet0 to this VID/address. A value of 0 means that the frame is not sent on ringlet0.

sdbStaticMcastHopsRinglet1

The hop count to use when sending frames on ringlet1 to this VID/address. A value of 0 means that the frame is not sent on ringlet1.

sdbStaticMcastState

The state of this entry.

ENABLED—This entry is active and is usable by the datapath.

DISABLED—(otherwise).

sdbStaticMcastExtend

Determines whether multicast frames being sent to this VID/address are always transmitted as extended frames, with the outer destination address set to SAS_GROUP_ADDRESS, causing them to be learned by SDB during reception.

TRUE—Multicast frames are always sent as extended frames, with SAS_GROUP_ADDRESS as the destination address.
 FALSE—(otherwise).

14.2.6 SAS statistics counters

sasTxDirectedFrames
sasTxMcastScopedFrames
sasTxUndirectedFrames
 Counters for SAS transmission types.
sdbLearnFulls
 Counter of the number of times the *SdbLearn()* function could not add a new entry due to SDB occupancy and either overwrote an existing entry or did not learn the new information.
sdbLocalPurges
sdbNotifyPurges
sdbTopologyPurges
 Counters for the number of SDB purges.

14.3 SAS association database

The SAS association database (SDB) is used by the SasTransmit state machine to look up target addresses for client add frames. The lookup is based on the destination MAC address and VID (if VLAN-aware). It contains association information in the form of association entries that are either:

- a) Static, and explicitly configured by management action; or
- b) Dynamic, and automatically entered into the association database.

A FID is used to index the SDB for forwarding behavior. From a management perspective, a VID is used to index static entries, whereas a FID is used to index dynamic entries.

For SAS to run VLAN-aware, it needs access to the client's VID-to-FID mapping tables. The means of access is not specified.

14.3.1 Logical representation of SAS tables

The SAS tables are partitioned into multiple components, as listed below.

- a) SDB unicast lookup table (see Table 14.1): *sdbUcastEntry*. This table provides an association between the end station MAC/FID and the target address to be used in the *da* field. If no end station MAC address entry is found, then SAS will use the SAS_GROUP_ADDRESS value in the *da* field. See 14.5.3.1.4, for a description of the SasTransmit state machine.
- b) SDB static multicast lookup table (see Table 14.2): *sdbMcastEntry*. This table provides an association between the multicast (group) address and the ringlet0 and ringlet1 hop count values to be used by the multicast transmissions.
- c) Static unicast management table (see Table 14.3): *sdbStaticUcast*. This table contains static unicast information provided by management. Enabling an entry in this table creates an equivalent entry in the SDB unicast lookup table (with VID mapped to FID). See 14.3.2 for a description of the static unicast entries.
- d) Static multicast management table (see Table 14.4) *sdbStaticMcast*. This table contains static multicast information provided by management. Enabling an entry in this table creates an equivalent entry in the SDB multicast lookup table (with VID mapped to FID). See 14.3.3 for a description of the static multicast entries.

Table 14.1—Example SDB for unicast lookup

sdbEntryFid	sdbEntryDestAddr	sdbEntryTargetAddr	sdbEntryType
12	00-10-A4-97-A8-00	00-10-A4-97-A0-01	LEARNED
300	00-10-A4-97-A8-01	00-10-A4-97-A0-02	MGMT
300	00-10-A4-97-A8-02	00-10-A4-97-A0-03	MGMT
3001	00-10-A4-97-A8-03	00-10-A4-97-A0-04	LEARNED
3001	00-10-A4-97-A8-04	00-10-A4-97-A0-05	LEARNED
99	FF-FF-FF-FF-FF-FF	00-10-A4-97-A0-06	MGMT

Table 14.2—Example SDB for multicast lookup

sdbEntry FID	sdbEntryDestAddr	sdbStatic McastHops Ringlet0	sdbStatic McastHops Ringlet1	sdbStaticMcast Extend
12	AC-10-A4-97-A8-00	1	1	TRUE
300	AC-10-A4-97-A8-01	0	3	TRUE
300	AC-10-A4-97-A8-02	3	2	TRUE
3001	AC-10-A4-97-A8-03	4	2	TRUE
3001	AC-10-A4-97-A8-04	3	0	TRUE

Table 14.3—Example static unicast management table

sdbStatic UcastVID	sdbStaticUcastDest Addr	sdbStaticUcastTarget Addr	sdbStaticUcast Type	sdbStaticUcast State
101	00-10-A4-97-B8-00	00-10-A4-97-B0-01	VID_ADDR	DISABLED
250	00-10-A4-97-A8-01	00-10-A4-97-A0-02	VID_ADDR	ENABLED
251	00-10-A4-97-A8-02	00-10-A4-97-A0-03	VID_ADDR	ENABLED
102	00-10-A4-97-B8-03	00-10-A4-97-B0-04	VID_ADDR	DISABLED
99	FF-FF-FF-FF-FF-FF	00-10-A4-97-B0-05	VID	ENABLED
66	FF-FF-FF-FF-FF-FF	00-10-A4-97-B0-06	VID	DISABLED

Table 14.4—Example static multicast management table

sdbStaticMca stVID	sdbStaticMca stAddr	sdbStatic Mca stHo psRinglet0	sdbStatic Mca stHo psRinglet1	sdbStatic Mca stState	sdbStatic Mca stExt end
101	AC-10-A4-97-A8-00	1	1	ENABLED	TRUE
250	AC-10-A4-97-A8-01	0	3	ENABLED	TRUE
251	AC-10-A4-97-A8-02	3	2	ENABLED	TRUE
102	AC-10-A4-97-A8-03	4	2	ENABLED	TRUE

14.3.2 Static unicast entries

A static unicast management entry (see 14.2.5.4) specifies the following:

- a) The type of entry. Specifies whether the static entry should only use the VID or the VID plus the end station MAC address to make its forwarding decision.
- b) The VID of the entry. A value of 0 matches untagged or priority tagged frames when SAS is running VLAN-aware, or all frames when SAS is running VLAN-unaware.
- c) An end station (individual) MAC address if the entry type indicates VID plus end station MAC address. Otherwise, this address is set to the broadcast address.
- d) The target address for this entry.
- e) The state that indicates whether the static entry is used for forwarding (i.e., logically, copied into the SDB unicast lookup table).

14.3.3 Static multicast entries

A static multicast management entry (see 14.2.5.5) specifies the following:

- a) The VID of the entry. A value of 0 matches untagged or priority tagged frames when SAS is running VLAN-aware, or all frames when SAS is running VLAN-unaware.
- b) A multicast address.
- c) A multicast transmission scope, comprising:
 - 1) Hops to send on ringlet0.
 - 2) Hops to send on ringlet1.
- d) The state that indicates whether the static entry is used for forwarding (i.e., logically, copied into the SDB multicast lookup table).
- e) An indication that frames matching this entry should be transmitted with the SAS_GROUP_ADDRESS.

14.3.4 Dynamic entries

A dynamic entry (see 14.2.5.2) specifies the following:

- a) An individual MAC address;
- b) The FID for this entry;
- c) Target address for this entry;
- d) The type is LEARNED;

- e) The age for this entry.

A FID identifies a set of VLANs among which shared VLAN learning takes place. Any pair of FIDs identifies two sets of VLANs between which independent VLAN learning takes place.

Dynamic entries are created and updated by the SAS learning process (14.4.1). They shall be automatically removed after a specified time (i.e., aging time) has elapsed since the entry was created or last updated, as shown in Table 14.5. No more than one dynamic entry shall be created in the association database for a given combination of MAC address and FID.

Dynamic entries cannot be created or updated by management.

The aging out of dynamic entries ensures that end stations that have been moved to a different part of the bridged network will not be permanently prevented from receiving frames. It also takes account of changes in the active topology of the bridged network that can cause end stations to appear to move, from the point of view of the bridge.

If the value of aging time can be set by management, SAS should use values in the range specified, with a granularity of 1 second.

Table 14.5—Aging time parameter value

Parameter	Recommended default value	Range
Aging time	300 seconds	10–1 000 000 seconds

14.3.5 Mapping of VIDs to FIDs

The mapping of VIDs to FIDs within SAS determines which VLANs share a common association database (SDB). The VID to FID mapping table is provided by either the bridge client or station management.

A VLAN-aware SAS shall support between 1 and 4095 FIDs. For the purposes of the management operations, FIDs are numbered from 0 through N-1, where N is the maximum number of FIDs supported by the implementation.

14.4 SDB operations

14.4.1 SAS learning process

The SAS learning process is responsible for creating and updating SDB entries. It supports the following VLAN learning scenarios:

- a) VLAN-aware
 - Untagged traffic
 - Tagged traffic
- b) VLAN-unaware

If the SDB is full, then a new entry (i.e., an FID/address pair that does not appear in the SDB) may replace an existing one. The method of replacement is not standardized. Replacing an entry (rather than discarding new information), reduces flooding when the bridged network is changing rapidly.

14.4.1.1 VLAN-aware

SAS identifies frames to pass to the learning process by looking for extended frames with one of:

- a) $frame.da$ equal to SAS_GROUP_ADDRESS
- b) Unicast $frame.da$ with the flooding form set to FI_NONE

SAS reads the source ring local address (the $frame.sa$ field), the source end station MAC address (the $frame.saExtended$ field), and, if present, the VID extracted from the first 4 bytes of the *serviceDataUnit* from the received frame. Untagged or priority tagged frames use the NULL VID. This information is used to create or update dynamic entries in the SDB.

14.4.1.2 VLAN-unaware

The VLAN-unaware learning process follows the same rules as stated in 14.4.1.1 using only the NULL VID.

14.4.2 Querying the SDB

For a given VID, a given individual MAC address specification can be included in the association database in a static entry, a dynamic entry, both, or neither. Static entries take precedence over dynamic entries.

More than one static entry can match a frame depending on VID-to-FID mapping. If the SDB contains multiple static entries containing the same end station MAC address and same FID (but different VIDs) having different target addresses, the entry that is selected by SAS is not defined by this standard but should be selected consistently.

14.4.3 SDB entry removal

Dynamic SDB entries are removed for the following reasons:

- a) Aging timer expiry (14.3.4).
- b) Ring topology change that causes one or more stations to be unreachable (11.2.7).
- c) Remote client using SAS notify (12.2.3) to request an SDB purge of some or all entries associated with its station address.
- d) MLME request to purge selected entries found in the SDB. The parameters of the purge request are *purgeScope* and *purgeValue*. The relationship between *purgeScope* and *purgeValue* is shown in Table 14.6. The *sdbLocalPurges* counter shall be incremented by one for each request.

Table 14.6—purgeScope to purgeValue relationship

<i>purgeScope</i>	<i>purgeValue</i>
ALL	—
STATION	station address
VID	VID value
MSTI	MSTI value
VID_MAP	VID map (see Table 12.4)
MSTI_MAP	MSTI map (see Table 12.4)

Static SDB entries can be removed by station management.

A SAS-enabled station shall be capable of performing a complete purge of all dynamic entries in its SDB. Compliant implementations may be capable of performing partial purges of the dynamic entries in their SDBs. If a partial purge request is received, stations that cannot perform this action perform a complete purge instead. Table 14.7 describes all possible purges and whether they are requested via MLME, SAS notify, or topology change.

Table 14.7—Purge invocations

Purge type	Invocation
ALL	{MLME}
VID	{MLME}
MSTI	{MLME}
VID_MAP	{MLME}
MSTI_MAP	{MLME}
Target station address	{SAS notify, Topology change}
Target station & VID	{SAS notify}
Target station & VID_MAP	{SAS notify}
Target station & MSTI	{SAS notify}
Target station & MSTI_MAP	{SAS notify}

14.5 SAS operations

14.5.1 Principles of unicast operation

The principal elements of SAS unicast operation are as follows:

- a) Choosing directed or undirected transmission for frames.
- b) Associating a MAC address and/or VID with a ring local address.
- c) Maintenance of the information required to make frame transmission choices.
- d) Management of the above.

Information used by SAS operations is in the SDB tables and the VID-to-FID mapping table (see 14.3).

A VID refers to a VLAN identifier found in the client frame. An FID is an internal system identifier that denotes a grouping of one or more VIDs.

As per 9.3.2.3 in IEEE Std 802.1Q, frames transmitted by VLAN bridges will never contain a VLAN value of 0. These frames will either be transmitted as untagged or transmitted using another VLAN ID. Also, tagged frames will never contain a VLAN identifier value of 4095. These values are reserved. Host clients can transmit priority tagged frames having a VLAN identifier of 0. If the client frame does not contain a VLAN identifier (i.e., the frame is untagged), or if the client frame is only priority tagged (VID = 0), SAS will internally assign the NULL VID (i.e., VID = 0). Consequently, the operations on the aforementioned

tables need not change when processing tagged and untagged frames. The mechanism used to establish the VID-to-FID associations is outside the scope of this standard. Implementations should align with the mechanisms specified by 8.10.7 of IEEE Std 802.1Q.

The mechanism to modify information found in the SDB is described in 14.3.

14.5.1.1 VID-only operation

SAS supports VID-only static entries. The function of these entries is to resolve a target address for all transmitted frames (unicast or multicast) matching the static VID entry (with VID mapped to FID), regardless of the client destination address. These entries take precedence over configured entries in the SAS multicast tables.

14.5.1.2 Frame transmission

Transmit frames processed by SAS use either directed or undirected transmission, based on address associations in the SDB.

Transmission decisions are made based on:

- a) The value of the *enableSas* variable.
- b) The *destination_address* parameter of the MA_DATA.request primitive.
- c) The VID extracted from the *mac_service_data_unit* parameter of the MA_DATA.request primitive.
- d) The information contained in the SDB for the MAC address and VID.
- e) The *request_sas* parameter of the MA_DATA.request primitive.

SAS uses directed transmission if an entry for the destination address and VID is found in the SDB. Otherwise, the frame is sent undirected. When SAS is enabled, SAS will extend all data frames, unless the value of the *request_sas* parameter in the MA_DATA.request primitive is FALSE.

14.5.1.2.1 MAC service support

The MAC service interface is described in 6.4.

Figure 14.3 and Figure 14.4 illustrate the actions an enabled SAS takes on the MA_DATA.request fields in forming a data frame for directed and undirected transmissions, respectively. SAS controls the contents of the *da*, *sa*, and *fi* fields. For both types of transmissions, the station MAC address is used for the value of the *sa* field.

For a directed transmission, which is illustrated in Figure 14.3, the *destination_address* is used (along with the VID from the *mac_service_data_unit*) to query the SDB (i.e., look up) for a target address to be placed into the *da* field. The other frame fields are populated as shown, including setting the *fi* subfield of the *extendedControl* field to FI_NONE.

For an undirected transmission (e.g., the target address is not found), the *da* is set to the value of SAS_GROUP_ADDRESS, and the *fi* subfield of the *extendedControl* field is set to indicate that the frame is to be flooded, as illustrated in Figure 14.4. The other fields are populated as shown.

14.5.2 Principles of multicast operation

SAS multicast scoping is an optional feature. If implemented, it uses SDB lookup results to set multicast frame initial TTL values to only include stations requiring the transmissions.

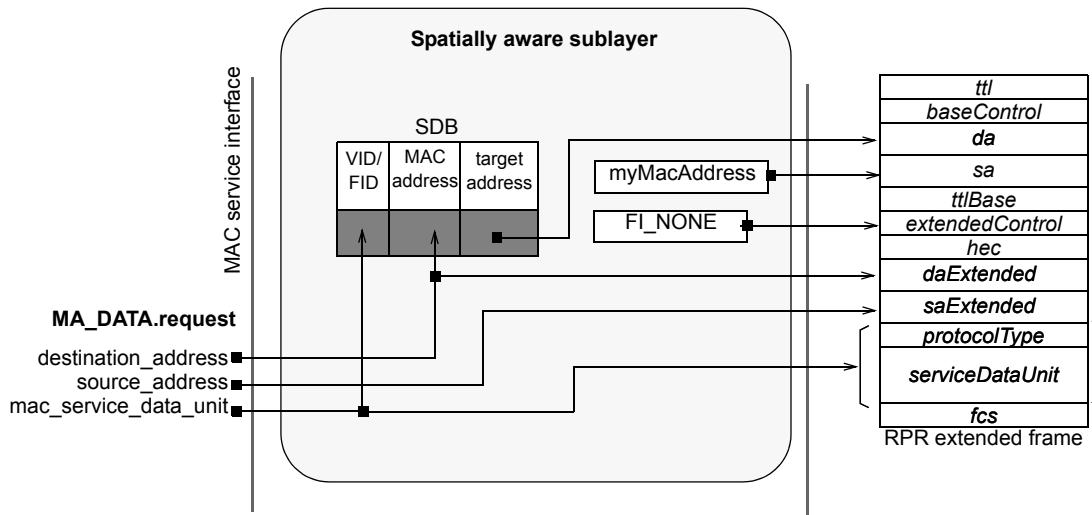


Figure 14.3—SAS directed transmission

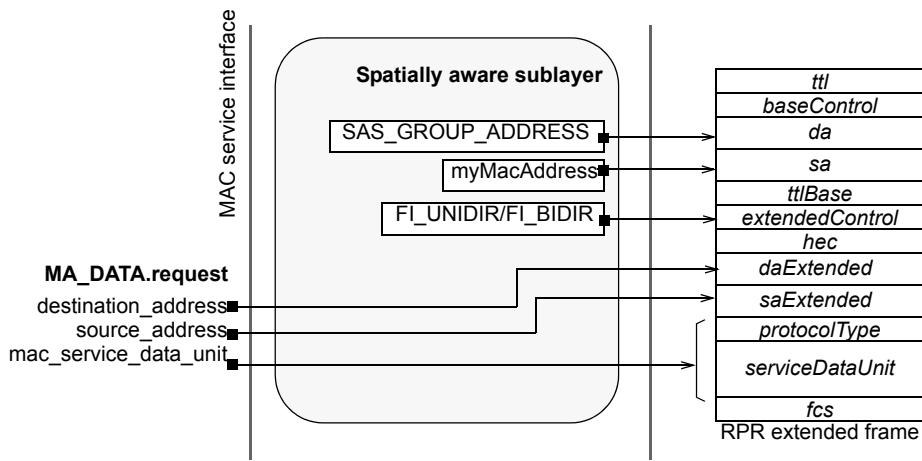


Figure 14.4—SAS undirected transmission

Scoping decisions are made based on:

- The value of the *enableSas* variable.
- The value of the *optionSasMcastScope* variable.
- The destination_address parameter of the MA_DATA.request primitive.
- The VID (if present) extracted from the mac_service_data_unit parameter of the MA_DATA.request primitive.
- The information contained in the SDB static group address table for the destination group address and VID.
- The request_sas parameter of the MA_DATA.request primitive.

On topology changes, station management is responsible for validating the scope of multicast entries to ensure:

- Multicast frames are delivered to the appropriate set of stations.
- No frame duplication occurs, by limiting the combined number of hops in each entry to be less than the number of stations in the ring.

Entries that do not pass these tests are updated or disabled.

SAS tests for multicast frames and passes them to the ringlet selection state machine. The frame is extended if *sdbStaticMcastExtend* is TRUE. The ringlet selection state machine then looks up the client destination group address and VID in the static group address table. If an entry is found, the value of the *ttl* field of the frame is set to scope the travel over the ring.

SAS multicast uses the VID/FID mapping table, and the static multicast management table, to manage static entries in the multicast lookup table. The multicast lookup table associates multicast address and FID with ringlet0/ringlet1 hops values.

14.5.3 Transmit operation

The transmit operation is initiated upon receipt of an MA_DATA.request (see 6.4.1).

14.5.3.1 SasTransmit state machine

This subclause specifies the state machine used by SAS to modify address information of frames being transmitted.

The following subclauses (14.5.3.1.1–14.5.3.1.3) describe parameters used within the context of this state machine.

14.5.3.1.1 SasTransmit state machine definitions

NULL

See 14.2.3.

Q_TX_RS

Q_TX_SAS

See 14.2.3.

RI_0

RI_1

See 14.2.3.

SAS_GROUP_ADDRESS

See 14.2.3.

14.5.3.1.2 SasTransmit state machine variables

destination_address

destination_address_extended

enableSas

See 14.2.3.

frame

The RPR data frame (see 9.2) constructed from the MA_DATA.request primitive (see 6.4.1).

flooding_form

See 14.2.3.

myMacAddress
 See 14.2.3.
optionSasTestReachability
 See 14.2.2.
request_sas
source_address
source_address_extended
 See 14.2.3.
targetAddr
 The target address returned from the *SdbLookup()* routine.

14.5.3.1.3 SasTransmit state machine routines

Dequeue(queue)
Enqueue(queue, frame)
Multicast(macAddress)
Provided(parameter)
Reachable(address, ringlet)
 See 14.2.3.
SasEncodeAddr()

This routine encodes the addresses for the frame based on the results of the SDB lookup, as specified in Equation (14.1).

```
Void SasEncodeAddr(address) (14.1)
{
    destination_address_extended = destination_address;
    destination_address = address;

    if (Provided(source_address))
    {
        source_address_extended = source_address;
        source_address = myMACAddress;
    }
    else
        source_address_extended = myMacAddress;

    if (Provided(flooding_form))
        if Unicast(destination_address)
            flooding_form = FI_NONE;
        else
            flooding_form = myFloodingForm;
}
```

SdbLookup(frame)

This routine returns the target address located by searching the *sdbUcastEntry* table using VID (mapped to FID) and *frame.da*. An *sdbEntryDestAddr* value of FF-FF-FF-FF-FF-FF (i.e., a static VID-only entry) matches all values of *frame.da*.

macAddress—The associated ring local address.

NULL—The result if no association is found in the SDB.

SasMulticastEncode(frame)

This routine determines whether this *frame* will receive multicast scoping treatment and, if so, how to encode the *frame*. It will return NULL (i.e., no multicast scoping) if there is a matching static VLAN-only entry in the *sdbStaticUcast* table. Otherwise a value of EXTEND, NOEXTEND, or NULL is returned based on the value of the VID and the value of the *da* field in the *frame*, from the *sdbMcastEntry* table.

EXTEND—A value of EXTEND is returned when the *frame* matches an entry within the *sdbMcastEntry* table and the *sdbStaticMcastExtend* associated with that entry is set to TRUE.

NOEXTEND—A value of NOEXTEND is returned when the *frame* matches an entry within the *sdbMcastEntry* table and the *sdbStaticMcastExtend* associated with that entry is set to FALSE.

NULL—(Otherwise.)

Unicast(macAddress)

See 14.2.3.

14.5.3.1.4 SasTransmit state table

The SasTransmit state machine specified in Table 14.8 implements the functions necessary to determine whether to send a frame using directed or undirected transmission and then modifies frame fields accordingly. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 14.8—SasTransmit state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_TX_SAS)) != NULL	1	—	TEST
	—	2	—	START
TEST	!enableSas	3	—	TRANSMIT
	Provided(request_sas) && !request_sas	4	—	
	—	5	—	EXTEND
EXTEND	SasMulticastEncode(frame) == EXTEND	6	SasEncodeAddr(SAS_GROUP_ADDRESS);	TRANSMIT
	SasMulticastEncode(frame) == NOEXTEND	7	—	
	(targetAddr = SdbLookup(frame)) != NULL && (!optionSasTestReachability Reachable(targetAddr, RI_0) Reachable(targetAddr, RI_1))	8	sasTxDirectedFrames += 1; SasEncodeAddr(targetAddr);	
	—	9	sasTxUnDirectedFrames += 1; SasEncodeAddr(SAS_GROUP_ADDRESS);	
TRANSMIT	—	10	Enqueue(Q_TX_RS, frame);	START

Row 14.8-1: A request to transmit a frame was received.

Row 14.8-2: No request to transmit a frame. Wait for such a request (generated from the MA_DATA.request primitive).

Row 14.8-3: SAS is not enabled.

Row 14.8-4: The client has explicitly indicated via the request_sas parameter that it wants to bypass SAS.

Row 14.8-5: Perform SAS processing on the frame.

Row 14.8-6: If the multicast frame matches an entry within the *sdbMcastEntry* table and the *sdbStaticMcastExtend* value associated with that entry is set to TRUE, then extend the frame and assign the SAS_GROUP_ADDRESS value to *frame.da*, prior to enqueueing the frame on the ringlet selection Tx queue.

Row 14.8-7: If the multicast frame matches an entry within the *sdbMcastEntry* table and the *sdbStaticMcastExtend* value associated with that entry is set to FALSE, then enqueue the frame on the ringlet selection Tx queue.

Row 14.8-8: Search the SDB for a matching entry and get its target address. If a target address is found (subject to the optional reachability test), then set the value of the *da* field to the target address.

Row 14.8-9: The address parameters are modified such that the value of the *da* field is the reserved value SAS_GROUP_ADDRESS.

Row 14.8-10: Move the frame to the ringlet selection queue.

14.5.4 SAS receive operation

This subclause specifies the SAS state machine for receiving frames from the MAC datapath and the actions taken after receipt of the frames.

The SAS receive operation provides the following functionality:

- a) Receiving data frames, and updating the SDB.
- b) Passing data frames to the MAC client.
- c) Mapping of frame addresses.

14.5.4.1 SasReceive state machine

This subclause specifies the state machine used by SAS to learn and modify address information of frames being received.

The following subclauses (14.5.4.1.1–14.5.4.1.3) describe parameters used within the context of this state machine.

14.5.4.1.1 SasReceive state machine definitions

FLOOD_NONE

See 14.2.3.

NULL

See 14.2.3.

Q_RX_PIRC

Q_RX_SAS

See 14.2.4.

SAS_GROUP_ADDRESS

See 14.2.3.

14.5.4.1.2 SasReceive state machine variables

flooding_form

See 14.2.3.

frame

The contents of an RPR frame.

remapRxAddrs

See 14.2.3.

14.5.4.1.3 SasReceive state machine routines

Contract(frame)

Modifies *frame* from an extended data frame to a basic data frame.

Dequeue(queue)

Enqueue(queue,frame)

See 14.2.3.

SdbLearn(frame)

Updates the SDB with the association among the *frame.sa*, *frame.saExtended*, and the VID of the frame, and resets the age of the entry. If this is a new association, and no free entry is available, increment the *sdbLearnFulls* counter (see 14.4.1).

Unicast(address)

See 14.2.3.

14.5.4.1.4 SasReceive state table

The SasReceive state machine specified in Table 14.9 implements the functions necessary for processing frames received by SAS from the ring. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 14.9—SasReceive state table

Current state		Row	Next state	
state	condition		action	state
START	(frame = Dequeue(Q_RX_SAS)) != NULL	1	—	LEARN
	—	2	—	START
LEARN	!enableSas	3	—	MAP
	(frame.ef == 1) && ((frame.da == SAS_GROUP_ADDRESS) (Unicast(frame.da) && flooding_form == FLOOD_NONE))	4	SdbLearn(frame);	
	—	5	—	
MAP	(frame.ef == 1) && (remapRxAddrs)	6	frame.sa = frame.saExtended; frame.da = frame.daExtended; frame.ef = 0; Contract(frame);	FINAL
	—	7	—	
FINAL	—	8	Enqueue(Q_RX_PIRC, frame);	START

Row 14.9-1: Fetch frame from the SAS queue.

Row 14.9-2: Since there is currently no new frame present in the SAS queue, continue polling for a frame in the SAS queue.

Row 14.9-3: SAS is not enabled, so skip SAS processing.

Row 14.9-4: Learn the *sa*, *saExtended*, and optional VID association, if appropriate.

Row 14.9-5: Otherwise prepare to send the frame to the client.

Row 14.9-6: Regardless of the setting of *enableSas*, if *remapRxAddrs* is TRUE, then alter extended frames by mapping source extended address to source address, destination extended address to destination address and contracting the frame.

Row 14.9-7: No need to manipulate the frame addresses prior to dispatch to the client.

Row 14.9-8: Pass the frame to the PIRC sublayer.

14.6 Protocol Implementation Conformance Statement (PICS) proforma for Clause 14²⁶

14.6.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 14, Spatially aware sublayer, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005 as amended.

14.6.2 Identification

14.6.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

14.6.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, Spatially aware sublayer
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

²⁶Copyright release for PICS proformas: Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

14.6.3 PICS tables for Clause 14**14.6.3.1 SAS**

Item	Feature	Subclause	Value/Comment	Status	Support
SAS*	SAS support	14.1	Spatially aware sublayer supported	O	Yes [] No []

14.6.3.2 State machines

Item	Feature	Subclause	Value/Comment	Status	Support
SM1	SasTransmit	14.5.3.1.4	SasTransmit state table is supported	SAS:M	Yes []
SM2	SasReceive	14.5.4.1.4	SasReceive state table is supported	SAS:M	Yes []

14.6.3.3 SDB entries

Item	Feature	Subclause	Value/Comment	Status	Support
SE1	Dynamic entry removal	14.3.4	Automatically causes dynamic entry to be removed after a specified time.	SAS:M	Yes []
SE2	Single dynamic entry per MAC and FID combination	14.3.4	A single dynamic entry will be created for a given MAC and FID combination.	SAS:M	Yes []
SE3	Support FIDs	14.3.5	VLAN-aware SAS can support between 1 and 4095 FIDs.	SAS:M	Yes []
SE4	Dynamic entry addition	14.4.1	A new dynamic entry can replace an existing entry.	SAS:O	Yes [] No []

14.6.3.4 SDB purges

Item	Feature	Subclause	Value/Comment	Status	Support
SP1	<i>sdbLocalPurges</i> counter increment	14.4.3	Increment <i>sdbLocalPurges</i> counter by 1 each time MLME request to purge selected entries found in the SDB.	SAS:M	Yes []
SP2	Complete purge of SDB	14.4.3	Support complete purge of all dynamic entries in the SDB.	SAS:M	Yes []
SP3	Partial purge of SDB	14.4.3	Support partial purge of dynamic entries in the SDB.	SAS:O	Yes [] No []

14.6.3.5 Multicast

Item	Feature	Subclause	Value/Comment	Status	Support
MC1	Multicast scoping	14.5.2	SAS multicast scoping supported	SAS:O	Yes [] No []

15. Protected inter-ring connection

15.1 Protected inter-ring connection overview

Protected inter-ring connection (PIRC) is an optional sublayer of the MAC that may be implemented. PIRC supports rings interconnected through dual-station homing, providing fast recovery, loop prevention, and load balancing features. The interconnecting stations may be a discrete network element on each ring with back-to-back connections or a single network element with multiple stations on each of the rings.

PIRC provides protection from interconnect station failures in less than 50 ms. Non-interconnect stations benefit from this fast recovery without any need to implement the optional PIRC sublayer. PIRC also provides maintenance commands (e.g., manual switch-over) to facilitate servicing of interconnect stations with no user impact.

PIRC always prevents loops in steady-state operation. Temporary loops might occur during the transient process between normal and failure states.

An active/standby algorithm is the mandatory method for forwarding across an interconnect. PIRC defines several load balancing options for an operator. Optionally, load balancing algorithms based upon VLAN hash, static VLAN assignments, or other methods are allowed but not specified in this standard. These methods must follow several defined constraints. All load balancing designs shall allow unicast traffic to be forwarded without persistent flooding or loss of connectivity. Any hashing required will not be based on MAC addresses.

15.1.1 PIRC stations

A pair of PIRC stations are shown in Figure 15.1. PIRC mate stations (Station A and Station B) are stations on the same interconnected ring. Alternatively, Stations C and D can also be PIRC mate stations (instead of Stations A and B). In a topology of multiple rings, all the rings should have protection group members with the exception of one of the rings, in order to provide connectivity between the stations without loops.

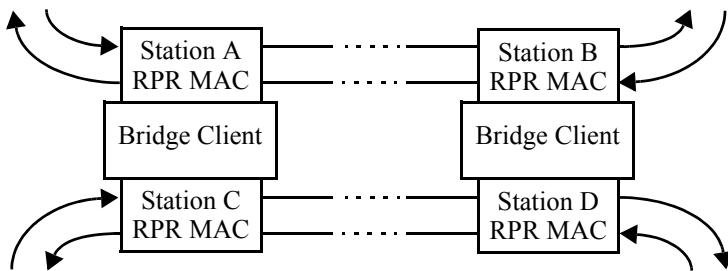


Figure 15.1—PIRC station types

15.1.1.1 PIRC information communication between PIRC mate stations

PIRC messages are exchanged between PIRC mate stations and consist of the following information:

- a) ok—indication that the station is alive, the operational status is up, no misconfiguration has been detected, and the protection status is not in a failure, forced switch, or wait to restore condition.
- b) ms—indication that the station is in a manual switch protection state.
- c) pr—indication that the station is provisioned as the primary protection group member.
- d) lb—indication that the station is provisioned in PIRC load balancing mode.

These mate messages are also used as a keep-alive to detect the existence of a provisioned PIRC mate station.

15.1.2 PIRC load balancing

By default there is no load balancing. If load balancing is implemented, it is implementation specific.

NOTE—The ATD organization-specific ATT defined in 11.4.8 could be used to provide information on implementation-specific load balancing.

15.1.2.1 Load-balancing based on VLAN hashing

VLAN-based hashing is one of the possible methods for the vendor specific load-balancing implementations. In this method, the function pircMyFlow() (see 15.4.1) is implemented as a comparison of the result of a bit-wise-xor of the VID of the forwarded frame to ‘1’.

15.1.2.2 Load balancing based on static VLAN assignments

VLANs are assigned to one of the PIRC interconnecting stations by manual configuration to achieve load balancing. On each PIRC interconnecting station, the suitable VLAN range is configured, and it must be guaranteed that the intersection of the VLAN ranges configured at different interconnecting stations is empty. It is judged by the PIRC interconnecting stations based on the VLAN carried, whether the copy of the broadcast traffic is forwarded to the inter-ring. Each PIRC interconnecting stations only forwards the broadcast traffic from VLANs assigned to it. The unicast traffic is forwarded along the path learned through the broadcast traffic.

15.1.3 PIRC misconfiguration detection

There are two possible misconfigurations that can be detected:

- a) In a protection group both members are primary or both members are secondary.
- b) The members of a group have different settings for load balancing or not.

15.1.4 PIRC failure conditions

The following PIRC failure conditions are applicable for a PGM:

- a) Detection of PIRC misconfiguration.
- b) Loss of connectivity with PIRC mate station.

15.1.5 PIRC maintenance commands

The following PIRC maintenance commands are applicable for a PGM:

- a) PIRC forced switch command—A management directive that forces the PGM to deactivate inter-ring connectivity.
- b) PIRC manual switch command—A management directive that (if not overridden) deactivates the PGM inter-ring connectivity.
- c) PIRC idle command—A management directive that (if not overridden) activates PGM inter-ring connectivity.

15.1.6 PIRC reversion mode

A mode of operation of a PIRC station in which, upon clearing of PIRC failures, the station remains in a wait to restore protection state until the wait to restore condition is preempted by a higher priority protection condition elsewhere on the protection group, or PIRC_IDLE administrative request.

15.1.7 PIRC protection hierarchy

The effect of the protection condition and commands depends on the severity order, listed below, where the top-through-bottom conditions are listed in order of most-through-least severity. (See 15.2.1 for definitions of these conditions.)

- a) PIRC_FORCED_SWITCH
- b) PIRC_FAILURE
- c) PIRC_PROTECTING
- d) PIRC_MANUAL_SWITCH
- e) PIRC_WTR
- f) PIRC_CONTAINMENT
- g) PIRC_NO_REQUEST

NOTE—The value of the containment timer should be provisioned to the sum of the PIRC mate message forwarding delay + the message processing and protection update delay + ring transit latency.

15.1.8 PIRC ring-interconnection forwarding states

Three ring interconnection forwarding states are supported:

- a) Forward all—The PIRC station disables the PIRC filtering in the inter-ring connection. This forwarding state is engaged while the PIRC protection status is PIRC_PROTECTING. In addition, for non-load-balancing primary protection group members, this forwarding state is engaged while the PIRC protection status is PIRC_NO_REQUEST.
- b) Forward mine—The PIRC station enables the PIRC selective filtering in the inter-ring connection. This forwarding state is applicable only to PIRC load-balancing stations. This forwarding state is engaged while the PIRC protection status is PIRC_NO_REQUEST or PIRC_CONTAINMENT state if preceded by PIRC_NO_REQUEST state.
- c) Forward none—The PIRC station deactivates the inter-ring connection. This forwarding state is engaged while the PIRC protection status is PIRC_FORCED_SWITCH, PIRC_FAILURE, PIRC_WTR, PIRC_MANUAL_SWITCH, or PIRC_CONTAINMENT unless preceded by PIRC_NO_REQUEST state. In addition, for non load balancing secondary protection group members, this forwarding state is engaged while the PIRC protection status is PIRC_NO_REQUEST or PIRC_CONTAINMENT.

15.2 Terminology and variables

15.2.1 Common state machine definitions

PIRC_CONTAINMENT

The PIRC state preceding PGM activation of inter-ring connectivity, the PIRC containment timer prevents frame duplication during PIRC transient events.

PIRC_FAILURE

The PIRC state during PIRC misconfiguration condition, or loss of ring interface.

PIRC_FORCED_SWITCH

The PIRC state following PIRC creation, or subsequent to a PIRC forced switch administrative command.

PIRC_MANUAL_SWITCH

The PIRC state following PIRC manual switch administrative command.

PIRC_NO_REQUEST

The PIRC state while PIRC is not deactivated using management directive or other failure condition.

PIRC_PROTECTING

The PIRC state while PIRC mate station deactivates inter-ring connectivity.

PIRC_WTR

The PIRC state after PIRC_FAILURE, the PIRC wait-to-restore timer improves stability in the presence of transient failures.

15.2.2 Common state machine variables*adminPircReq*

An administrative PIRC protection request, with the following values:

PIRC_FS—Indicates a PIRC forced switch command.

PIRC_IDLE—Indicates the clearing of PIRC_WTR state or a PIRC forced switch or PIRC manual switch command.

PIRC_MS—Indicates a PIRC manual switch command.

PIRC_NULL—No PIRC administrative protection request is available.

15.2.3 Common state machine routines*pircConfigDefect()*

Determines if a configuration defect exists, as specified by Equation (15.1).

$$(\text{pircLbDefect} \mid\mid \text{pircPrDefect}) \quad (15.1)$$

pircContainmentExpired()

Determines if the protection state is PIRC_CONTAINMENT state and the timer expired, as described by Equation (15.2).

$$((\text{pircInfo.protState} == \text{PIRC_CONTAINMENT}) \&\& \\ ((\text{currentTime} - \text{pircInfo.startTime}) \geq \text{pircProv.pcTimeout})) \quad (15.2)$$

pircFwdAll()

Determines if the protection group member should forward all data frames (see 15.1.8), as described by Equation (15.3).

$$((\text{pircInfo.protState} == \text{PIRC_PROTECTING}) \mid\mid \\ !\text{pircProv.pircLoadBalancing} \&\& \\ \text{pircProv.pircPrimary} \&\& \\ (\text{pircInfo.protState} == \text{PIRC_NO_REQUEST})) \quad (15.3)$$

pircFwdNone()

Determines if the protection group member should not forward all data frames (see 15.1.8), as described by Equation (15.4).

```
( (pircInfo.protState == PIRC_FS) ||
  (pircInfo.protState == PIRC_FAILURE) ||
  (pircInfo.protState == PIRC_MS) ||
  (pircInfo.protState == PIRC_WTR) ||
  !pircInfo.pcFwd &&
  (pircInfo.protState == PIRC_CONTAINMENT) ||
  !pircProv.pircLoadBalancing &&
  !pircProv.pircPrimary &&
  (pircInfo.protState == PIRC_NO_REQUEST) )
```

(15.4)
pircMateMsgTimeOut()

Determines if PIRC mate messages are no longer received at the expected rate, as described by Equation (15.5).

```
((currentTime - pircMateMsg.lastTimeStamp) >= pircProv.mateKeepAliveInterval) (15.5)
```

pircMyFlow(frame)

Performs load-balancing forwarding decision. Implementation is vendor specific.

TRUE—The protection group member should forward the data frame.

FALSE—(Otherwise.)

pircWtrExpired()

Determines if the protection state is the PIRC_WTR state and if the timer expired, as described by Equation (15.6).

```
((pircInfo.protState == PIRC_WTR) && pircProv.pircRevertive &&
 ((currentTime-pircInfo.startTime) >= pircProv.wtrTimeout))
```

(15.6)

15.2.4 Variables and literals defined in other clauses

This clause references the following parameters, literals, variables, and routines defined in Clause 6:

enablePirc

This clause references the following parameters, literals, variables, and routines defined in Clause 7:

currentTime
Dequeue()
Enqueue()
NULL
Q_TX_OAM
Q_TX_PIRC
Q_TX_RS

This clause references the following literals defined in Clause 9:

CT_OAM_PIRC_STATUS

This clause references the following parameters, literals, variables, and routines defined in Clause 11:

pircLbDefect
pircPrDefect
pircTopologyChange

This clause references the following parameters, literals, variables, and routines defined in Clause 14:

Reachable()

This clause references the following parameters, literals, variables, and routines defined in Annex D:

ifOperStatus

15.3 PIRC database

The PIRC database is partitioned into multiple components, as described in Figure 11.29 and listed below.

- a) Provisioning information (see 15.3.1)
- b) Dynamic information (see 15.3.2)
- c) Mate information (see 15.3.3)

15.3.1 pircProv provisioning database

The pircProv database consists of all the PIRC administrative provisioning for the PIRC station. See Table 15.1.

Table 15.1—PIRC pircProv database

Variable	Description	Values
pircRevertive	Whether the PIRC protection is revertive	Range: [TRUE, FALSE] Default: FALSE
pircLoadBalancing	Whether PIRC load balances	Range: [TRUE, FALSE] Default: FALSE
fastMateMsgRpt	The number of times a PIRC mate message should be retransmitted after PIRC status change	Range: [1, 10] Resolution: 1 Default: 3
fastMateMsgInterval	The time interval between PIRC mate message retransmission after PIRC status change	Range: [1 ms, 1000 ms] Resolution: 1 ms Default: 10 ms
slowMateMsgInterval	The time interval between periodic PIRC mate message	Range: [10 ms, 10000 ms] Resolution: 10 ms Default: 100 ms
wtrTimeout	PIRC WTR timeout interval	Range: [0 s, 100 s] Resolution: 1 s Default: 10 s
pcTimeout	PIRC containment timeout interval	Range: [0 ms, 1000 ms] Resolution: 10 ms Default: 50 ms
mateKeepAliveInterval	The time interval without reception of any PIRC mate message for determining of PIRC mate connectivity loss	Range: [30 ms, 100000 ms] Resolution: 10 ms Default: 350 ms
pircPrimary	Indicates this station is the primary station in the protection group	Range: [TRUE, FALSE] Default: TRUE

15.3.2 pircInfo dynamic information database

The pircInfo database consists of all the PIRC dynamic information for the PIRC station. See Table 15.2.

Table 15.2—PIRC pircInfo database

Variable	Description	Values
protState	PIRC protection state (see 15.1.8)	—
startTime	triggering time of the WTR or containment timer	—
ifOperStatus	the station ifOperStatus at the time of the last PIRC status update	—
active	indicates if the PIRC station could perform ring-interconnection functionality	Range: [TRUE, FALSE]
pcFwd	relevant only for load-balancing PIRC stations – indicates if the PIRC station should forward its PIRC flows during PIRC containment state	Range: [TRUE, FALSE]
pircMateLoss	indicates loss of communication with the PIRC Mate	Range: [TRUE, FALSE]

15.3.3 pircMateMsg mate information database

The pircMateMsg database consists of all the parameters related to PIRC mate messages for the PIRC station. See Table 15.3.

Table 15.3—PIRC pircMateMsg database

Variable	Description	Values
fastMsgCnt	number of fast PIRC mate messages remained to be retransmitted (after PIRC status change)	Range: [0, 10] Resolution: 1
lastTimeStamp	the time of the last received PIRC mate message	—
macAddress	MAC address of last received PIRC mate message	—
ok	the ok field from the last received PIRC mate message	Range: [TRUE, FALSE]
ms	the ms field from the last received PIRC mate message	Range: [TRUE, FALSE]
pr	the pr field from the last received PIRC mate message	Range: [TRUE, FALSE]
lb	the lb field from the last received PIRC mate message	Range: [TRUE, FALSE]

15.4 State machines

15.4.1 PircMateMsgReceive state machine

15.4.1.1 PircMateMsgReceive state machine overview

This subclause specifies the state machine for processing PIRC mate messages.

15.4.1.2 PircMateMsgReceive state machine definitions

There are no state machine definitions defined within this subclause.

15.4.1.3 PircMateMsgReceive state machine variables

currentTime

See 15.2.4.

enablePirc

See 15.2.4.

pircInfo

See 15.3.2.

pircLbDefect

See 15.2.4.

pircMateMsg

See 15.3.3.

pircPrDefect

See 15.2.4.

pircProv

See 15.3.1.

15.4.1.4 PircMateMsgReceive state machine routines

There are no state machine routines defined within this subclause.

15.4.1.5 PircMateMsgReceive state table

Table 15.4—PircMateMsgReceive state table

Current state		Row	Next state	
State	Condition		Action	State
START	!enablePIRC	1	—	START
RX	—	2	<p><i>pircInfo.pircMateLoss</i> = FALSE;</p> <p><i>pircMateMsg.lastTimeStamp</i> = <i>currentTime</i>;</p> <p><i>pircMateMsg.macAddress</i> = <i>frame.sa</i>;</p> <p><i>pircMateMsg.ok</i> = <i>frame.ok</i>;</p> <p><i>pircMateMsg.ms</i> = <i>frame.ms</i>;</p> <p><i>pircMateMsg.lb</i> = <i>frame.lb</i>;</p> <p><i>pircMateMsg.ms</i> = <i>frame.ms</i>;</p> <p><i>pircLbDefect</i> = (<i>pircProv.pircLoadBalancing</i> != <i>pircMateMsg.lb</i>);</p> <p><i>pircPrDefect</i> = (<i>pircProv.pircPrimary</i> == <i>pircMateMsg.pr</i>);</p>	RETURN

Row 15.4-1: Do not process received PIRC status messages if PIRC is not enabled.

Row 15.4-2: Communication with PIRC mate is available, and update *pircMateMsg* database.

15.4.2 PircControl state machine

15.4.2.1 PircControl state machine overview

This subclause specifies the state machine for determining if the PIRC protection state should be updated and triggering of fast transmission of PIRC mate messages.

15.4.2.2 PircControl state machine definitions

PIRC_NULL

See 15.2.1.

15.4.2.3 PircControl state machine variables

adminPircReq

See 15.2.2.

currentTime

See 15.2.4.

enablePirc

See 15.2.4.

ifOperStatus

See 15.2.4.

oldPircInfo

An image of the PIRC info database before an update for comparison to determine if changes occurred.

pircInfo

See 15.3.2.

pircLbDefect

See 15.2.4.

pircMateMsg

See 15.3.3.

pircPrDefect

See 15.2.4.

pircProv

See 15.3.1.

pircTopologyChange

See 15.2.4.

15.4.2.4 PircControl state machine routines

The routines below are used in the TopologyControl state machine.

pircContainmentExpired()

See 15.2.3.

pircMateLostEvent()

Determines if the event of PIRC mate communication loss has occurred, as described by Equation (15.7).

$$(\text{!pircInfo.pircMateLoss \&& pircMateMsgTimeOut}()) \quad (15.7)$$

pircMateMsgChange()

Determines if there is a change in the PIRC mate station message, as described by Equation (15.8).

$$(\text{pircOldMateMsg} \neq \text{pircMateMsg}) \quad (15.8)$$

pircProtUpdate()

A routine whose functionality is defined by the PircProtectionUpdate state machine.

pircWtrExpired()

See 15.2.3.

15.4.2.5 PircControl state table

Table 15.5—PircControl state table

Current State		Row	Next State	
State	Condition		Action	State
START	!enablePirc	1	—	START
CHK	pircInfo.ifOperStatus != ifOperStatus	2	oldPircInfo = pircInfo; pircTopologyChange = FALSE; pircProtUpdate();	TXMSG
	adminPircReq != PIRC_NULL	3		
	pircTopologyChange	4		
	pircMateMsgChange()	5		
	pircMateMsgLostEvent()	6		
	pircWtrExpired()	7		
	pircContainmentExpired()	8		
	—	9	—	START
TXMSG	oldPircInfo != pircInfo	10	pircMateMsg.fastMsgCnt = pircProv.fastMateMsgRpt;	START
	—	11	—	

Row 15.5-1: Do not process received PIRC status messages if PIRC is not enabled.

Row 15.5-2: Update the PIRC protection status if the station interface operStatus has changed.

Row 15.5-3: Update the PIRC protection status following a PIRC administrative protection request.

Row 15.5-4: Update the PIRC protection status following a topology change event.

Row 15.5-5: Update the PIRC protection status following a change of the mate PIRC message.

Row 15.5-6: Update the PIRC protection status following an event of PIRC mate communication loss.

Row 15.5-7: Update the PIRC protection status when PIRC WTR timer expires.

Row 15.5-8: Update the PIRC protection status when PIRC containment timer expires.

Row 15.5-9: No need to update the PIRC protection status.

Row 15.5-10: If the PIRC protection status has changed reset the fast mate messages transmit counter.

Row 15.5-11: No change in the PIRC protection status.

15.4.3 PircProtectionUpdate state machine

15.4.3.1 PircProtectionUpdate state machine overview

This subclause specifies the state machine for updating the PIRC protection status.

15.4.3.2 PircProtectionUpdate state machine definitions

PIRC_CONTAINMENT

See 15.2.1.

PIRC_FAILURE

See 15.2.1.

PIRC_FORCED_SWITCH

See 15.2.1.

PIRC_FS

See 15.2.2.

PIRC_IDLE
 See 15.2.2.
PIRC_MANUAL_SWITCH
 See 15.2.2.
PIRC_MS
 See 15.2.2.
PIRC_NO_REQUEST
 See 15.2.1.
PIRC_NULL
 See 15.2.2.
PIRC_PROTECTING
 See 15.2.1.
PIRC_WTR
 See 15.2.1.

15.4.3.3 PircProtectionUpdate state machine variables

currentTime
 See 15.2.4.
ifOperStatus
 See 15.2.4.
pircInfo
 See 15.3.2.
pircMateMsg
 See 15.3.3.
pircOldMateMsg
 An image of the pircMateMsg data base at the time of the last PIRC protection status update.
pircProv
 See 15.3.1.

15.4.3.4 PircProtectionUpdate state machine routines

pircContainmentExpired()
 See 15.2.3.
pircMateMsgTimeOut()
 See 15.2.3.
pircMateOk()
 Determines if the PIRC mate station is capable of being activated for ring-interconnection, as described by Equation (15.9).

```
(!pircMateMsgTimeOut() && pircMateMsg.ok &&
(Reachable(pircMateMsg.macAddress, RI_0) ||
Reachable(pircMateMsg.macAddress, RI_1)))
```

(15.9)

pircStationOk()
 Determines if the PIRC station is capable of being activated for ring-interconnection, as described by Equation (15.10).

```
(pircInfo.ifOperStatus == up) && !pircConfigDefect())
```

(15.10)

pircWtrExpired()
 See 15.2.3.

15.4.3.5 PircProtectionUpdate state table

Table 15.6—PircProtectionUpdate state table

Current state		Row	Next state	
State	Condition		Action	State
START	—	1	pircInfo.ifOperStatus = ifOperStatus; pircOldMateMsg = pircMateMsg; pircInfo.pircMateLoss = pircMateMsgTimeOut();	ADMIN
ADMIN	(adminPircReq == PIRC_FS) (pircInfo.protState == PIRC_FORCED_SWITCH)	2	pircInfo.protState = PIRC_FORCED_SWITCH; pircInfo.active = FALSE;	FINAL
	!pircStationOk()	3	pircInfo.protState = PIRC_FAILURE; pircInfo.active = FALSE;	
	!pircMateOk() ((pircInfo.protState != PIRC_MANUAL_SWITCH) && pircMateMsg.ms)	4	pircInfo.active = TRUE;	PROTECT
	(adminPircReq != PIRC_IDLE) && !pircMateMsg.ms && ((adminPircReq == PIRC_MS) (pircInfo.protState == PIRC_MANUAL_SWITCH))	5	pircInfo.protState = PIRC_MANUAL_SWITCH; pircInfo.active = FALSE;	FINAL
	—	6	—	NOFAIL
PROTECT	(pircInfo.protState == PIRC_PROTECTING) (!pircProv.pircLoadBalancing && (pircInfo.protState == PIRC_NO_REQUEST))	7	pircInfo.protState = PIRC_PROTECTING	FINAL
	pircInfo.protState != PIRC_CONTAINMENT	8	pircInfo.protState = PIRC_CONTAINMENT; pircInfo.startTime = currentTime; pircInfo.pcFwd = (pircInfo.protState == PIRC_NO_REQUEST);	
	pircContainmentExpired()	9	pircInfo.protState = PIRC_PROTECTING;	
	—	10	—	

Table 15.6—PircProtectionUpdate state table (continued)

Current state		Row	Next state	
State	Condition		Action	State
NOFAIL	pircInfo.protState == PIRC_NO_REQUEST	11	—	FINAL
	pircInfo.protState == PIRC_PROTECTING	12	pircInfo.protState = PIRC_NO_REQUEST;	
	(pircInfo.protState == PIRC_CONTAINMENT) && pircInfo.pcFwd	13	pircInfo.protState = PIRC_NO_REQUEST;	
	pircCONTAINMENTExpired()	14	pircInfo.protState = PIRC_NO_REQUEST;	
	pircInfo.protState == PIRC_FAILURE	15	pircInfo.protState = PIRC_WTR; pircInfo.startTime = currentTime;	
	(pircInfo.protState == PIRC_WTR) && (adminPircReq == PIRC_NULL) && !pircWtrExpired()	16	—	
	pircInfo.protState != PIRC_CONTAINMENT	17	pircInfo.protState = PIRC_CONTAINMENT; pircInfo.startTime = currentTime; pircInfo.pcFwd = FALSE; pircInfo.active = TRUE;	
FINAL	—	18	adminPircReq = PIRC_NULL;	RETURN

Row 15.6-1: Save the current PIRC protection info and PIRC mate message, and set pircMateLoss.

Row 15.6-2: Check if current state is PIRC_Force_Switch or such administrative request was issued.

Row 15.6-3: Check for local failure in the PIRC station.

Row 15.6-4: Check if PIRC protection is required for PIRC mate.

Row 15.6-5: Check if current state is PIRC_Manual_Switch or such administrative request was issued.

Row 15.6-6: No PIRC failure or other PIRC maintenance request was issued.

Row 15.6-7: Remain in PIRC_PROTECTING state, or go into that state if PIRC is idle and not load balancing.

Row 15.6-8: If the current state is not PIRC_CONTAINMENT, go into PIRC_CONTAINMENT and set the start time and the pcFwd indication according to current state.

Row 15.6-9: If PIRC containment expired go into PIRC_PROTECTING state.

Row 15.6-10: If PIRC containment has not expired remain in PIRC_CONTAINMENT state

Row 15.6-11: No additional action is needed if the PIRC protection state is already PIRC_NO_REQUEST.

Row 15.6-12: If the current state is PIRC_PROTECTING go into PIRC_NO_REQUEST state.

Row 15.6-13: If the current state is PIRC_CONTAINMENT and forwarding is allowed, go into PIRC_NO_REQUEST state.

Row 15.6-14: If PIRC containment timer expired go into PIRC_NO_REQUEST state.

Row 15.6-15: If the current state is PIRC_FAILURE go into PIRC_WTR and set the start time.

Row 15.6-16: If WTR timer not expired and no PIRC administrative request remain in PIRC_WTR state.

Row 15.6-17: If PIRC state is not PIRC_CONTAINMENT, go into PIRC_CONTAINMENT and set the start time and the pcFwd indication according to current state.

Row 15.6-18: Clear the administrative request indication and return.

15.4.4 PircMateMsgTransmit state machine

15.4.4.1 PircMateMsgTransmit state machine overview

This subclause specifies the state machine for transmission of the PIRC mate status messages.

15.4.4.2 PircMateMsgTransmit state machine definitions

`CT_OAM_PIRC_STATUS`

See 15.2.4.

`Q_TX_OAM`

See 15.2.4.

15.4.4.3 PircMateMsgTransmit state machine variables

currentTime

See 15.2.4.

enablePirc

See 15.2.4.

frame

The contents of an RPR frame.

pircInfo

See 15.3.2.

pircMateMsg

See 15.3.3.

pircProv

See 15.3.1.

15.4.4.4 PircMateMsgTransmit state machine routines

Enqueue()

See 15.2.4.

15.4.4.5 PircMateMsgTransmit state table**Table 15.7—PircMateMsgTransmit state table**

Current State		Row	Next State	
State	Condition		Action	State
START	!enablePirc	1	—	START
CHK	pircMateMsg.fastMsgCnt == pircProv.fastMateMsgRpt	2	pircMateMsg.fastMsgCnt--;	TX
	(pircMateMsg.fastMsgCnt > 0) && (currentTime – pircMateMsg.lastMsgTime >= pircProv.fastMateMsgInterval)	3		
	currentTime – pircMateMsg.lastMsgTime >= pircProv.slowMateMsgInterval	4		
	—	5	—	START
TX	—	6	pircMateMsg.lastMsgTime = currentTime; frame.controlType = CT_OAM_PIRC_STATUS; frame.status.ok = pircInfo.active; frame.status.ms = pircInfo.protState == PIRC_MANUAL_SWITCH; frame.status.lb = pircProv.pircLoadBalancing; frame.status.pr = pircProv.pircPrimary; Enqueue(Q_TX_OAM, frame);	START

Row 15.7-1: Do not transmit PIRC status messages if PIRC is not enabled.

Row 15.7-2: If the fast PIRC message counter equals the maximum value, decrement the counter and send a message.

Row 15.7-3: If the fast PIRC message counter is greater than zero and the fast message transmission interval has elapsed from the previous message transmission, decrement the counter and send a message.

Row 15.7-4: If the slow message transmission interval has elapsed from the previous message transmission send a message.

Row 15.7-5: No need to send a PIRC mate status message.

Row 15.7-6: Set the message transmission time, frame parameters and enqueue the frame.

15.5 PIRC operations

15.5.1 PIRC receive operation

This subclause specifies the PIRC state machine for receiving frames from the MAC datapath and the actions taken after receipt of the frames. The PIRC receive operation provides frame filtering.

15.5.1.1 PircReceive state machine

15.5.1.1.1 PircReceive state machine definitions

NULL

See 15.2.4.

Q_RX_PIRC

See 15.2.4.

Q_RX_CLIENT

See 7.2.1.

15.5.1.1.2 PircReceive state machine variables

enablePirc

See 15.2.4.

frame

The contents of an RPR frame.

pircMateMsg

See 15.3.3.

15.5.1.1.3 PircReceive state machine routines

Dequeue()

See 15.2.4.

Enqueue()

See 15.2.4.

pircFwdAll()

See 15.2.3.

pircFwdNone()

See 15.2.3.

pircMateMsgTimeOut()

See 15.2.3.

pircMyFlow()

See 15.2.3.

15.5.1.1.4 PircReceive state table**Table 15.8—PircReceive state table**

Current State		Row	Next State	
State	Condition		Action	State
START	(frame = Dequeue(Q_RX_PIRC)) != NULL	1	—	CHK
	—	2	—	START
CHK	!enablePirc	3	—	FINAL
	pircFwdNone()	4	—	START
	!pircMateMsgTimeOut() && (frame.sa == pircMateMsg.macAddress)	5	—	
	pircFwdAll()	6	—	FINAL
	!pircMyFlow(frame)	7	—	START
FINAL	—	8	Enqueue(Q_RX_CLIENT, frame);	START

Row 15.8-1: Fetch frame from the PIRC receive queue.

Row 15.8-2: Since there is currently no new frame present in the PIRC receive queue, continue polling for a frame in the PIRC receive queue.

Row 15.8-3: PIRC is not enabled, so skip PIRC processing.

Row 15.8-4: This protection group member should not forward any frame.

Row 15.8-5: Data communication between the clients of the protection group members should be through another “non PIRC” ring connected to the client of both stations.

Row 15.8-6: This protection group member should forward all frames.

Row 15.8-7: This protection group member should perform load-balancing forwarding decision.

Row 15.8-8: Pass the frame to the client.

15.5.2 PIRC transmit operation

This subclause specifies the PIRC state machine for receiving frames from the MAC client and the actions taken after receipt of the frames. The PIRC transmit operation provides frame filtering.

15.5.2.1 PircTransmit state machine**15.5.2.1.1 PircTransmit state machine definitions**

NULL

See 15.2.4.

Q_TX_PIRC

See 15.2.4.

Q_TX_RS

See 15.2.4.

15.5.2.1.2 PircTransmit state machine variables

enablePirc

See 15.2.4.

frame

The contents of an RPR frame.

15.5.2.1.3 PircTransmit state machine routines

Dequeue()

See 15.2.4.

Enqueue()

See 15.2.4.

pircFwdAll()

See 15.2.3.

pircFwdNone()

See 15.2.3.

pircMyFlow()

See 15.2.3.

15.5.2.1.4 PircTransmit state table

Table 15.9—PircTransmit state table

Current State		Row	Next State	
State	Condition		Action	State
START	(frame = Dequeue(Q_TX_PIRC)) != NULL	1	—	CHK
	—	2	—	START
CHK	!enablePirc	3	—	FINAL
	pircFwdNone()	4	—	START
	pircFwdAll()	5	—	FINAL
	!pircMyFlow(frame)	6	—	START
FINAL	—	7	Enqueue(Q_TX_RS, frame);	START

Row 15.9-1: A request to transmit a frame was received.

Row 15.9-2: No request to transmit a frame. Wait for such a request (generated from the MA_DATA.request primitive).

Row 15.9-3: PIRC is not enabled.

Row 15.9-4: This protection group member should not forward any frame.

Row 15.9-5: This protection group member should forward all frames.

Row 15.9-6: This protection group member should perform load-balancing forwarding decision.

Row 15.9-7: Move the frame to the ringlet selection queue.

15.6 Protocol Implementation Conformance Statement (PICS) proforma for Clause 15²⁷

15.6.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 15, Protected inter-ring connection, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005 as amended.

15.6.2 Identification

15.6.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

15.6.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, Protected inter-ring connection
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

²⁷Copyright release for PICS proformas: Users of this standard may freely reproduce the PICS proforma in this clause so that it can be used for its intended purpose and may further publish the completed PICS.

15.6.3 PICS tables for Clause 15**15.6.3.1 PIRC**

Item	Feature	Subclause	Value/Comment	Status	Support
PIRC	PIRC support	15.1	Protected inter-ring connection sublayer supported	O	Yes [] No []

15.6.3.2 PIRC state machines

Item	Feature	Subclause	Value/Comment	Status	Support
SM1	PircMateMsgReceive	Figure 15. 4.1.5	PircMateMsgReceive state table is supported	PIRC:M	Yes []
SM2	PircControl	Figure 15. 4.2.5	PircControl state table is supported	PIRC:M	Yes []
SM3	PircProtectionUpdate	Figure 15. 4.3.5	PircProtectionUpdate state table is supported	PIRC:M	Yes []
SM4	PircMateMsgTransmit	Figure 15. 4.4.5	PircMateMsgTransmit state table is supported	PIRC:M	Yes []
SM5	PircReceive	Figure 15. 5.1.1.4	PircReceive state table is supported	PIRC:M	Yes []
SM6	PircTransmit	Figure 15. 5.2.1.4	PircTransmit state table is supported	PIRC:M	Yes []

15.6.3.3 Load-balancing

Item	Feature	Subclause	Value/Comment	Status	Support
LB	Load-balancing support	15.1.2	Protected inter-ring connection with load-balancing is supported	PIRC:O	Yes [] No []

Annex A

(informative)

Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

- [B1] ANSI T1.231-1997, Digital Hierarchy—Layer 1 In-Service Digital Transmission Performance Monitoring Standardization.²⁸
- [B2] ANSI T1.503-2002, Network Performance Parameters for Dedicated Digital Services—Definitions and Measurements.
- [B3] ANSI X3.4-1998, American National Standard for Information Systems—Coded Character Sets—7-Bit American National Standard Code for Information Interchange.
- [B4] Bates, R. J., *Broadband Telecommunications Handbook*. New York: McGraw-Hill, 2002.
- [B5] IEEE Std 802.5-1998, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 5: Token ring access method and physical layer.²⁹
- [B6] IETF RFC 854: Telnet Protocol Specification, J. Postel and J. Reynolds, May 1983.³⁰
- [B7] IETF RFC 1903: Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2), J. Case et al., Jan. 1996.
- [B8] IETF RFC 2558: Definitions of Managed Objects for the SONET/SDH Interface Type, K. Tesink, Mar. 1999.
- [B9] IETF RFC 2570: Introduction to Version 3 of the Internet-standard Network Management Framework, J. Case et al., Apr. 1999.
- [B10] IETF RFC 2574: User based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), U. Blumenthal et al., Apr. 1999.
- [B11] IETF RFC 2575: View based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), B. Wijnen et. al., Apr. 1999.
- [B12] IETF RFC 3410: Introduction and Applicability Statements for Internet Standard Management Framework, J. Case et al., Dec. 2002.

²⁸ANSI publications are available from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

²⁹IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854-1441, USA (<http://standards.ieee.org/>).

³⁰IETF publications are available via the World Wide Web at <http://www.ietf.org>.

[B13] ISO/IEC 8802-5:1998E [IEEE Std 802.5-1998E], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 5: Token ring access method and Physical Layer specifications.³¹

[B14] ISO/IEC 9314-2:1989, Information processing systems—Fibre Distributed Data Interface (FDDI)—Part 2: Token Ring Media Access Control (MAC).

[B15] ISO/IEC 15802-1:1995, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Part 1: Medium Access Control (MAC) service definition.

[B16] Sexton, M., and Reid, A., *Broadband Networking: ATM, SDH, and SONET*. Norwood, MA: Artech House, 1998.

[B17] Telcordia Technologies GR-1230-CORE, SONET Bidirectional Line-Switched Ring Equipment Generic Criteria (A Module of TSGR, FR-440), Issue 4, Dec. 1998.

³¹ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iso.ch/>). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (<http://global.ihg.com/>). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

Annex B

(normative)

PacketPHY reconciliation sublayers

B.1 Overview

This annex defines two reconciliation sublayers for use with PacketPHYS. The first is a 1 Gb/s PacketPHY reconciliation sublayer (PRS-1) that maps the logical primitives at the RPR MAC physical layer service interface to the gigabit media independent interface (GMII). The GMII is defined in IEEE Std 802.3-2008. The second is a 10 Gb/s PacketPHY reconciliation sublayer (PRS-10) that maps the logical primitives at the RPR MAC physical layer service interface to the 10 Gb media independent interface (XGMII). Optionally, an XGMII extender sublayer (XGXS) may be used to provide a 10 Gb attachment unit interface (XAUI) instead of the XGMII. The XAUI, XGMII, and XGXS are defined in IEEE Std 802.3-2008.

The relationship of the PRS-1 and PRS-10 to RPR and IEEE 802.3 sublayers is shown in Figure B.1.

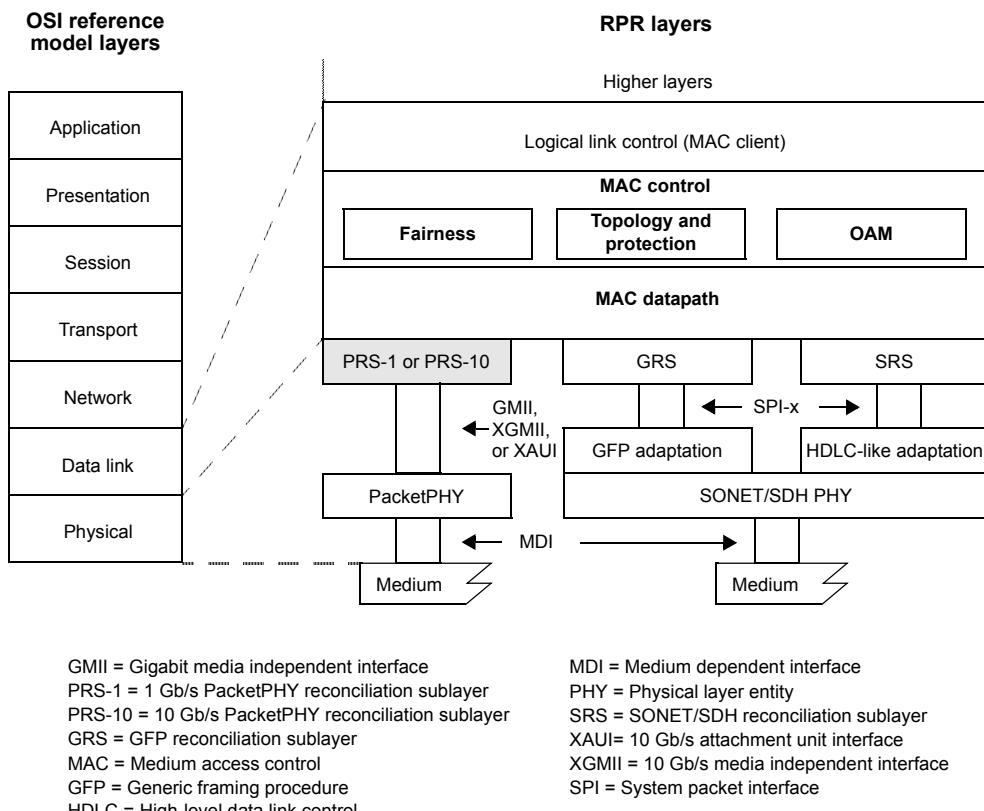


Figure B.1—Reconciliation sublayer relationship to the ISO/IEC OSI reference model

B.2 1 Gb/s PacketPHY reconciliation sublayer (PRS-1)

The PRS-1 converts the logical physical layer service interface of the RPR MAC to the GMII.

B.2.1 General requirements

B.2.1.1 Summary of major concepts

The following major concepts are implemented by the PRS-1:

- a) The PRS-1 maps the signals provided at the GMII to the logical physical layer service interface primitives provided at the MAC.
- b) Each direction of data transfer is independent and serviced by data, delimiter, error, and clock signals.
- c) In the transmit direction, the PRS-1 accepts frames from the MAC, inserts preamble and interpacket gap, and generates corresponding signals on the GMII.
- d) In the receive direction, the PRS-1 receives signals from the GMII, removes preamble and interpacket gap, and conveys frames to the MAC.
- e) In addition to the GMII, link status signals convey a signal fail indication to the PRS-1.

B.2.1.2 Rate of operation

The GMII provided by the PRS-1 is capable of supporting a data rate of 1000 Mb/s. This rate does not correspond to a fixed MAC data rate, because preamble and interpacket gap are generated by the PRS-1.

B.2.1.3 GMII structure

The GMII is composed of independent transmit and receive paths and a management interface controlled by a station management entity. Figure B.2 shows a schematic view of the PRS-1 inputs and outputs.

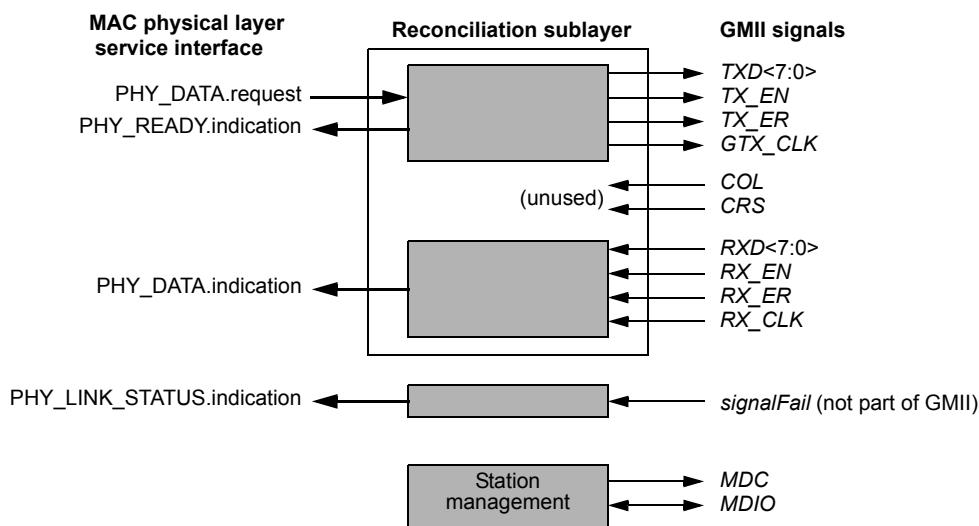


Figure B.2—PRS-1 inputs and outputs

The transmit datapath consists of the following signals:

TXD<7:0>

The transmit data signals with bit ordering as shown in Figure B.3-a.

TX_EN

A transmit enable signal that is asserted when valid data are present on *TXD<7:0>*.

TX_ER

A transmit error signal that propagates errors and controls carrier extension.

GTX_CLK

A continuous transmit clock provided by the reconciliation sublayer.

COL

A collision detect signal that is not used by this standard.

The receive datapath consists of the following signals:

RXD<7:0>

The receive data signals with bit ordering as shown in Figure B.3-b.

RX_DV

A receive data valid signal that is asserted when valid data are available on *RXD<7:0>*.

RX_ER

A receive error signal indicating that an error was detected in the frame currently being transferred by *RXD<7:0>*.

RX_CLK

A continuous receive clock provided by the PHY.

CRS

A carrier sense signal that is not used by this standard.

PHY management signals:

MDIO

A bidirectional (management data input/output) signal used to communicate with the PHY.

MDC

A (management data clock) signal used to synchronize MDIO communications.

The GMII includes *CRS* (carrier sense) and *COL* (collision detect) signals that are not defined for the PRS-1 because only full duplex PHYs are supported. The GMII also includes a management interface consisting of bidirectional data (*MDIO*) and clock (*MDC*) signals. All of the GMII signals are fully defined in IEEE Std 802.3-2008, Clause 35.

B.2.1.4 Receive/transmit bit ordering

Data are transmitted on *TXD<7:0>* and received on *RXD<7:0>* in a least significant through most significant bit order, as shown in Figure B.3-a and Figure B.3-b, respectively.

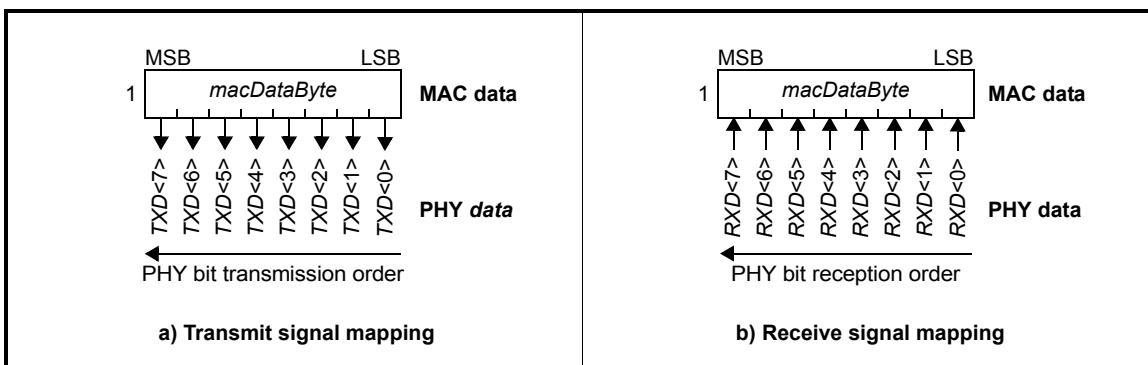


Figure B.3—PRS-1 signal mapping

B.2.1.5 Link status signals

B.2.1.5.1 *signalFail*

In addition to the GMII, a *signalFail* signal is used to convey link status from the PHY to the PRS-1. The PRS-1 shall include a *signalFail* input with the following characteristics:

- a) When asserted, this signal indicates a PHY link failure corresponding to the “link is down” indication by the *Link Status* register bit as defined in IEEE Std 802.3-2008, Clause 22.
- b) When deasserted, this signal indicates that no link failure is detected, corresponding to the “link is up” indication by the *Link Status* register bit as defined in IEEE Std 802.3-2008, Clause 22.

The *signalFail* signal may not be directly available from the PHY.

B.2.1.5.2 *signalDegrade*

The *signalDegrade* signal is not defined for the PRS-1.

B.2.1.6 Mapping of GMII and link status signals to service interface primitives

The PRS-1 shall map the signals provided at the GMII and link status signals to the MAC physical layer service interface primitives defined in Clause 8. Mappings for the following primitives are defined:

- PHY_DATA.request
- PHY_DATA.indication
- PHY_LINK_STATUS.indication
- PHY_READY.indication

B.2.1.6.1 Mapping of PHY_DATA.request

The semantics of the primitives are as follows:

```
PHY_DATA.request
(
    OUTPUT_FRAME,
    LENGTH
)
```

The parameters of the PHY_DATA.request are described below.

OUTPUT_FRAME

Data that are conveyed to the PHY using the signals *TXD<7:0>* when *TX_EN* is asserted. The data signals transition synchronously with *GTX_CLK*. *TX_EN* is deasserted to indicate that valid data are not present. EXTEND and EXTEND_ERROR functions are not supported. *TX_ER* is only asserted to indicate the propagation of a transmit error. The PRS-1 shall prepend a preamble and generate an interpacket gap for each frame as described in B.2.3.1.

LENGTH

Available for reconciliation sublayers to indicate the length of the OUTPUT_FRAME parameter.

(null)—Not used by the PRS-1.

B.2.1.6.2 Mapping of PHY_DATA.indication

The semantics of the primitives are as follows:

```
PHY_DATA.indication
(
    INPUT_FRAME,
    STATUS,
    LENGTH
)
```

The parameters of the PHY_DATA.indication are described below.

INPUT_FRAME

Data that are derived from the signals $RXD<7:0>$ while RX_DV is asserted. Each primitive generated to the MAC sublayer entity corresponds to a PHY_DATA.request issued by the MAC at the other end of the link connecting two RPR stations. The INPUT_FRAME consists of the complete RPR frame beginning after a valid *Start Frame Delimiter*, and continuing until a RX_DV is deasserted. Synchronization between the PRS-1 and the PHY is achieved using the RX_CLK signal. The PRS-1 shall delete the leading preamble and the interpacket gap for each frame as described in B.2.3.2.

STATUS

Indicates whether an error was conveyed by the PHY.

ERROR—Both RX_DV and RX_ER asserted during frame reception.

OK—(Otherwise).

LENGTH

Available for reconciliation sublayers to indicate the length of the INPUT_FRAME parameter.
(null)—Not used by the PRS-1.

B.2.1.6.3 Mapping of PHY_LINK_STATUS.indication

The PHY_LINK_STATUS.indication service primitive is generated by the PRS-1 to indicate PHY link status. The semantics of the primitives are as follows:

```
PHY_LINK_STATUS.indication
(
    LINK_STATUS
)
```

The parameters of the PHY_LINK_STATUS.indication are described below.

LINK_STATUS

An indication of link status.

NO_DEFECT—The link_fault value has the value OK and the signalFail signal is deasserted.

SIGNAL_FAIL—Either of:

link_fault has the values Local Fault or Remote Fault, or *signalFail* is asserted.

SIGNAL_DEGRADE—Not generated by the PRS-1.

DEGRADE_FAIL—Not generated by the PRS-1.

NOTE—A SIGNAL_DEGRADE indication is never generated by the PRS-1.

B.2.1.6.4 Mapping of PHY_READY.indication

The semantics of the primitives are as follows:

```
PHY_READY.indication
(
    READY_STATUS
)
```

The parameters of the PHY_READY.indication are described below.

READY_STATUS

An indication of whether the RS is in the process of transmitting a preamble.

NOT_READY—the RS is in the process of transmitting a preamble, a frame, or the IPG.

READY—(otherwise).

B.2.2 GMII data stream

Data frames transmitted through the GMII are transferred within the GMII data stream. The data stream is a combination of data bytes and control signals.

The GMII datastream shall meet the requirements of 35.2.3 of IEEE Std 802.3-2008, with the exception of 35.2.3.1 and 35.2.3.5. Carrier extension is not defined for the PRS-1.

B.2.3 GMII functional specifications

B.2.3.1 Transmit

B.2.3.1.1 General requirements

The PRS-1 shall meet the transmit functional requirements specified in 35.2.2.1, 35.2.2.3, 35.2.2.4, and 35.2.2.5 of IEEE Std 802.3-2008.

B.2.3.1.2 Preamble

The RPR MAC frame does not contain a preamble. The PRS-1 shall prepend a preamble and start frame delimiter (SFD) to each frame transmitted on the GMII in accordance with 35.2.3.2.1 of IEEE Std 802.3-2008.

The reconciliation sublayer requests the MAC to pause between the transmission of frames to allow insertion of the preamble using the PHY_READY.indication primitive.

B.2.3.1.3 Interpacket gap

The RPR MAC does not transmit interpacket gap or *Idle* characters. The PRS-1 shall insert a minimum interpacket gap in accordance with IEEE Std 802.3-2008, Clause 4 for operation at 1000 Mb/s.

The reconciliation sublayer requests the MAC to pause between the transmission of frames to allow insertion of the interpacket gap using the PHY_READY.indication primitive.

B.2.3.2 Receive

The PRS-1 shall meet the receive functional requirements specified in 35.2.2.2, 35.2.2.6, 35.2.2.7, and 35.2.2.8 of IEEE Std 802.3-2008.

B.2.3.3 Error and fault handling

The PRS-1 shall meet the error and fault handling functional requirements specified in 35.2.1.5 and 35.2.1.6 of IEEE Std 802.3-2008.

B.2.3.4 CRS (carrier sense) and COL (collision) signals

Because the PRS-1 supports only full duplex PHYs, the *CRS* (carrier sense) and *COL* (collision) signals shall be ignored.

B.2.4 Electrical characteristics

B.2.4.1 GMII

The GMII provided by the PRS-1 shall meet the electrical requirements of 35.4 of IEEE Std 802.3-2008.

B.2.4.2 Link status signals

The link status signals shall meet the electrical characteristics specified in IEEE Std 802.3-2008, Clause 35, for the GMII *MDIO* and *MDC* signals.

B.3 10 Gb/s PacketPHY Reconciliation Sublayer (PRS-10)

B.3.1 General requirements

The PRS-10 converts the logical physical layer service interface of the RPR MAC to the XGMII. Optionally, an XGXS may be used to provide a XAUI instead of the XGMII interface. Although the XGMII is an optional interface, it is used as a basis for specification of the PRS-10, and an implementation using the XGXS to provide a XAUI shall behave functionally as if the XGMII were implemented.

B.3.1.1 Summary of major concepts

- a) The PRS-10 maps the signal set provided at the XGMII to the physical layer service interface primitives provided at the MAC.
- b) Each direction of data transfer is independent and serviced by data, control, and clock signals.
- c) In the transmit direction, the PRS-10 accepts frames from the MAC, inserts Start control characters, preamble, Terminate control characters, and Idle control characters to generate continuous data or control characters on the XGMII.
- d) In the receive direction, the PRS-10 expects continuous data or control characters from the XGMII, removes Start control characters, preamble, Terminate control characters, and Idle control characters and conveys frames to the MAC.
- e) The PRS-10 participates in link fault detection and reporting.
- f) The PRS-10 may use an XGXS to provide a XAUI instead of the XGMII.

B.3.1.2 Relationship to other sublayers

The relationship of the PRS-10 to RPR and IEEE 802.3 sublayers is shown in Figure B.1.

B.3.1.3 Rate of operation

The XGMII provided by the PRS-10 transfers data or control characters at a fixed rate of 10.0 Gb/s. This rate does not correspond to a fixed MAC data rate, because preamble and interpacket gap are generated by the PRS-10.

B.3.1.4 XGMII structure

The XGMII is composed of independent transmit and receive paths. Figure B.4 shows a schematic view of the PRS-10 inputs and outputs.

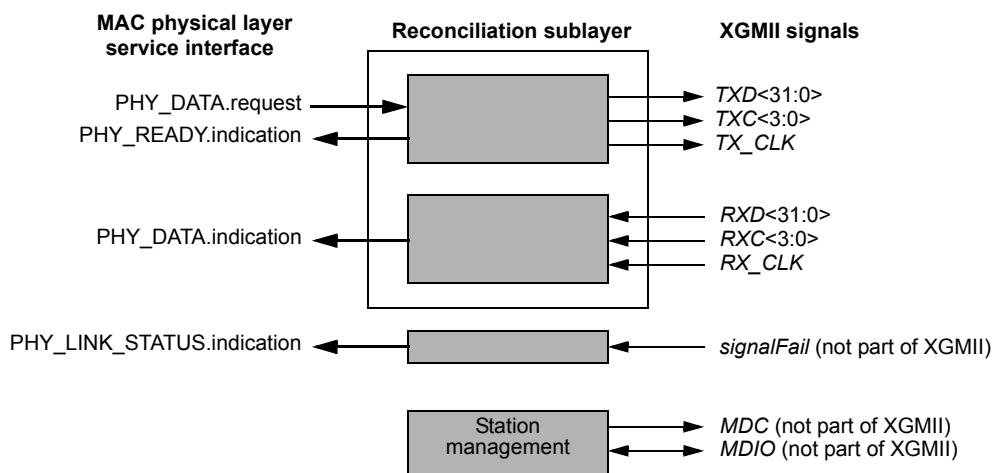


Figure B.4—PRS-10 inputs and outputs

The transmit datapath consists of the following signals:

TXD<31:0>

A group of four bytes that transmit data or control characters with the bit ordering shown in Figure B.5-a.

TXC<3:0>

Control signals that identify each of the *TXD<31:0>* bytes as data or control characters. *TXC<3:0>* are asserted to indicate that the corresponding bytes of *TXD<31:0>* are control characters. *TXC<3:0>* are de-asserted to indicate that the corresponding bytes of *TXD<31:0>* are data characters.

TX_CLK

A continuous transmit clock provided by the reconciliation sublayer. The *TXD<31:0>* and *TXC<3:0>* signals are conveyed from the RS to the PHY on each edge of *TX_CLK*.

The receive datapath consists of the following signals:

RXD<31:0>

A group of four bytes that receive data or control characters, with the bit ordering shown in Figure B.5-b.

RXC<3:0>

Control signals that identify each of the *RXD<31:0>* bytes as data or control characters. *RXC<3:0>* are asserted to indicate that the corresponding bytes of *RXD<31:0>* are control characters. *RXC<3:0>* are de-asserted to indicate that the corresponding bytes of *RXD<31:0>* are data characters.

RX_CLK

Continuous receive clock provided by the PHY. The *RXD<31:0>* and *RXC<3:0>* signals are conveyed from the PHY to the RS on each edge of *RX_CLK*.

The MDIO and MDC signals are described in B.2.1.3.

B.3.1.5 Receive/transmit bit ordering

Data are transmitted on *TXD<7:0>* and received on *RXD<7:0>* in a least significant through most significant bit order, as specified in Figure B.5-a and Figure B.5-b, respectively.

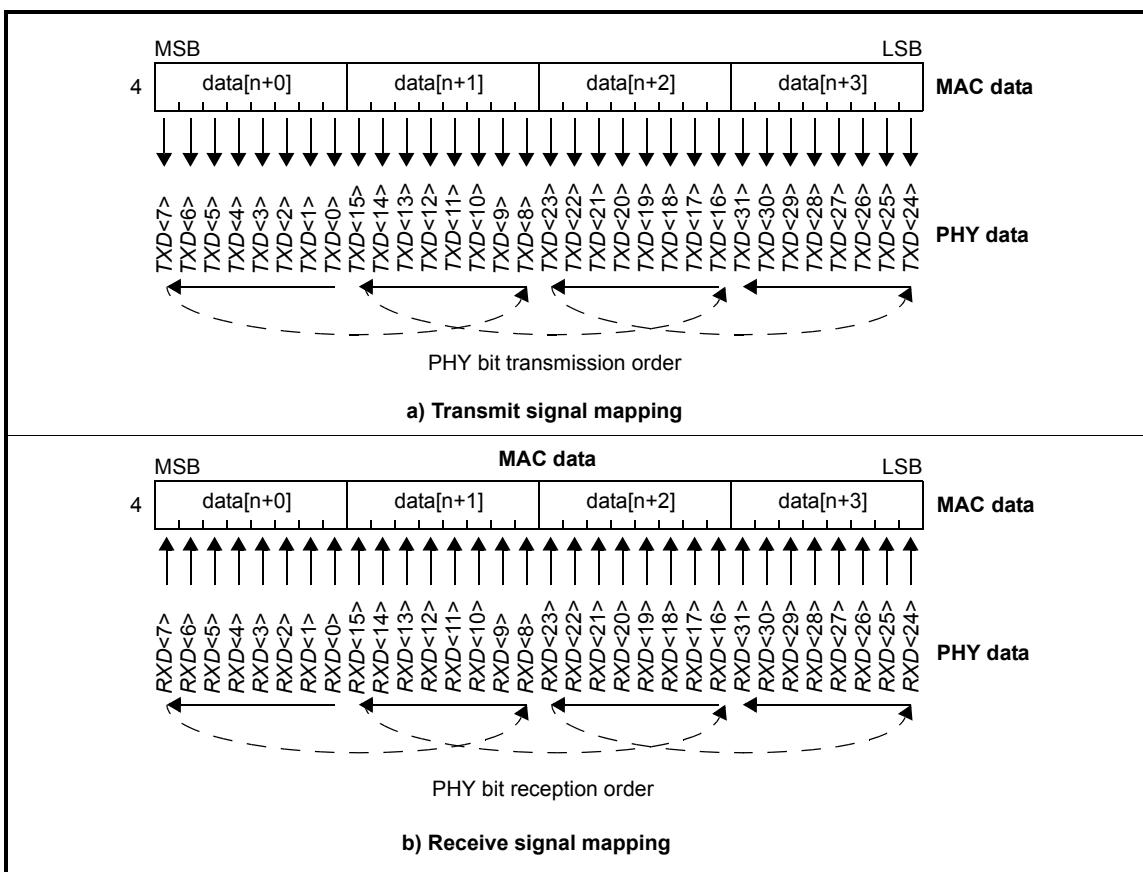


Figure B.5—PRS-10 signal mapping

The 32 data and 4 control signals in each direction shall be organized into four lanes, as shown in Table B.1. The four lanes in each direction share a common clock—*TX_CLK* for transmit and *RX_CLK* for receive. The four lanes are used in round-robin sequence to carry the datastream. The first byte is aligned to lane 0, the second to lane 1, the third to lane 2, the fourth to lane 3, and then repeating with the fifth to lane 0. Delimiters and interpacket gap characters are encoded on the *TXD* and *RXD* signals with the control code indicated by assertion of *TXC* and *RXC*, respectively.

The XGMII signals are fully defined in IEEE Std 802.3-2008, Clause 46, Link status signals.

Table B.1—Transmit and receive lane associations

Transmit		Receive		Lane
Data	Control	Data	Control	
<i>TXD<7:0></i>	<i>TXC<0></i>	<i>RXD<7:0></i>	<i>RXC<0></i>	0
<i>TXD<15:8></i>	<i>TXC<1></i>	<i>RXD<15:8></i>	<i>RXC<1></i>	1
<i>TXD<23:16></i>	<i>TXC<2></i>	<i>RXD<23:16></i>	<i>RXC<2></i>	2
<i>TXD<31:24></i>	<i>TXC<3></i>	<i>RXD<31:24></i>	<i>RXC<3></i>	3

B.3.1.5.1 *signalFail*

In addition to the XGMII, a *signalFail* signal is used to convey link status from the PHY to the PRS-10. The PRS-10 shall include a *signalFail* input with the following characteristics:

- a) When asserted, this signal indicates a PHY link failure corresponding to the “link is down” indication by the *Link Status* register bit as defined in IEEE Std 802.3-2008, Clause 45.2.1.2.
- b) When deasserted, this signal indicates that no link failure is detected, corresponding to the “link is up” indication by the *Link Status* register bit as defined in IEEE Std 802.3-2008, Clause 45.2.1.2.

The *signalFail* signal may not be directly available from the PHY. An RPR system implementation may use the XGMII link fault signaling to detect link status, or it may generate the *signalFail* signal from the MDIO or from optional signals that may be provided by the PHY.

B.3.1.5.2 *signalDegrade*

The *signalDegrade* signal is not defined for the PRS-10.

B.3.1.6 Mapping of XGMII signals to MAC physical layer service interface primitives

The PRS-10 shall map the signals provided at the XGMII to the MAC physical layer service interface primitives defined in Clause 8. Mappings for the following primitives are defined:

- *PHY_DATA.request*
- *PHY_DATA.indication*
- *PHY_LINK_STATUS.indication*
- *PHY_READY.indication*

B.3.1.6.1 Mapping of *PHY_DATA.request*

The semantics of the primitive are as follows:

```
PHY_DATA.request
(
    OUTPUT_FRAME,
    LENGTH
)
```

The parameters of the PHY_DATA.request are as follows:

OUTPUT_FRAME

Data conveyed to the PHY by the signals $TXD<31:0>$ and $TXC<3:0>$ on each TX_CLK edge. A frame transferred by PHY_DATA.request transaction shall be mapped to TxD signals in sequence ($TXD<0:7>$, ... $TXD<24:31>$, $TXD<0:7>$). The PRS-10 shall prepend a *Start* control character, preamble, and *Start Frame Delimiter* to each frame as described in B.3.3.1.2, and append a *Terminate* control character and *Idle* control characters to each frame as described in B.3.3.1.3.

LENGTH

Available for reconciliation sublayers to indicate the length of the INPUT_FRAME parameter.
(null)—Not used by the PRS-10.

B.3.1.6.2 Mapping of PHY_DATA.indication

The semantics of the primitives are as follows:

PHY_DATA.indication

```
(  
    INPUT_FRAME,  
    STATUS,  
    LENGTH  
)
```

The parameters of the PHY_DATA.indication are described below.

INPUT_FRAME

Data derived from the signals $RXD<31:0>$ and $RXC<3:0>$ received from the PHY on each edge of RX_CLK. Each primitive generated to the MAC sublayer entity corresponds to a PHY_DATA.request issued by the MAC at the other end of the link connecting two RPR stations. The INPUT_FRAME consists of the complete RPR frame beginning eight bytes following a valid *Start* control character, and continuing until a *Terminate* control character. The *Start* control character, preamble, *Start Frame Delimiter*, *Terminate* control character, and *Idle* control characters are not part of an RPR frame, and they are removed prior to transfer of a frame to the MAC.

STATUS

A null parameter not used by the PRS-10.

LENGTH

A null parameter not used by the PRS-10.

To assure robust operation, the value of the data transferred to the MAC may be changed by the PRS-10 as required by XGMII error indications (see B.3.3.3). Sequence ordered_sets are not indicated to the MAC (see B.3.3.3).

B.3.1.6.3 Mapping of PHY_LINK_STATUS.indication

The semantics of the primitive are as follows:

PHY_LINK_STATUS.indication

```
(  
    LINK_STATUS  
)
```

The parameters of the PHY_LINK_STATUS.indication are as follows:

LINK_STATUS

An indication of link status.

NO_DEFECT—*link_fault* has the value OK and the *signalFail* signal is deasserted.

SIGNAL_FAIL—Either of:

- link_fault* has the local fault condition, or
- signalFail* is asserted.

SIGNAL_DEGRADE—Not generated by the PRS-10.

DEGRADE_FAIL—Not generated by the PRS-10.

The PHY_LINK_STATUS.indication service primitive shall be generated by the PRS-10 whenever there is a change in the *link_fault* variable defined in IEEE Std 802.3-2008, Clause 46.3.4, or any time there is a change in the *signalFail* signal described in B.2.1.5.1.

B.3.1.6.4 Mapping of PHY_READY.indication

The semantics of the primitive are as follows:

```
PHY_READY.indication
(
    READY_STATUS
)
```

The parameters of the PHY_READY.indication are described below.

READY_STATUS

An indication of when the PRS-10 is ready to accept another frame.

NOT_READY—the PRS-10 is transmitting a preamble, or a frame, or the IPG.

READY—(otherwise).

B.3.2 XGMII data stream

Data frames transmitted through the XGMII are transferred within the XGMII data stream. The data stream is a sequence of bytes, where each byte conveys either a data or control character.

The XGMII datastream shall meet the requirements of 46.2 of IEEE Std 802.3-2008, with the exception of references to the MAC interpacket gap in 46.2.1. The interpacket gap for the RPR PRS-10 is defined in B.3.3.1.3.

B.3.3 Functional specifications

B.3.3.1 Transmit

B.3.3.1.1 General requirements

The PRS-10 shall meet the transmit functional requirements specified in 46.3.1.1 through 46.3.1.3 of IEEE Std 802.3-2008.

B.3.3.1.2 Preamble

The MAC RPR frame does not contain a preamble. The PRS-10 shall prepend a preamble including a Start control character, and it will prepend a Start Frame Delimiter to each frame transmitted on the XGMII in accordance with 46.2.2 of IEEE Std 802.3-2008.

The reconciliation sublayer requests the MAC to pause between the transmission of frames to allow insertion of the preamble using the PHY_READY.indication primitive.

B.3.3.1.3 Interpacket gap

The RPR MAC does not transmit interpacket gap or Idle characters. The PRS-10 shall insert a minimum interpacket gap including the *Terminate* control character in accordance with IEEE Std 802.3-2008, Clause 4 and 46.3.1.4.

The reconciliation sublayer requests the MAC to pause between the transmission of frames to allow insertion of the interpacket gap using the PHY_READY.indication primitive.

B.3.3.2 Receive

The PRS-10 shall meet the receive functional requirements specified in 46.3.2 of IEEE Std 802.3-2008.

B.3.3.3 Error and fault handling

The PRS-10 shall meet the error and fault handling functional requirements specified in 46.3.3 of IEEE Std 802.3-2008.

B.3.4 Electrical characteristics

The XGMII provided by the PRS-10 shall meet the electrical requirements specified in 46.4 of IEEE Std 802.3-2008.

B.3.5 XGXS and XAUI

The XGXS may be implemented to provide a XAUI instead of the XGMII interface.

If implemented, the XGXS and XAUI shall meet all requirements specified in Clause 47 of IEEE Std 802.3-2008.

B.4 Protocol Implementation Conformance Statement (PICS) proforma for Annex B³²

B.4.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Annex B, PacketPHY reconciliation sublayers, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

B.4.2 Identification

B.4.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

B.4.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, PacketPHY reconciliation sublayers
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

³²Copyright release for PICS proformas: Users of this standard may freely reproduce the PICS proforma in this annex so that it can be used for its intended purpose and may further publish the completed PICS.

B.4.3 Major capabilities/options

Item	Feature	Subclause	Value/Comment	Status	Support
C1*	PRS-1	B.2	1 Gb/s PacketPHY	O	Yes [] No[]
C2*	PRS-10	B.3	10 Gb/s PacketPHY	O	Yes [] No[]

B.4.4 PICS tables for Annex B

B.4.4.1 PRS-1

Item	Feature	Subclause	Value/Comment	Status	Support
G1	Signal fail	B.2.1.5.1	Provide a <i>signalFail</i> input	C1:M	Yes []
G2	Service interface	B.2.1.6	Map GMII to service interface primitives	C1:M	Yes []
G3	Preamble and interpacket gap	B.2.1.6.1	Add a preamble and generate interpacket gap	C1:M	Yes []
G4	GMII data stream	B.2.2	Generate a GMII-compliant data stream	C1:M	Yes []
G5	Transmit	B.2.3.1.1	Provide 802.3-compatible transmit functionality	C1:M	Yes []
G6	Preamble	B.2.3.1.2	Add a preamble to RPR frames	C1:M	Yes []
G7	Interpacket gap	B.2.3.1.3	Generate an interpacket gap between RPR frames	C1:M	Yes []
G8	Receive	B.2.3.2	Provide 802.3-compatible receive functionality	C1:M	Yes []
G9	Error and fault handling	B.2.3.3	Provide 802.3-compatible error and fault handling	C1:M	Yes []
G10	CRS and COL signals	B.2.3.4	Ignore CRS and COL signals	C1:M	Yes []
G11	GMII electrical requirements	B.2.4.1	Provide 802.3-compatible GMII electrical characteristics	C1:M	Yes []
G12	Link status electrical requirements	B.2.4.2	Provide 802.3-compatible link status signal electrical characteristics	C1:M	Yes []

B.4.4.2 PRS-10

Item	Feature	Subclause	Value/Comment	Status	Support
X1	XGMII functionality	B.3.1	An implementation without an XGMII must behave functionally as if the XGMII was implemented	C2:M	Yes []
X2	XGMII organization	B.3.1.4	XGMII TX and RX each organized as 4 lanes	C2:M	Yes []
X3	Signal fail	B.3.1.5.1	Provide a <i>signalFail</i> input	C2:M	Yes []
X4	Service interface	B.3.1.6	Map XGMII to service interface primitives	C2:M	Yes []
X5	TX signal mapping	B.3.1.6.1	Map a TX frame to TX signals	C2:M	Yes []
X6	Preamble and interpacket gap	B.3.1.6.1	Add a preamble and generate interpacket gap	C2:M	Yes []
X7	Link status mapping	B.3.1.6.3	Map <i>signalFail</i> to the link status primitive	C2:M	Yes []
X8	XGMII data stream	B.3.2	Generate an XGMII-compliant data stream	C2:M	Yes []
X9	Transmit	B.3.3.1.1	Provide 802.3-compatible transmit functionality	C2:M	Yes []
X10	Preamble	B.3.3.1.2	Add a preamble to RPR frames	C2:M	Yes []
X11	Interpacket gap	B.3.3.1.3	Generate an interpacket gap between RPR frames	C2:M	Yes []
X12	Receive	B.3.3.2	Provide 802.3-compatible receive functionality	C2:M	Yes []
X13	Error and fault handling	B.3.3.3	Provide 802.3-compatible error and fault handling	C2:M	Yes []
X14	Electrical requirements	B.3.4	Provide 802.3-compatible electrical characteristics	C2:M	Yes []
X15	XAUI	B.3.5	Specification compliant to XAUI	C2:O	Yes [] No []

Annex C

(normative)

SONET/SDH reconciliation sublayers

C.1 Overview

This annex defines two families of reconciliation sublayers for use with SONET/SDH physical layer entities (PHYs). The first is the SONET/SDH reconciliation sublayer (SRS), which maps the logical primitives at the RPR MAC physical layer service interface to Optical Interconnect Forum (OIF) implementation agreements: 8-bit and 32-bit OIF System Packet Interface Level 3 (SPI-3), OIF System Packet Interface Level 4 Phase 1 (SPI-4.1), and OIF System Packet Interface Level 4 Phase 2 (SPI-4.2) interfaces. The second is the GFP reconciliation sublayer (GRS), which also maps the logical primitives at the RPR MAC physical layer service interface to the same system packet interface (SPI) interfaces. The SRS is intended for use with GFP without core header insert or for byte-synchronous HDLC-like framing adaptation sublayers. The GRS is intended for use only with a GFP adaptation sublayer with core header insert. The SRS and GRS are identical, except that the GRS conveys frame length information to the GFP adaptation sublayer, thus eliminating the need to calculate this parameter in the PHY device and incur a store and forward delay.

C.1.1 Relationship to other sublayers

The relationship of the SRS and GRS to RPR sublayers and to SONET/SDH physical layers is shown in Figure C.1.

The expected functions of the GFP and HDLC-like adaptation sublayer in the PHY device are to perform idle insertion in case no frame is to be sent on the medium in the transmit direction and idle deletion and frame delineation in the receive direction. Across the SPI interface, only delineated frames are transmitted.

C.1.2 SRS and GRS interfaces

This annex describes four SRS and GRS implementations using the following electrical interfaces specified by the Optical Interworking Forum (OIF):

- a) SPI Level 3 (SPI-3) with 8-bit transmit and receive data paths operating at 155 Mb/s to 622 Mb/s
- b) SPI Level 3 (SPI-3) with 32-bit transmit and receive data paths operating at 155 Mb/s to 2.5 Gb/s
- c) SPI Level 4 Phase 1 (SPI-4.1) operating at 200 Mb/s to 10 Gb/s
- d) SPI Level 4 Phase 2 (SPI-4.2) operating at 622 Mb/s to 10 Gb/s

The SRS and GRS for each of the interfaces are electrically identical, except that the GRS conveys frame length information. The SRS does not convey frame length information.

The SPI interfaces support multi-PHY devices. In the context of a single MAC, each MAC datapath entity is connected to one instance of a physical device. Multi-PHY support is discussed in this clause and can be supported if the SPI interface can handle the aggregate bandwidth of the attached ringlets.

Figure C.2 shows the designation of the SPI interface with each ringlet connected through a separate SPI interface to its matching MAC datapath entity.

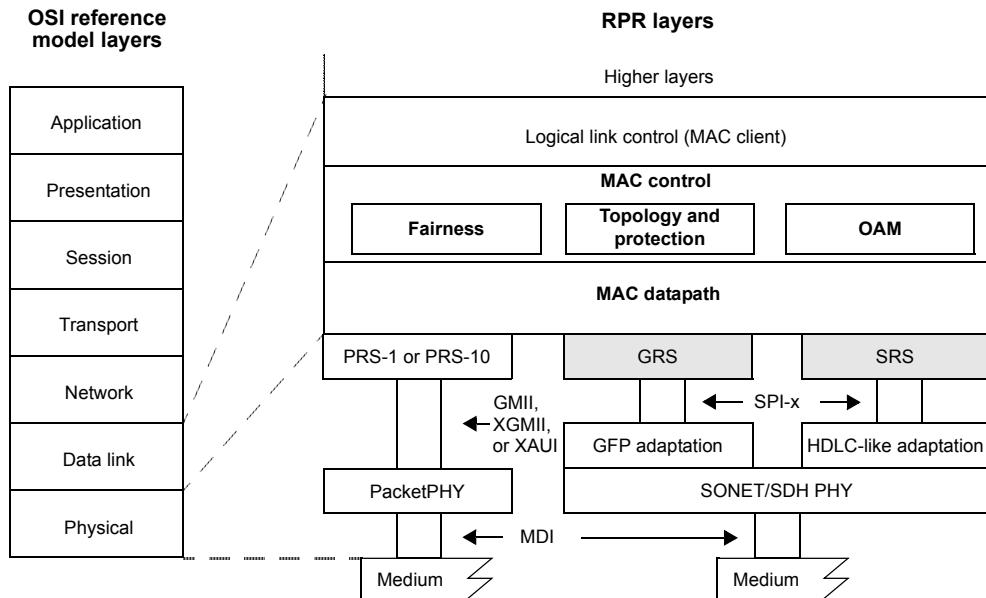


Figure C.1—Reconciliation sublayer relationship to the ISO/IEC OSI reference model

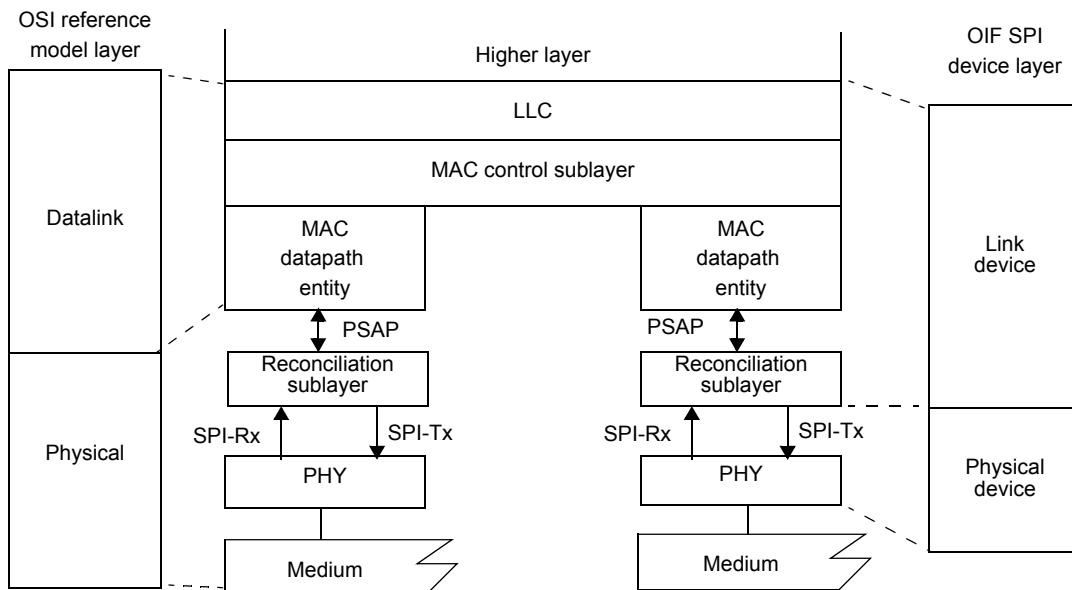


Figure C.2—OSI and OIF reference model layers

C.1.3 Link status signals

In addition to the SPI-3 interface, *signalFail* and *signalDegradate* signals are used to convey link status from the PHY to the reconciliation sublayer.

The *signalFail* and *signalDegradate* signals may not be directly available from the PHY. The reconciliation sublayer provides the physical interface and maps these signals to physical layer interface primitives, i.e., `PHY_LINK_STATUS.indication`. A MAC implementation is required to generate these signals based on the specific PHY implementation.

C.1.3.1 *signalFail*

The reconciliation sublayer shall include a *signalFail* input. An asserted *signalFail* indicates a PHY signal fail condition. Depending on the medium and the medium-dependent transmission method, signal fail may indicate a *Loss of Signal* or a *Loss of Frame*.

NOTE—These are only some examples of the attributes of *signalFail*; this list is not comprehensive.

C.1.3.2 *signalDegradate*

The reconciliation sublayer shall include a *signalDegradate* input. An asserted *signalDegradate* indicates a PHY signal degrade condition. If the PHY does not provide a *signalDegradate* status, then the DEGRADE status is not reported.

C.1.3.3 Mapping of link status signals to service interface primitives

The reconciliation sublayer shall map the signals provided at the physical interface to the MAC physical layer service interface primitives defined in Clause 8. Mappings for the following primitives are defined:

```
PHY_LINK_STATUS.indication
(
    LINK_STATUS
)
```

There are two physical pins that receive the PHY link status if provided by the PHY. These two physical signals are *signalFail* and *signalDegradate*.

The two physical link status inputs map the signals at these pins to the `LINK_STATUS` attribute, as illustrated in Table C.1.

Table C.1—*signalFail* and *signalDegradate* mapping

<i>signalFail</i>	<i>signalDegradate</i>	<code>LINK_STATUS</code>
inactive	inactive	NO_DEFECT
inactive	active	SIGNAL_DEGRADE
active	inactive	SIGNAL_FAIL
active	active	DEGRADE_FAIL

C.1.3.3.1 When generated

This primitive is invoked by the reconciliation sublayer after it determines that the PHY is not performing properly. For example, assume that the received RFCLK has failed. This problem would cause the SPI receive interface to not be able to receive data. Note that `LINK_STATUS==FAIL` is also generated upon reception of signals from the PHY device, which it has generated to indicate that the local physical sublayer has failed.

This signal is indicated as `signalFail` and `signalDegrade` in Figure C.7.

C.1.3.3.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.1.4 Electrical specifications

The `signalDegrade` and `signalFail` inputs shall be 3.3V LVTTL interfaces in accordance with the interface requirements specified in EIA/JEDEC JESD8-B.

C.2 Physical frame format for SRS and GRS

The physical electrical interface for the GRS and the SRS is identical for both data and control signals. The difference between the GFP and the HDLC like-adaptation layers is the delineation mechanism and hence different frame formats, specifically the header prefix.

C.2.1 SRS physical frame format

The HDLC-like adaptation sublayer performs byte-stuffing for flag and escape characters contained within the RPR frame and performs encapsulation within the HDLC-like frame structure as defined in 8.4.2.2. LAPS also performs byte-stuffing but performs encapsulation as defined in 8.4.2.3.

In the HDLC-like adaptation interface, this sublayer inserts the start of frame flag, $7E_{16}$, and at least one escape flag to mark the end of frame. For consecutive frame transmissions, the minimum requirement is one escape flag to separate the two frames.

The HDLC-like adaptation sublayer thus has to perform escaping for any $7E_{16}$ occurrence in the frame. The frame transmitted across this interface is the RPR frame as specified in Clause 9.

As shown in Figure C.3, the generation of the CRC and the frame transmission are in opposite bit order, and the effectiveness of the CRC is diminished. However, the CRC still ensures coverage of any 2-byte error in the frame.

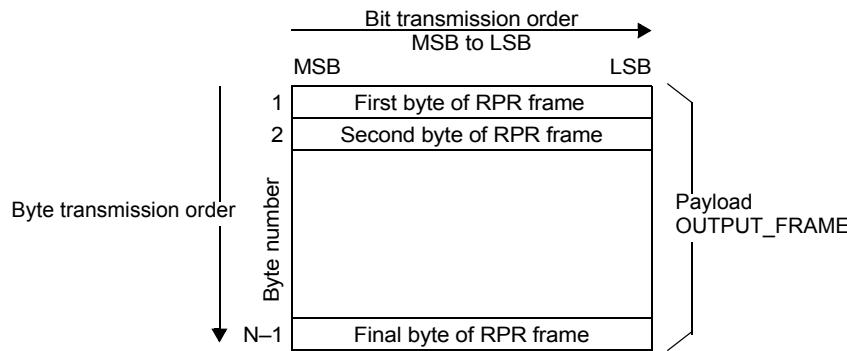


Figure C.3—SRS frame structure

C.2.2 GRS physical frame format

The GFP delineation method provides a deterministic inflation factor that is not frame content sensitive. It uses a length value protected by a *cHEC* to provide robust delineation under degrad channel condition. The HEC provides single error correction. In the case of GRS interface, the length and the HEC fields are part of the frame transmitted across the physical interface. The frame format is shown in Figure C.4.

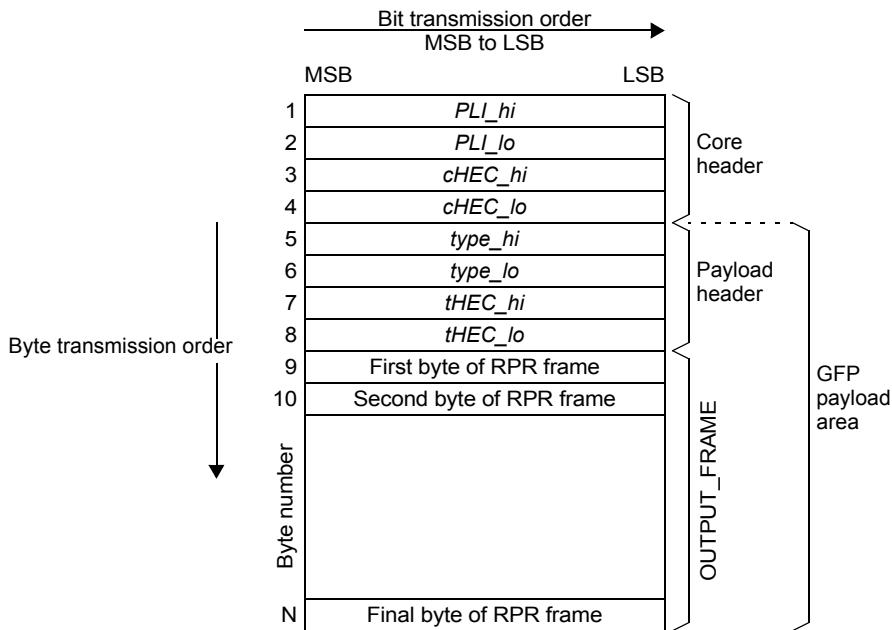


Figure C.4—GRS frame structure

As shown in Figure C.4, the generation of the RPR frame FCS and the frame transmission are in opposite bit order, and the effectiveness of the CRC is diminished. However, the CRC still ensures coverage of any 2-byte error in the frame.

The length field identifies the number of bytes from the RPR header to the end of CRC-32, inclusive.

The calculation of the *cHEC* is optional as the GFP adaptation sublayer can insert the actual *cHEC* value. If the GRS does not calculate the *cHEC*, then it shall insert zeros.

The type field is constructed in the reconciliation sublayer and prepended to the OUTPUT_FRAME.

When operating in the GRS mode, if the length primitive is not received across the PSAP interface, i.e., a (null) value is received across the logical interface, then the GRS shall generate the required GFP core header field. Alternatively, if the SRS is used, then the GFP adaption sublayer generates the required GFP headers.

The calculation of the HEC uses the polynomial of Equation (C.1).

$$\text{CRC16} = x^{16} + x^{12} + x^5 + 1 \quad (\text{C.1})$$

Depending on the field the HEC protects over, it is called *cHEC* or *tHEC*.

C.2.2.1 Payload length identifier

The GFP core header consists of two fields that support frame delineation: *PLI* (payload length indicator) and *cHEC*, as illustrated in Figure C.5.

The core header supports frame delineation. The core header consists of two fields: a payload length indicator and a *cHEC*. The 16-bit *PLI* value specifies the number of bytes in the payload area. The *cHEC* is calculated over the *PLI* only.

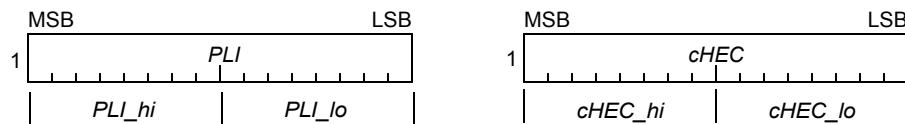


Figure C.5—GFP core header

C.2.2.2 GFP payload header

The basic GFP payload header consists of two fields: type and *tHEC*. For RPR applications, the non-extension-type payload header format is used. The *tHEC* is calculated over the type field only.

The GFP type field consists of the following subfields: (1) a 3-bit *PTI* (payload type identifier) field, (2) a *PFI* (payload FCS indicator) bit, (3) a 4-bit *EXI* (extension header identifier) field, and (4) an 8-bit *UPI* (user payload identifier) field, as illustrated in Figure C.6.

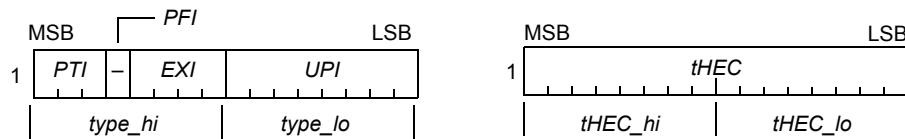


Figure C.6—GFP payload header

The definitions of the values are provided in ITU-T G.7041. The *UPI* value is listed in ITU-T G.7041 Amendment 2. The recommended values for the above fields are specified in Table C.2.

NOTE—The assignment of this number is by ITU-T. Refer to ITU-T G.7041.

Table C.2—GFP payload header field values

Field	Recommended value	Comments
<i>PTI</i>	000_2	Client data
<i>PFI</i>	0	Absent of GFP FCS
<i>EXI</i>	0000_2	Null extension header
<i>UPI</i>	$0000\ 1010_2$	RPR user payload type

C.3 SRS and GRS using the 8-bit SPI-3 interface

The SRS and GRS may be implemented with an 8-bit OIF SPI-3 interface.

C.3.1 General requirements

The following subclauses provide a brief description of the signals for the SPI-3 interface and their interaction with the MAC physical layer service interface primitive.

In case of any discrepancy in the signal description, the SPI-3 specification takes precedence.

C.3.1.1 Summary of major concepts

The SRS and GRS using the 8-bit SPI-3 interface have the following characteristics:

- a) The SRS and GRS map the signals provided at the 8-bit SPI-3 interface to the logical physical layer service interface primitives provided at the MAC.
- b) Each direction of data transfer is independent, and serviced by 8-bit data, delimiter, error, and clock signals.
- c) Data and delimiters are synchronous to clock references.
- d) The SPI-3 interface supports logical channels with individual byte-level or frame-level out-of-band flow control.
- e) The SRS and GRS using the 8-bit SPI-3 interface support full-duplex operation only.

C.3.1.2 Rate of operation

The SRS and GRS using the 8-bit SPI-3 interface are capable of supporting data rates of 155 Mb/s to 622 Mb/s, limited to an effective operation granularity of 155 Mb/s. This is not an interface limit but the result of physical coding which maps to increments of STS-1 or STS-3 SONET rates and increments of STM-1 for SDH rates.

SONET/SDH PHYs that provide an SPI-3 with an 8-bit datapath shall support operations, at the SONET/SDH Path level, at the selected rate on the SPI-3. PHYs reports the rates at which they are operating via the management interface. The PHY is expected to operate in either streaming, flow-through, or cut-through modes.

C.3.1.3 8-bit SPI-3 structure

The 8-bit SPI-3 interface is composed of independent transmit and receive paths. The MAC behaves as a link layer device.

SPI-3 interfaces can operate in byte-level or packet-level mode. For byte-level transfers, the FIFO status information is presented on a cycle-by-cycle basis. A PHY device indicates the transmit frame available status via the selected or directed method signals, *STPA* and *DTPA*[], respectively, for the multi-PHY case. *DTPA* signals are provided for simpler implementation to reduce the need for addressing. Support for one or the other is optional, but at least one is to be supported to ensure the PHY FIFO does not overrun.

For packet-level transfers, the FIFO status information applies to segments of frame data. The reconciliation sublayer polls the status of the PHY for transmit ready status. The reconciliation sublayer polls one of multiple available PHYs by asserting the TADR address of the PHY, which responds via a common *PTPA* signal.

Additional signals *TENB*, *RENB*, and *RVAL* are used by the protocol to indicate valid data on the bus.

SPI-3 in each direction uses eight data signals (*TDAT* [7:0] and *RDAT* [7:0]), clocks (*TFCLK* and *RFCLK*), start of frame delimiter signals (*TSOP* and *RSOP*), end of frame delimiter signals (*TEOP* and *REOP*), and other control signals described in C.3.2. These signals are fully defined in the SPI-3 implementation agreement document.

Figure C.7 shows a schematic view of the SRS and GRS inputs and outputs using the 8-bit SPI-3 interface.

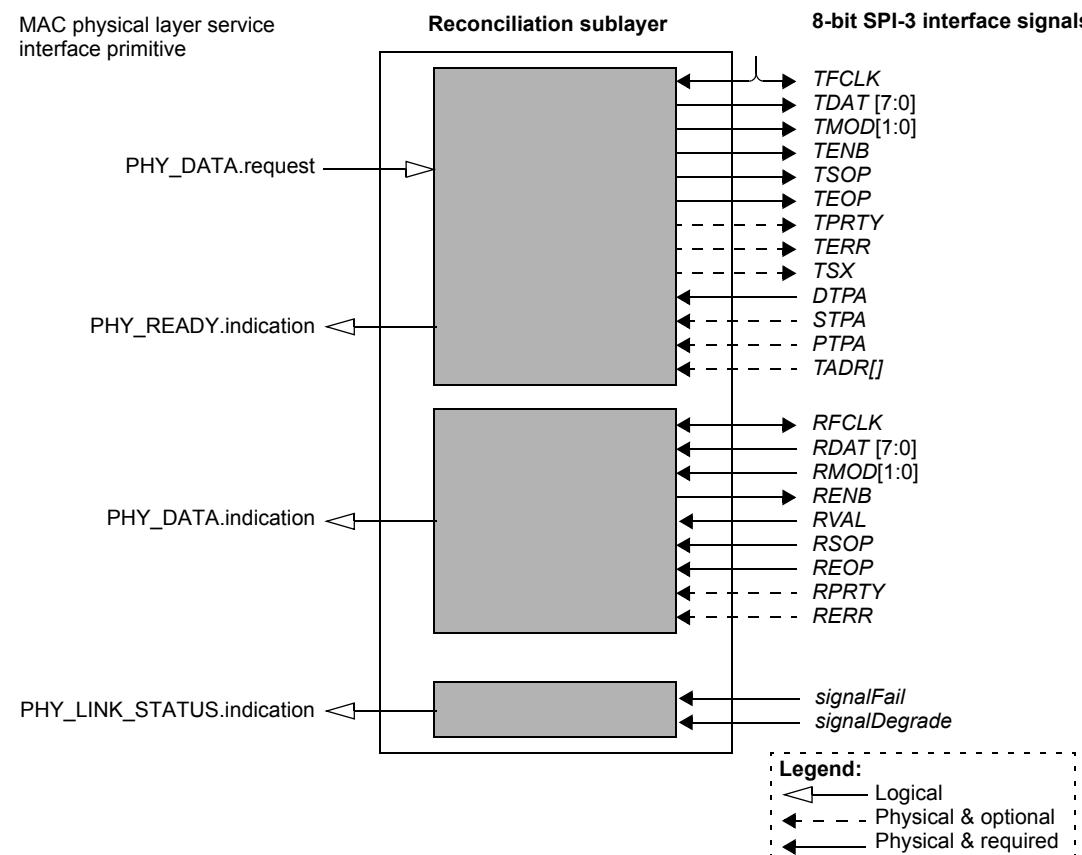


Figure C.7—SRS/GRS inputs and outputs using SPI-3(8)

The 8-bit SPI-3 interface signals are described below. All transmit signals are expected to be updated and sampled using the rising edge of the transmit clock, *TFCLK*; all receive signals are expected to be updated and sampled using the rising edge of the receive clock, *RFCLK*.

TFCLK

A (transmit clock) continuous externally sourced clock used to synchronize data transfer.

TERR (optional)

An optional (transmit error indicator) signal that indicates an error in the current frame. There is no direct mapping from the MAC logical primitives.

TENB

A (transmit write enable) signal that indicates valid symbol transfer at the interface.

TDAT [7:0]

A group of *TDAT* [7:0] (transmit packet data bus) signals forming the transmit data bus, with bit order shown in Figure C.8-a.

TPRTY (optional)

An optional (transmit bus parity) signal that indicates parity calculated over the *TDAT* [7:0] bus. There is no direct mapping of *TPRTY* to the MAC logical primitives.

TSX (optional)

An optional (transmit start of transfer) signal used to signal an in-band port address on the *TDAT* [7:0] bus. There is no direct mapping to the MAC logical primitives.

TSOP

A (transmit start of packet) signal that marks the start of a frame boundary on the *TDAT* [7:0] bus.

TEOP

A (transmit end of packet) signal that marks the end of a frame boundary on the *TDAT* [7:0] bus.

TADR (optional)*PTPA* (optional)

The (transmit PHY address and polled-PHY transmit packet available) signals used in frame-transfer mode for multi-port PHY applications. There is no direct mapping from the MAC logical primitives.

DTPA

A (direct transmit packet available) signal used in the byte-level mode. The result of *DTPA* is used to generate a *PHY_READY.indication*.

STPA (optional)

An optional (selected transmit packet available) status indication signal used in byte-transfer mode for multi-port PHY application. The result of *STPA* is used to generate a *PHY_READY.indication*.

RFCLK

A (receive FIFO write clock) signal that provides a continuous clock.

RVAL

A (receive data valid) signal that indicates the validity of the receive data.

RENB

A (receive read enable) signal that controls the flow of data from the PHYs.

RDAT [7:0]

The (receive packet data bus) signals the supply data, with bit order shown in Figure C.8-b.

RPRTY

An optional (receive bus parity) signal that supplies parity calculated over the *RDAT* [7:0] bus.

RSOP

A (receive start of packet) signal that indicates the start of a frame boundary on the *RDAT* [7:0] bus.

REOP

A (receive end of packet) signal that indicates the end of a frame boundary on the *TDAT* [7:0] bus.

Upon receiving an *REOP*, the data received from *RSOP* constitute a received frame.

SRS: The received frame formulates the *INPUT_FRAME* attribute of the *PHY_DATA.indication* primitive.

GRS: The first 2 bytes of the received frame is mapped into the *LENGTH* attribute and the GFP payload less the header is mapped into the *INPUT_FRAME* attribute of the *PHY_DATA.indication* primitive.

RERR (optional)

An optional (receive error indication) signal that indicates an error in the current frame and does not interfere with the transfer of the frame.

RSX (optional)

An optional (receive start of transfer) signal that indicates an in-band port address on the RDAT [7:0] bus. There is no direct mapping to the MAC logical primitives.

C.3.1.4 Transmit/receive bit ordering

Data are transmitted on TDAT [7:0] and received on RDAT [7:0] in a most significant through least significant bit order, as specified in Figure C.8-a and Figure C.8-b, respectively.

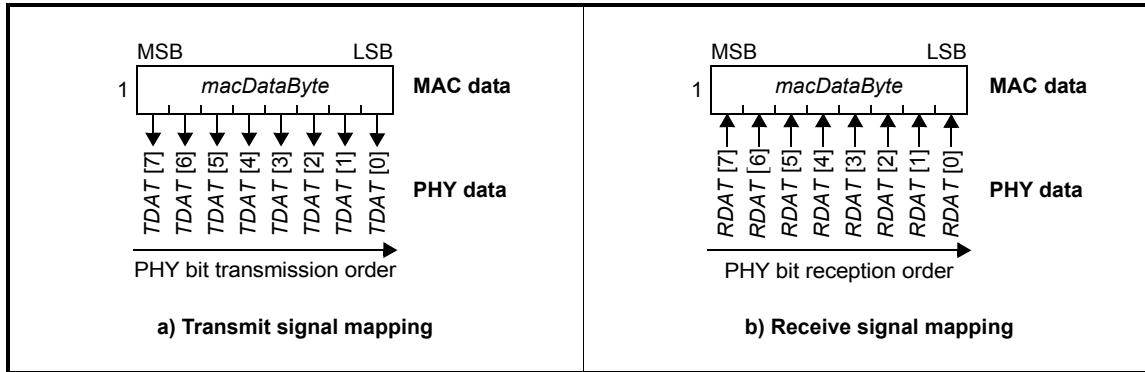


Figure C.8—8-bit SPI-3 transmit signal mapping

C.3.2 Mapping of SPI-3 signals to service interface primitives

The reconciliation sublayer shall map the signals provided at the SPI-3 interface to the MAC physical layer service interface primitives defined in Clause 8. Mappings for the following primitives are defined:

- PHY_DATA.request
- PHY_DATA.indication
- PHY_READY.indication

C.3.2.1 Mapping of PHY_DATA.request

The semantics of the service primitives are as follows:

```
PHY_DATA.request
(
    OUTPUT_FRAME,
    LENGTH
)
```

The parameters of the PHY_DATA.request are described below.

OUTPUT_FRAME
Data bytes that are mapped into the payload.

LENGTH
SRS: A (null) or (non-null) parameter that is ignored.
GRS: A (non-null) value that is mapped into the *PLI* field of the GFP core header.

C.3.2.1.1 When generated

This primitive is invoked by the MAC_datapath entity when it has performed its tasks after it has received a request from its client.

C.3.2.1.2 Effect of receipt

The receipt of this primitive shall cause the reconciliation sublayer to generate the physical frame format required when operating in GRS or SRS mode (see C.2.1 and C.2.2). The reconciliation sublayer will start transferring the physical frame to the PHY according to the SPI-3 protocol: generating *TSOP*, *TDAT* [7:0], *TMOD*, and *TEOP* signals.

C.3.2.2 Mapping of PHY_DATA.indication

The semantics of the primitives are as follows:

```
PHY_DATA.indication
(
    INPUT_FRAME,
    STATUS,
    LENGTH
)
```

The parameters of the PHY_DATA.indication are described below.

INPUT_FRAME

Data bytes that are mapped from the received payload.

STATUS

Identifies frames that are received with error indications from the PHY, for example, due to assertion of a error signal on the PHY electrical interface.

 ERROR—The PHY signaled an error during frame transmission to the reconciliation sublayer.
 OK—(Otherwise).

LENGTH

 SRS: A (null) parameter.

 GRS: A (non-null) value that is mapped from the *PLI* field in the GFP core header.

C.3.2.2.1 When generated

This primitive is invoked by the reconciliation sublayer after it has received a complete frame from the physical interface. The INPUT_FRAME is derived from the signals *RDAT* [31:0], as signaled by the status *RSOP*, *REOP*, *RSX*, in accordance to SPI-3 specification.

The *STATUS* generates OK unless an active error signal is received on RERR.

C.3.2.2.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.3.2.3 Mapping of PHY_READY.indication

The semantics of the primitives are as follows:

```

PHY_READY.indication
(
    READY_STATUS
)

```

The parameters of the PHY_READY.indication are described below.

READY—The reconciliation sublayer is able to accept a frame.
NOT_READY—(Otherwise).

C.3.2.3.1 When generated

Depending on single or multiple PHY mode, *STPA*, *PTPA*, or *DTPA*[] are indications from the PHY to the reconciliation sublayer that the PHY is ready to receive data on the *TDAT* [7:0] bus. A valid indication from the PHY is mapped to the PHY_READY.indication primitive and sent to the reconciliation sublayer client.

C.3.2.3.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.3.3 SRS and GRS 8-bit SPI datastream

There are no electrical differences and no signaling protocols between the GRS and the SRS sublayers. The only difference is the physical frame format as specified in C.2.1 and C.2.2.

C.3.4 Functional specifications

The SRS and GRS using 8-bit SPI-3 interfaces shall meet the functional requirements of paragraphs 8.4, 9, 10, and 11 of the SPI-3 implementation agreement.

C.3.5 Electrical specifications

The SRS and GRS using 8-bit SPI-3 interfaces shall meet the transmit electrical timing requirements of paragraph 10.3, and the receive electrical timing requirements of paragraph 11.3, of the SPI-3 implementation agreement.

The electrical characteristics for *signalDegrade* and *signalFail* are LVTTL and compatible with the supply voltage of the SPI-3 bus.

C.4 SRS and GRS using the 32-bit SPI-3 interface

The SRS and GRS may be implemented with a 32-bit SPI-3 interface.

C.4.1 General requirements

This clause provides a brief description of the signals for the SPI-3 interface and their interaction with the MAC physical layer service interface primitives.

In the event of a discrepancy in the signal description, the OIF SPI-3 specification takes precedence.

C.4.1.1 Summary of major concepts

The SRS and GRS using the 32-bit SPI-3 interface have the following characteristics:

- a) The SRS and GRS map the signals provided at the 32-bit SPI-3 interface to the logical physical layer service interface primitives provided at the MAC.
- b) Each direction of data transfer is independent and serviced by 32-bit data, delimiter, error, and clock signals.
- c) Data and delimiters are synchronous to clock references.
- d) The SPI-3 interface supports logical channels with individual byte-level or frame-level out-of-band flow control.
- e) The SRS and GRS, when using the 32-bit SPI-3 interface, support full-duplex operation only.

C.4.1.2 Rate of operation

The SRS and GRS, when using the 32-bit SPI-3 interface, are capable of supporting data rates of 155 Mb/s to 2.5 Gb/s. The granularity is dependent on the mapping of the SONET/SDH synchronous payload. The interface clock is set fast enough so that the PHY transmit FIFO does not starve and the PHY receive FIFO does not overrun.

SONET/SDH PHYs that provide an SPI-3 with a 32-bit datapath shall support operations, at the SONET/SDH Path level, at the selected rate on the SPI-3. PHYs report the rates at which they are operating via the management interface. The PHY is expected to operate in either streaming, flow-through, or cut-through modes.

C.4.1.3 32-bit SPI-3 structure

The SPI interface can operate in byte-level or packet-level mode. For byte-level transfers, the FIFO status information is presented on a cycle-by-cycle basis. The PHY device indicates the transmit frame available status via the *STPA* or *DTPA*[] signals. *DTPA* signals are provided for simpler implementation to reduce the need for addressing. Support for one or the other is optional, but at least one is to be supported to ensure that the PHY FIFO does not overrun.

For packet-level transfers, the FIFO status information applies to segments of frame data. The reconciliation sublayer polls the status of the PHY for transmit ready status. The reconciliation sublayer polls one of multiple available PHYs by asserting the TADR address of the PHY, which responds via a common PTPA signal.

Additional signals *TENB*, *RENB*, and *RVAL* are used by the protocol to indicate valid data on the bus.

The 32-bit SPI-3 interface is composed of independent transmit and receive paths. Each direction uses 32 data signals (*TDAT*[31:0] and *RDAT*[31:0]), word modulo signals (*TMOD*[1:0] and *RMOD*[1:0]), clocks (*TFCLK* and *RFCLK*), start of packet delimiter signals (*TSOP* and *RSOP*), end of packet delimiter signals (*TEOP* and *REOP*), and other control signals shown in Figure C.10. These signals are fully defined in the SPI-3 implementation agreement. Figure C.9 shows a schematic view of the SRS and GRS inputs and outputs using the 32-bit SPI-3 interface.

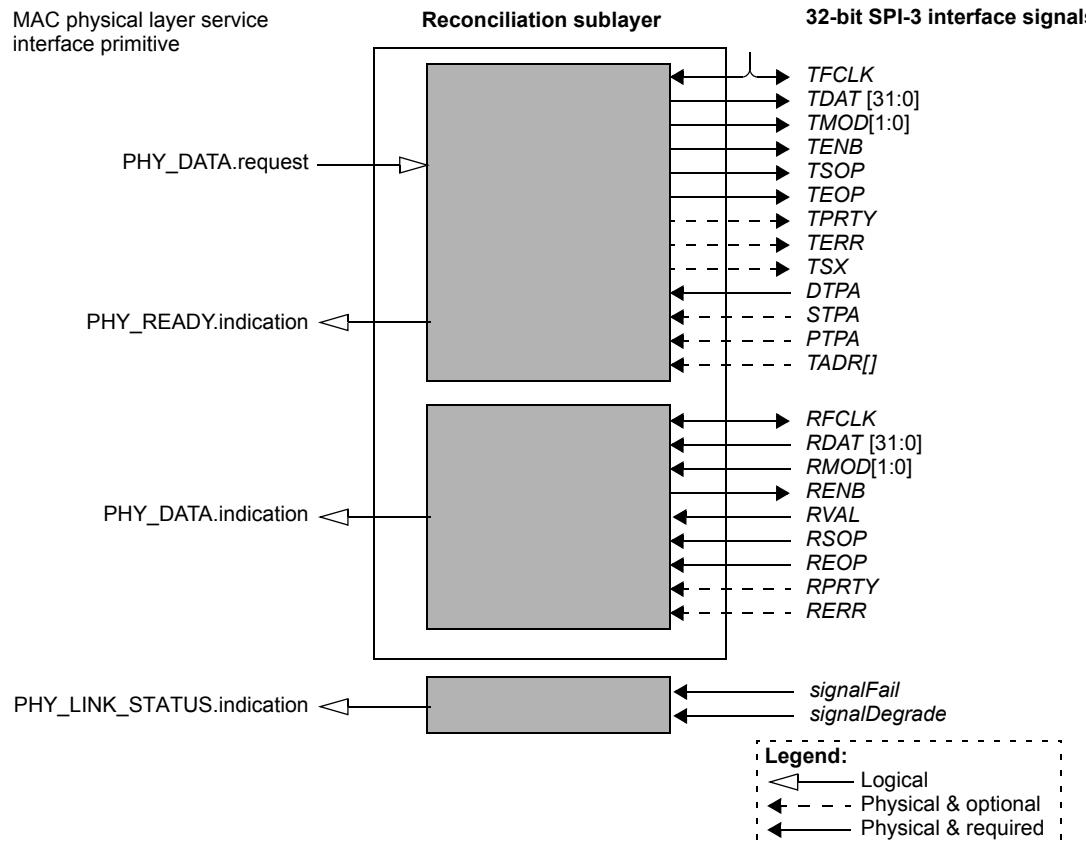


Figure C.9—SRS/GRS inputs and outputs using 32-bit SPI-3

The 32-bit SPI-3 interface signals are described in the following subclauses. All signals are expected to be updated and sampled using the rising edge of the transmit clock, *TFCLK*.

TFCLK

A continuous (transmit clock) externally sourced clock used to synchronize data transfer.

TERR

An optional (transmit error indicator) signal that indicates an error in the current frame. There is no direct mapping from *TERR* to the MAC logical primitives.

TENB

A (transmit write enable) signal that indicates valid symbol transfer at the interface.

TDAT [31:0]

The (transmit packet data bus) 32-bit transmit data bus signals, transmitted most significant to least significant bit/byte order, as shown in Figure C.10-a.

TMOD[1:0]

A pair of (transmit word modulo) signals that specifies the number of valid bytes in *TDAT*[31:0], as shown in Table C.3.

Table C.3—*TMOD[1:0]* and corresponding valid data

<i>TMOD[1:0]</i>	Valid data
00_2	<i>TDAT</i> [31:0]
01_2	<i>TDAT</i> [31:8]
10_2	<i>TDAT</i> [31:16]
11_2	<i>TDAT</i> [31:24]

TPRTY

A (transmit bus parity) parity signal calculated over the *TDAT*[31:0] bus. There is no direct mapping from the MAC logical primitives.

***TSX* (optional)**

An optional (transmit start of transfer) signal used to signal an in-band port address on the *TDAT*[31:0] bus. There is no direct mapping to MAC logical interface primitives.

TSOP

A (transmit start of packet) signal that marks the start of a frame boundary on the *TDAT*[31:0] bus.

TEOP

A (transmit end of packet) signal that marks the end of a frame boundary on the *TDAT*[31:0] bus.

***TADR* and *PTPA* (optional)**

The optional (transmit PHY address and polled-PHY transmit packet available) signals used in frame-transfer mode for multi-port PHY applications. There is no direct mapping from the MAC logical interface primitives.

DTPA

A (direct transmit packet available) signal used in the byte-level mode. The result of *DTPA* is used to generate a PHY_READY.indication.

***SPTA* (optional)**

An optional (selected transmit packet available) status indication signal used in byte-transfer mode for multi-port PHY applications. The result of *SPTA* is used to generate a PHY_READY.indication.

RFCLK

The (receive FIFO write clock) continuous receive clock signal.

RVAL

The (receive data valid) signal that indicates the validity of the receive data.

RENB

The (receive read enable) signal that controls the flow of data from the PHY.

***RDAT*[31:0]**

The (receive packet data bus) 32-bit transmit data bus signals, transmitted most significant to least significant bit/byte order, as shown in Figure C.10-b.

RPRTY

An (receive parity) odd parity signal calculated over the *RDAT*[31:0] bus. There is no direct mapping to the MAC logical interface primitives.

RMOD[1:0]

A pair of (receive word modulo) signals that indicates the number of valid bytes of data in *RDAT* [31:0], as shown in Table C.4.

Table C.4—*RMOD[1:0]* and corresponding valid data

<i>RMOD[1:0]</i>	Valid data
00_2	<i>RDAT</i> [31:0]
01_2	<i>RDAT</i> [31:8]
10_2	<i>RDAT</i> [31:16]
11_2	<i>RDAT</i> [31:24]

RSOP

A (receive start of packet) signal that marks the start of a frame boundary on the *RDAT* [31:0] bus.

REOP

A (receive end of packet) signal that marks the end of a frame boundary on the *RDAT* [31:0] bus.

***RERR* (optional)**

An optional (receive error indicator) signal that indicates an error in the current frame and does not interfere with the transfer of the frame.

RSX

A (receive start of transfer) signal that indicates when the in-band port address is present on the *RDAT* [31:0] bus. *RSX* is optional for single-channel PHY devices, but required for multi-port PHY devices.

C.4.1.4 Transmit/receive bit ordering

Data are transmitted on *TDAT* [31:0] and received on *RDAT* [31:0] in a most significant through least significant bit order, as shown in Figure C.10-a and Figure C.10-b, respectively.

C.4.2 Mapping of SPI-3 signals to service interface primitives

The reconciliation sublayer shall map the signals provided at the SPI-3 interface to the MAC physical layer service interface primitives defined in Clause 8. Mappings for the following primitives are defined:

- *PHY_DATA.request*
- *PHY_DATA.indication*
- *PHY_READY.indication*

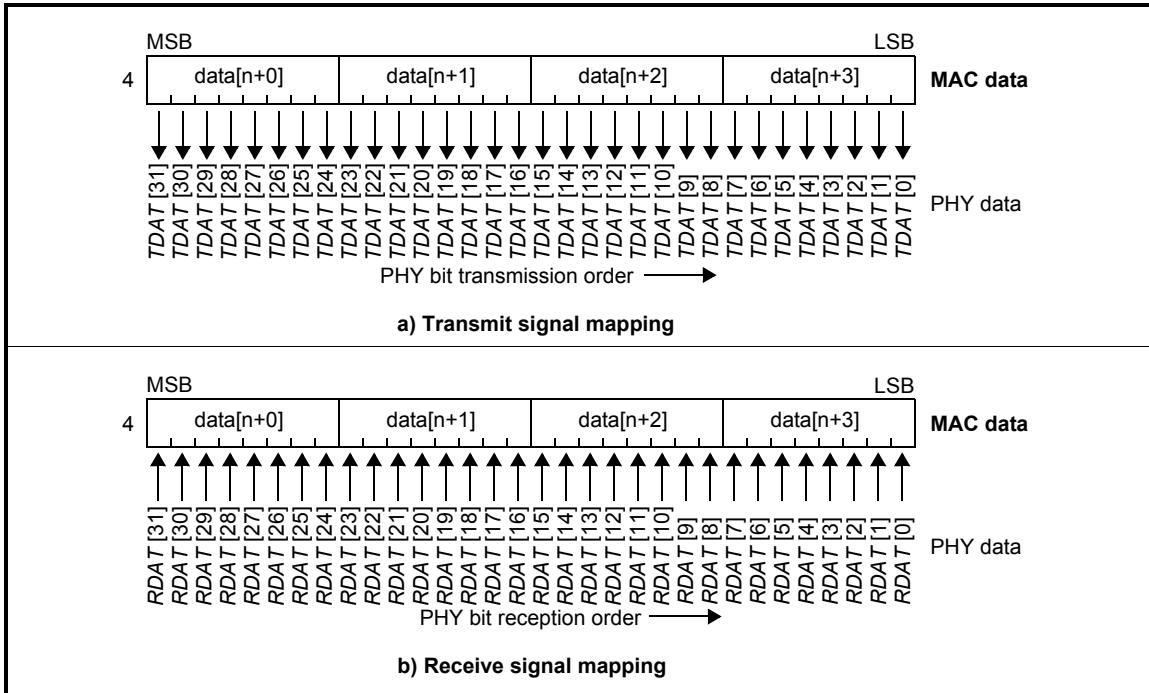


Figure C.10—32-bit SPI-3 signal mappings

C.4.2.1 Mapping of PHY_DATA.request

The semantics of the primitives are as follows:

```
PHY_DATA.request
(
    OUTPUT_FRAME,
    LENGTH
)
```

The parameters of the MA_DATA.request are described below.

OUTPUT_FRAME

A complete RPR frame that is conveyed to the PHY by the signals *TDAT* [31:0] and *TMOD*[1:0].

LENGTH

SRS: A (null) parameter.

GRS:

(non-null)—A value that is mapped to the *PLI* field in the GFP core header.

(null)—Indicates the value of the LENGTH should not be used for PLI mapping.

On each *TFCLK* rising edge, 32 bits of data are transferred from the reconciliation sublayer to the PHY. Data are mapped to a *TDAT* [31:0] signal in sequence (*TDAT* [0:7], ..., *TDAT* [24:31], *TDAT* [0:7]), using the endian coding specified in C.4.2, and with the parity information in the *TPRTY* signal corresponding to the value of the previous 32-bit data word. The *TMOD*[1:0] signals are always fixed to a value of “00” except when the end-of-frame is transmitted.

When transmitting RPR MAC frames, the *TENB* signal is low, so the *TSX* signal is ignored. The usage of the *TSX* and *TENB* signals for in-band addressing with multi-port PHY is an implementation choice and out of the scope of this standard.

After transmission of the complete frame, the reconciliation sublayer generates an end-of-frame on the SPI-3 interface. The reconciliation sublayer requests transmission of 32 data bits by the PHY, together with the proper parity information in the *TPRTY* signal, containing the values of the previous *PHY_DATA.request* transactions not yet transmitted. When transmitting this *TDAT*[31:0] signal, the *TEOP* is also high and the *TMOD*[1:0] represents the number of valid data bytes on the *TDAT*[31:0] as defined in the SPI-3 specification.

The SRS and GRS do not generate preamble or interframe gap because they are not needed for SONET/SDH PHYs.

C.4.2.1.1 When generated

This primitive is invoked by the MAC_datapath entity when it has performed its tasks after it has received a request from its client.

C.4.2.1.2 Effect of receipt

The receipt of this primitive shall cause the reconciliation sublayer to generate the physical frame format required when operating in GRS or SRS mode (see C.2.1 and C.2.2). The reconciliation sublayer will start transferring the physical frame to the PHY according to the SPI-3 protocol: generating *TSOP*, *TDAT*[31:0], *TMOD*[1:0], and *TEOP* signals.

C.4.2.2 Mapping of *PHY_DATA.indication*

The semantics of the primitives are as follows:

```
PHY_DATA.indication
(
    INPUT_FRAME,
    STATUS,
    LENGTH
)
```

The parameters of the *PHY_DATA.indication* are described below.

INPUT_FRAME

Data derived from the signals *RFCLK*, *RVAL*, *RENB*, *RDAT*[31:0], *RPRTY*, *RMOD*[1:0], *RSOP*, *REOP*, *RERR*, *RSX*.

STATUS

Identifies frames that are received with error indications from the PHY, for example, due to assertion of an error signal on the PHY electrical interface.

ERROR—The PHY signaled an error during frame transmission to the reconciliation sublayer.

OK—(Otherwise).

LENGTH

Available for reconciliation sublayers to indicate the length of the *INPUT_FRAME* parameter.

(null)—Not used.

When receiving frames, the *RENB* signal is low and the *RVAL* is high, so the *RSX* signal is ignored. The usage of the *RSX* and *RENB* signals for in-band addressing with multi-port PHY is an implementation choice and out of the scope of this standard. If used, this feature should be compliant with the requirements of the SPI-3 specification.

The INPUT_FRAME values are derived from the signals *RMOD*[1:0] and *RDAT*[31:0] received from the PHY on each rising edge of the *RFCLK*. Each primitive generated to the MAC sublayer entity corresponds to a *PHY_DATA.request* issued by the MAC at the other end of the link connecting two RPR stations.

For each *RDAT*[31:0] during frame reception, the reconciliation sublayer receives four bytes of a frame until the end of frame (when the *REOP* is high), where one, two, three, or four bytes will be received from the *RDAT*[31:0] according to the value coded in the *RMOD*[1:0] as defined in the SPI-3 specification.

During frame reception, each *RDAT*[31:0] signal shall be mapped in sequence into a received frame (*RDAT*[0:7], ..., *RDAT*[24:31], *RDAT*[0:7]).

C.4.2.2.1 When generated

This primitive is invoked by the reconciliation sublayer after it has received a complete frame from the physical interface. The INPUT_FRAME is derived from the signals *RDAT*[31:0], *RMOD*[1:0], as signaled by the status *RSOP*, *REOP*, *RSX*, in accordance with the SPI-3 specification.

The STATUS generated is OK unless an active error signal is received on RERR.

C.4.2.2.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.4.2.3 Mapping of *PHY_READY.indication*

The semantics of the primitive are as follows:

```
PHY_READY.indication
(
    READY_STATUS
)
```

The parameters of the *PHY_READY.indication* are described below.

READY_STATUS

An attribute that takes the value of:

READY—RS can accept data request.

NOT_READY—RS indicates that the MAC should not send data.

C.4.2.3.1 When generated

Depending on single or multiple PHY mode, *STPA*, *PTPA*, or *DPTA*[], are indications from the PHY to the reconciliation sublayer that the PHY is ready to receive data on the *TDAT*[31:0] bus. A valid indication from the PHY is mapped to the *PHY_READY.indication* primitive and sent to the reconciliation sublayer client.

C.4.2.3.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.4.3 SRS and GRS 32-bit SPI datastream

There is no electrical difference and no signaling protocol between the GRS and the SRS sublayers. The only difference is the physical frame format as specified in C.2.1 and C.2.2.

C.4.4 Functional specifications

The SRS and GRS using 32-bit SPI-3 interfaces shall meet the functional requirements of paragraphs 8, 9, 10, and 11 of the SPI-3 implementation agreement.

C.4.5 Electrical timing specifications

The SRS and GRS using 32-bit SPI-3 interfaces shall meet the transmit AC timing requirements of paragraph 10.3, and the receive AC timing requirements of paragraph 11.3, of the SPI-3 implementation agreement.

C.5 SRS and GRS using the SPI-4 Phase 1 interface

The SRS and GRS may be implemented with an SPI-4.1 interface.

C.5.1 General requirements

This clause provides a brief description of the signals for the SPI-4.1 interface and their interactions with the MAC physical layer service interface primitives. In case of any discrepancy in the signal description, the OIF SPI-4.1 specification takes precedence over C.5.1.1.

C.5.1.1 Summary of major concepts

The SRS and GRS using the SPI-4.1 interface have the following characteristics:

- a) The SRS and GRS map the signals provided at the SPI-4.1 interface to the logical physical layer service interface primitives provided at the MAC.
- b) Each direction of data transfer is independent, and serviced by 64-bit data, delimiter, control, error, and clock signals.
- c) Data, control, and delimiter are source-synchronous HSTL signals.
- d) The SRS and GRS, when using the SPI-4.1 interface, support full-duplex operation only.

C.5.1.2 Rate of operation

The SRS and GRS using the SPI-4.1 interface are capable of supporting data rates of 622 Mb/s to 10 Gb/s.

C.5.1.3 SPI-4.1 structure

SPI-4.1 supports a single 64-bit interface and a 4 x 16-bit interface. The 4 x 16 mode is not supported within this standard. An RPR MAC is a single link layer device, and it requires the PHY interface to operate as a single PHY device.

Figure C.11 shows a schematic view of the SRS and GRS inputs and outputs using the 64-bit SPI-4.1 interface.

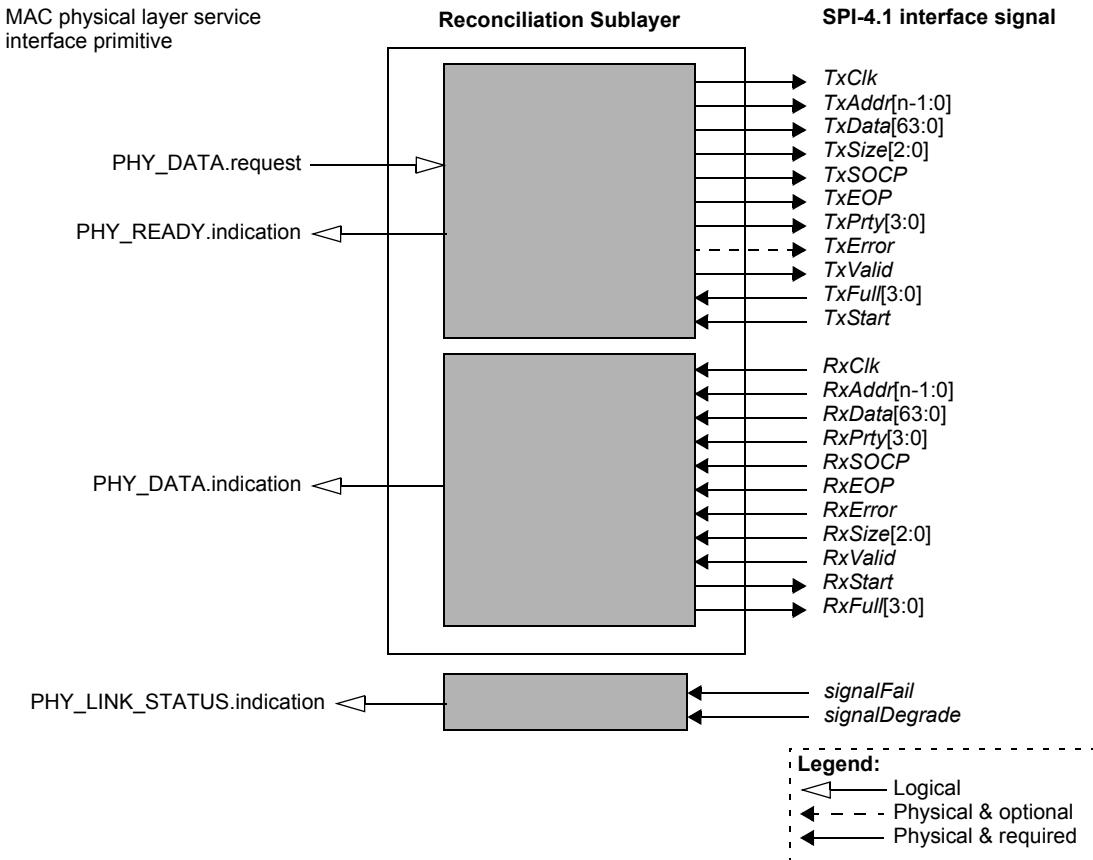


Figure C.11—SPI-4.1 single 64-bit interface signals

The SPI-4.1 signals are described below.

TxClock

A (transmit clock) continuous clock used to synchronize data transfer transactions between the reconciliation sublayer and the PHY layer device. SPI-4.1 uses source synchronous clocking.

TxError (optional)

An optional (transmit error) signal used to indicate that there is an error in the current frame. The *TxError* does not interfere with the transfer of data in the PHY. There is no direct mapping from the MAC logical interface primitives.

TxValid

A (transmit data valid) physical interface protocol signal for flow control at the interface level.

TxDATA[63:0]

A group of (transmit packet data bus) forming the 64-bit bus that carries the frame bytes to the selected transmit FIFO port whose address is select by the *TxAddr* signals. Data are transmitted on *TxDATA[63:0]* in a most significant through least significant bit/byte order, as specified in Figure C.11.

TxAddr[n-1:0]

A group of (transmit address) signals specifying the transmit PHY channel address.

TxSize[2:0]

A group of (transmit word modulo) signals for the 64-bit *TxDATA* bus that indicates the number of valid data bytes, as shown in Table C.5.

Table C.5—*TxSize* and corresponding valid data

<i>TxSize[2:0]</i>	Valid data
000_2	<i>TxDATA[63:0]</i>
001_2	<i>TxDATA[63:54]</i>
010_2	<i>TxDATA[63:48]</i>
011_2	<i>TxDATA[63:40]</i>
100_2	<i>TxDATA[63:32]</i>
101_2	<i>TxDATA[63:24]</i>
110_2	<i>TxDATA[63:16]</i>
111_2	<i>TxDATA[63:8]</i>

TxPrty[3:0]

A group of (transmit bus parity) signals calculated over the *TxDATA* bus, as illustrated in Table C.6. The parities are used only to detect interface error.

Table C.6—*TxPrty* and *TxDATA* parity

<i>TxPrty</i>	<i>TxDATA</i> parity
<i>TxPrty[0]</i>	Odd parity over <i>TxDATA[15:0]</i>
<i>TxPrty[1]</i>	Odd parity over <i>TxDATA[31:16]</i>
<i>TxPrty[2]</i>	Odd parity over <i>TxDATA[47:32]</i>
<i>TxPrty[3]</i>	Odd parity over <i>TxDATA[63:48]</i>

TxSOP

A (transmit start of packet) signal that delineates the frame boundaries on the *TxDATA* bus. When *TxSOP* is high, the start of the frame is present on the *TxDATA* bus. *TxSOP* is required to be present at the beginning of every frame.

TxEOP

A (transmit end of packet) signal that delineates the frame boundaries on the *TxDATA* bus. When *TxEOP* is high, the end of the frame is present on the *TxDATA* bus. *TxEOP* is required to be present at the end of every frame.

TxStart

A (transmit flow control frame start) signal that marks the start of the *TxFull* frame boundary, which is used to convey PHY transmit FIFO full status.

TxFull[3:0]

A (transmit flow control full) signal used by the PHY to signal the reconciliation sublayer that there is no room to receive data from the reconciliation sublayer for a port of a multiport PHY device.

RxClock

A (receive FIFO write clock) continuous clock signal used to synchronize data transfer transactions between the PHY and the reconciliation sublayer.

RxValid

A (receive data valid) signal that indicates when receive data signals are valid.

RxDATA[63:0]

A group of (receive packet data bus) signals forming the 64-bit bus that carries the frame bytes that are read from the PHY. Mapping of *RxDATA* signals is shown in Figure C.11.

RxPrty[3:0]

A group of (receive parity) parity signals calculated over the *RxDATA* bus, as illustrated in Table C.7.

Table C.7—*RxPrty* and *RxDATA* parity

<i>RxPrty</i>	<i>RxDATA</i> parity
<i>RxPrty[0]</i>	Odd parity over <i>RxDATA[15:0]</i>
<i>RxPrty[1]</i>	Odd parity over <i>RxDATA[31:16]</i>
<i>RxPrty[2]</i>	Odd parity over <i>RxDATA[47:32]</i>
<i>RxPrty[3]</i>	Odd parity over <i>RxDATA[63:48]</i>

RxAddr[n-1:0]

A group of (receive address) signals specifying the receive PHY channel address.

RxSize[2:0]

A group of (receive word modulo) signals for 64-bit *RxDATA[]* bus that indicates the number of valid bytes of data in *RxDATA*, as shown in Table C.8.

Table C.8—*RxSize* and corresponding valid data

<i>RxSize[2:0]</i>	Valid data
000_2	<i>TxDATA[63:0]</i>
001_2	<i>TxDATA[63:56]</i>
010_2	<i>TxDATA[63:48]</i>
011_2	<i>TxDATA[63:40]</i>
100_2	<i>TxDATA[63:32]</i>
101_2	<i>TxDATA[63:24]</i>
110_2	<i>TxDATA[63:16]</i>
111_2	<i>TxDATA[63:8]</i>

RxSOCP

A (receive start of packet) signal that delineates the frame boundaries on the *RxDATA[]* bus. When *RxSOCP* is high, the start of the frame is present on the *RxDATA* bus. *RxSOCP* is required to be present at the beginning of every frame.

RxEOP

A (receive end of packet) signal that delineates the frame boundaries on the *RxDATA* bus. When *RxEOP* is high, the end of the frame is present on the *RxDATA* bus. *RxEOP* is required to be present at the end of every frame.

RxError

A (receive error indicator) that indicates when the current frame is in error.

C.5.1.4 Transmit/receive bit ordering

Data are transmitted on *TxDAT* [63:0] and received on *RxDAT* [63:0] in a most significant through least significant bit order, as shown in Figure C.12-a and Figure C.12-b, respectively.

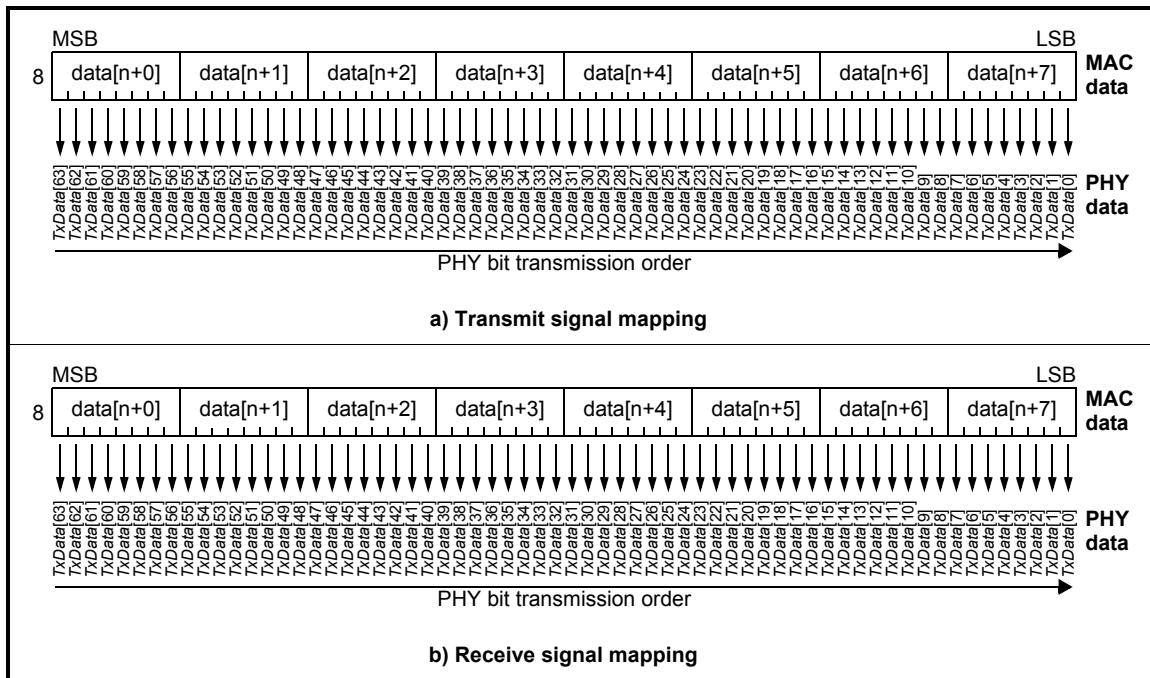


Figure C.12—64-bit SPI-4.1 signal mappings

C.5.2 Mapping of SPI-4 signals to service interface primitives

The reconciliation sublayer shall map the signals provided at the SPI-4 interface to the MAC physical layer service interface primitives defined in Clause 8. Mappings for the following primitives are defined:

- *PHY_DATA.request*
- *PHY_DATA.indication*
- *PHY_READY.indication*

C.5.2.1 Mapping of *PHY_DATA.request*

The semantics of the primitive are as follows:

```
PHY_DATA.request
(
    OUTPUT_FRAME,
    LENGTH
)
```

The parameters of the PHY_DATA.request are described below.

OUTPUT_FRAME

Data bytes mapped into series of TxData octets. The PHY inserts Idle. The start of the frame is signaled by the *TxSOP*, and the end aligns with the signaling of *TxEOP* and the correct value for *TxSize[2:0]*.

LENGTH

Is used by the GRS. It is mapped to the *PLI* field of the core header.

C.5.2.1.1 When generated

This primitive is invoked by the MAC_datapath entity when it has performed its tasks after it has received a request from its client.

C.5.2.1.2 Effect of receipt

The receipt of this primitive shall cause the reconciliation sublayer to generate the physical frame format required when operating in GRS or SRS mode (see C.2.1 and C.2.2).

C.5.2.2 Mapping of PHY_DATA.indication

The semantics of the primitive are as follows:

```
PHY_DATA.indication
(
    INPUT_FRAME,
    STATUS,
    LENGTH
)
```

The parameters of the PHY_DATA.request are described below.

INPUT_FRAME

GRS: Upon receiving a complete frame from the PHY, all valid octets received from *RxSOCP* and *RxEOP* less the GFP core header and payload header.

SRS: Upon receiving a complete frame from the PHY, all valid octets received from *RxSOP* and *RxEOP*.

STATUS

Indicates an interface error for the current data request:

ERROR—An active error signal is received on *RxError*.

OK—An active error signal is not received on *RxError*.

LENGTH

SRS: A (null) parameter.

GRS: The core header field *PLI*.

C.5.2.2.1 When generated

This primitive is invoked by the reconciliation sublayer after it has received a complete frame from the physical interface.

C.5.2.2.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.5.2.3 Mapping of PHY_READY.indication

The semantics of the primitive are as follows:

```
PHY_READY.indication
(
    READY_STATUS
)
```

The parameters of the PHY_READY.indication are described below.

READY_STATUS

An attribute that takes the value of:

READY—RS can accept a data request.

NOT_READY—RS indicates that the MAC should not send data.

C.5.2.3.1 When generated

Depending on single or multiple PHY mode, *TxStart* and *TxFull* are indications from the PHY to the reconciliation sublayer that the PHY is ready to receive data on the *TxDATA* bus. A valid indication from the PHY is mapped to the PHY_READY.indication primitive and sent to reconciliation sublayer client.

C.5.2.3.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.5.3 SRS and GRS 64-bit SPI datastream

There is no electrical difference and no signaling protocol between the GRS and the SRS sublayers. The only difference is the physical frame format as specified in C.2.1 and C.2.2.

C.5.4 Functional specifications

The SRS and GRS using SPI-4.1 interfaces shall meet the functional requirements of section 6 of the SPI-4.1 implementation agreement.

C.5.5 Electrical specifications

The SRS and GRS using the SPI-4.1 interface shall meet the electrical timing requirements of sections 10, 11, and 12 of the SPI-4 implementation agreement.

C.6 SRS and GRS using SPI-4.2 interface

The SRS and GRS may be implemented with an SPI-4.2 interface.

SPI-4.2 is an interface for frame and cell transfer between a physical layer (PHY) device and a link layer device, for aggregate bandwidths of OC-192 rate.

On both the transmit and receive interfaces, FIFO status information is sent separately from the corresponding data path. By taking FIFO status information out-of-band, it is possible to decouple the

transmit and receive interfaces so that each operates independently of the other. Such an arrangement makes SPI-4 suitable not only for bidirectional but also for unidirectional link layer devices.

In both the transmit and receive interfaces, the frame's address, delineation information, and error control coding is sent in-band with the data.

C.6.1 General requirements

This clause provides a brief description of the signals for the SPI-4.2 interface and their interactions with the MAC physical layer service interface primitives. In case of any discrepancy in the signal description, the OIF SPI-4.2 specification takes precedence over C.6.1.1.

C.6.1.1 Summary of major concepts

The SRS and GRS using the SPI-4.2 interface have the following characteristics:

- a) The SRS and GRS map the signals provided at the SPI-4.2 interface to the logical physical layer service interface primitives provided at the MAC.
- b) Each direction of data transfer is independent, and serviced by 16-bit data, control, and clock signals.
- c) Each direction of data transfer includes separate FIFO status channels consisting of status signals and status clocks.
- d) The SRS and GRS, when using the SPI-4.2 interface, support full-duplex operation only.

Figure C.13 shows a schematic view of the SRS and GRS inputs and outputs using the 64-bit SPI-4.2 interface.

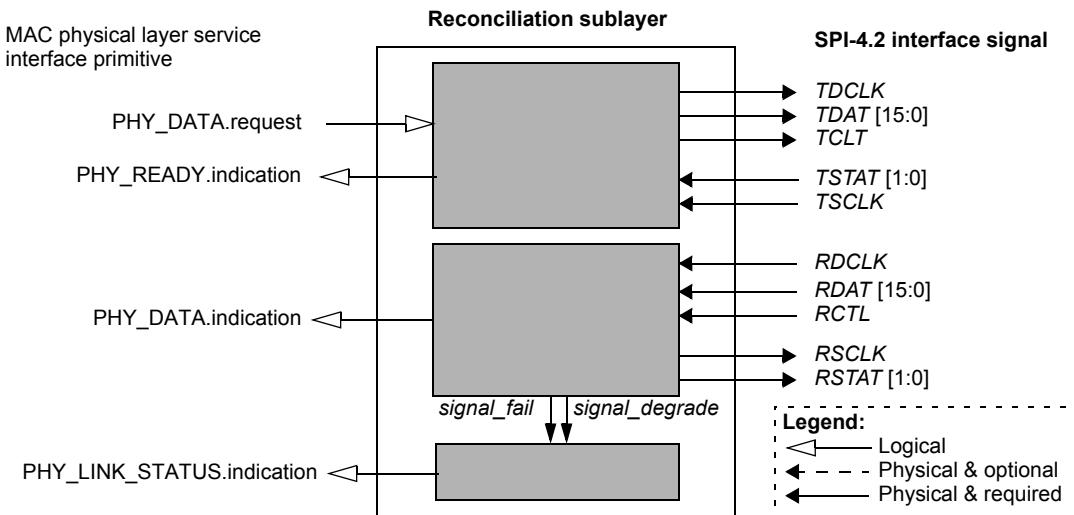


Figure C.13—SPI-4.2 interface signals

C.6.1.2 Rate of operation

The SRS and GRS using the SPI-4.2 interface are capable of supporting data rates of 622 Mb/s to 10 Gb/s.

C.6.1.3 SPI-4.2 structure

The SPI-4.2 interface signals are described below. Data and control lines are driven from the rising and falling edges of the clock.

TDCLK

A (transmit data clock) clock associated with *TDAT*[15:0] and *TCTL*. *TDCLK* provides the datapath source-synchronous double-edge clocking with a minimum frequency of 311 MHz. Data and control lines are driven off the rising and falling edges of the clock. *TDCLK* is sourced by the MAC to the PHY.

TDAT[15:0]

A group of (transmit data) signals that forms a 16-bit bus used to carry payload data and in-band control words from the Link Layer to the PHY device. A control word is present on *TDAT*[15:0] when *TCTL* is high. The minimum data rate for *TDAT*[15:0] is 622 Mb/s. Mapping of *TDAT*[15:0] is shown in Figure C.14-a.

TCTL

A (transmit control) signal that is high when a control word is present on *TDAT*[15:0]; otherwise, it is low. *TCTL* is sourced by the MAC to the PHY.

TSCLK

A (transmit status clock) clock signal associated with *TSTAT*[1:0] providing source-synchronous clocking. For low-voltage transistor-transistor logic (LVTTL) I/O, a maximum clock rate restraint is $\frac{1}{4}$ that of the datapath clock rate. Low-voltage differential signaling (LVDS) I/O allows a maximum of that equal to the datapath clock (double-edge clocking).

TSTAT[1:0]

A group of (transmit FIFO status) signals used to carry round-robin FIFO status information, along with associated error detection and framing. The maximum data rate for *TSTAT*[1:0] is dependent on the I/O type, either LVDS or LVTTL, and is limited to its respective *TSCLK* restraints. *TSTAT*[1:0] is sourced by the PHY to the MAC. The FIFO status formats are shown in Table C.9.

Table C.9—*TSTAT* and corresponding FIFO status

<i>TSTAT</i> [1:0]	FIFO status
00 ₂	STARVING
01 ₂	HUNGRY
10 ₂	SATISFIED
11 ₂	Reserved ^a

^aOIF specification indicates that “reserved” is for framing or to indicate a disabled link status

RDCLK

A (receive data clock) clock signal associated with *RDAT*[15:0] and *RCTL*. *RDCLK* provides the datapath source-synchronous double-edge clocking with a minimum frequency of 311 MHz. Data and control lines are driven off the rising and falling edges of the clock. *RDCLK* is sourced by the PHY to the MAC.

RDAT[15:0]

A group of (receive data) signals forming the 16-bit bus that carries payload data and in-band control from the PHY to the Link Layer device. A control word is present on *RDAT*[15:0] when *RCTL* is high. The minimum data rate for *RDAT*[15:0] is 622 Mb/s. Mapping of *RDAT*[15:0] is shown in Figure C.14-b.

RCTL

A (receive control) signal that is high when a control word is present on *RDAT* [15:0]; otherwise, it is low. *RCTL* is sourced by the PHY to the MAC.

RSCLK

A (receive status clock) clock signal associated with *RSTAT* [1:0] providing source-synchronous clocking. *RSCLK* is sourced by the MAC to the PHY. LVDS I/O allows a maximum of that equal to the datapath clock (double-edge clocking).

RSTAT [1:0]

A pair of (receive FIFO status) signals that carry round-robin FIFO status information, along with associated error detection and framing. The maximum data rate for *RSTAT* [1:0] is dependent on the I/O type, either LVDS or LVTTL, and is limited to its respective RSCLK restraints. *RSTAT* [1:0] is sourced by the MAC to the PHY. The FIFO status formats are shown in Table C.9.

Table C.10—*RSTAT* and corresponding FIFO status

<i>RSTAT</i> [1:0]	FIFO status	READY_STATUS
00 ₂	STARVING	READY
01 ₂	HUNGRY	READY
10 ₂	SATISFIED	NOT_READY
11 ₂	Reserved ^a	

^aThe OIF specification indicates that “reserved” is for framing or to indicate a disabled link status

C.6.1.4 Receive/transmit bit ordering

Data are transmitted on *TDAT* [15:0] and received on *RDAT* [15:0] in a most significant through least significant bit order, as shown in Figure C.14-a and Figure C.14-b.

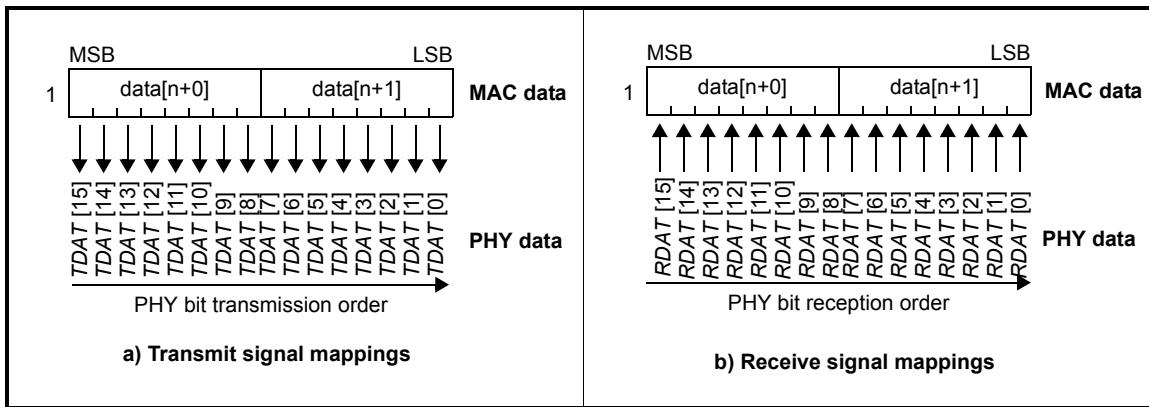


Figure C.14—SPI-4.2 signal mappings

C.6.2 Mapping of SPI-4 signals to service interface primitives

The reconciliation sublayer shall map the signals provided at the SPI-4 interface to the MAC physical layer service interface primitives defined in Clause 8. Mappings for the following primitives are defined:

- PHY_DATA.request
- PHY_DATA.indication
- PHY_READY.indication

C.6.2.1 Mapping of PHY_DATA.request

The semantics of the primitive are as follows:

```
PHY_DATA.request
(
    OUTPUT_FRAME,
    LENGTH
)
```

The parameters of the PHY_DATA.request are described below.

OUTPUT_FRAME
 Mapped from the received packet.
LENGTH
 Available for reconciliation sublayers to indicate the length of the INPUT_FRAME parameter.
SRS:
 (null)—Not used.
GRS:
 (non-null)—Mapped from the *PLI* field in the GFP core header.

C.6.2.1.1 When generated

This primitive is generated by the MAC after receiving a client request and after it has performed its tasks.

C.6.2.1.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.6.2.2 Mapping of PHY_DATA.indication

The semantics of the primitive are as follows:

```
PHY_DATA.indication
(
    INPUT_FRAME,
    STATUS,
    LENGTH
)
```

The parameters of the PHY_DATA.indication are described below.

INPUT_FRAME
 A data frame that is mapped from the received payload.

STATUS

An indication of whether the frame was received in error.
(null)—No error indication is provided.

LENGTH

SRS: A (null) parameter
GRS: A (non-null) value mapped from the *PLI* field in the GFP core header.

C.6.2.2.1 When generated

This primitive is invoked by the reconciliation sublayer after it has received a complete frame from the physical interface. The *INPUT_FRAME* is derived from the signals *RDAT* [31:0], *RSCLK*, and *RSTAT* [1:0] in accordance with the SPI-4.2 specification.

C.6.2.2.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.6.2.3 Mapping of PHY_READY.indication

The semantics of the primitive are as follows:

```
PHY_READY.indication
(
    READY_STATUS
)
```

The parameters of the PHY_READY.indication are described below.

READY_STATUS:

An indication of whether the reconciliation sublayer is able to accept a frame from the MAC.
READY—The reconciliation sublayer is able to accept a frame.
NOT_READY—(Otherwise).

C.6.2.3.1 When generated

READY is generated when *TSTAT* indicates HUNGRY or STARVING. NOT_READY is generated when *TSTAT* indicates SATISFIED.

C.6.2.3.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 7.

C.6.3 SRS and GRS SPI-4.2 datastream

There is no electrical difference and no signaling protocol between the GRS and the SRS sublayers. The only difference is the physical frame format as specified in C.2.1 and C.2.2.

C.6.4 Functional specifications

The SRS and GRS, when using SPI-4.2 interface, shall meet the functional requirements of section 6 of the OIF SPI-4.2 implementation agreement.

C.6.5 Electrical specifications

The SRS and GRS, when using the SPI-4.2 interface, shall meet the electrical timing requirements of sections 10, 11, and 12 of the SPI-4 implementation agreement.

C.7 Protocol Implementation Conformance Statement (PICS) proforma for Annex C³³

C.7.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Annex C, SONET/SDH reconciliation sublayers, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

C.7.2 Identification

C.7.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

C.7.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, SONET/SDH reconciliation sublayers
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

³³Copyright release for PICS proformas: Users of this standard may freely reproduce the PICS proforma in this annex so that it can be used for its intended purpose and may further publish the completed PICS.

C.7.3 Major capabilities/options

Item	Feature	Subclause	Value/Comment	Status	Support
C1*	8 bit SPI-3	C.3	155Mb/s to 622 Mb/s	O	Yes [] No []
C2*	32 bit SPI-3	C.4	155 Mb/s 2.5Gb/s	O	Yes [] No []
C3*	SPI-4.1	C.5	200Mb/s to 10 Gb/s	O	Yes [] No []
C4*	SPI-4.2	C.6	622 Mb/s to 10Gb/s	O	Yes [] No []

C.7.4 PICS tables for Annex C

C.7.4.1 Link status signals

Item	Feature	Subclause	Value/Comment	Status	Support
LS1	Signal fail	C.1.3.1	Provide a <i>signalFail</i> input	M	Yes []
LS2	Signal degrade	C.1.3.2	Provide a <i>signalDegrade</i> input	M	Yes []
LS3	Unused signal degrade input	C.1.3.2	Unused <i>signalDegrade</i> input to be tied to inactive	M	Yes []
LS4	Mapping	C.1.3.3	Link status signals are mapped to service primitives	M	Yes []

C.7.4.2 GFP cHEC

Item	Feature	Subclause	Value/Comment	Status	Support
CH1	Support GFP	C.2.2	Support GFP	O	Yes []
CH2	(Null) cHEC	C.2.2	GRS to insert zeros if no cHEC is calculated	CH1:M	Yes []
CH3	cHEC calculation	C.2.2	GRS calculation of cHEC	CH1:M	Yes []

C.7.4.3 SRS and GRS using 8-bit SPI-3

Item	Feature	Subclause	Value/Comment	Status	Support
SA1	Rate	C.3.1.2	PHYs must support operation at SPI-3 rates	C1:M	Yes []
SA2	<i>TDAT</i>	C.3.2	GRS must map valid <i>LENGTH</i> data to the PCI field	C1:M	Yes []
SA3	<i>RVAL</i>	C.3.4 C.3.1.2	Disregard signals when <i>RVAL</i> is low	C1:M	Yes []
SA4	<i>RENB</i>	C.3.4 C.3.1.2	Monitor <i>RENB</i> to determine when signals are valid	C1:M	Yes []
SA5	<i>RERR</i>	C.3.4 C.3.1.2	Assert <i>RERR</i> only when <i>REOP</i> is asserted	C1:M	Yes []
SA6	<i>RSX</i>	C.3.4 C.3.1.2	<i>RVAL</i> to be low when <i>RSX</i> is high	C1:M	Yes []
SA7	Mapping of service primitives	C.3.2	SPI-3 signals to be mapped to service primitives	C1:M	Yes []
SA8	PHY_DATA.request	C.3.2.1.2	Map signals to PHY_DATA.request	C1:M	Yes []
SA9	Functional specifications	C.3.4	Meet OIF functional specifications	C1:M	Yes []
SA10	Electrical specifications	C.3.5	Meet OIF electrical specifications	C1:M	Yes []

C.7.4.4 SRS and GRS using 32-bit SPI-3

Item	Feature	Subclause	Value/Comment	Status	Support
SB1	Rate	C.4.1.2	PHYs must support operation at SPI-3 rates	C2:M	Yes []
SB2	<i>RVAL</i>	C.4.4 C.4.1.3	Monitor <i>RVAL</i> to determine when signals are valid	C2:M	Yes []
SB3	<i>RENB</i>	C.4.4 C.4.1.3	Monitor <i>RENB</i> to determine when signals are valid	C2:M	Yes []
SB4	<i>RERR</i>	C.4.4 C.4.1.3	Assert <i>RERR</i> only when <i>REOP</i> is asserted	C2:M	Yes []
SB5	<i>RSX</i>	C.4.4 C.4.1.3	<i>RVAL</i> to be low when <i>RSX</i> is high	C2:M	Yes []
SB6	Mapping of service primitives	C.4.2	SPI-3 signals to be mapped to service primitives	C2:M	Yes []
SB7	PHY_DATA.request	C.4.2.1.2	Map signals to PHY_DATA.request	C2:M	Yes []

Item	Feature	Subclause	Value/Comment	Status	Support
SB8	PHY_DATA.indication	C.4.2.2	<i>Map PHY_DATA.indication to signals</i>	C2:M	Yes []
SB9	Functional specifications	C.4.4	Meet OIF functional specifications	C2:M	Yes []
SB10	Electrical specifications	C.4.5	Meet OIF electrical specifications	C2:M	Yes []

C.7.4.5 SRS and GRS using SPI-4.1

Item	Feature	Subclause	Value/Comment	Status	Support
SC1	Mapping of service primitives	C.5.2	SPI-4.1 signals to be mapped to service primitives	C3:M	Yes []
SC2	Functional specifications	C.5.4	Meet OIF functional specifications	C3:M	Yes []
SC3	Electrical specifications	C.4.5	Meet OIF electrical specifications	C3:M	Yes []

C.7.4.6 SRS and GRS using SPI-4.2

Item	Feature	Subclause	Value/Comment	Status	Support
SD1	Mapping of service primitives	C.6.2	SPI-4.1 signals to be mapped to service primitives	C4:M	Yes []
SD2	Functional specifications	C.6.4	Meet OIF functional specifications	C4:M	Yes []
SD3	Electrical specifications	C.6.5	Meet OIF electrical specifications	C4:M	Yes []

Annex D

(normative)

SNMP MIB definitions

D.1 Introduction

This clause defines a portion of the management information base (MIB) for use with network management protocols in the Internet community. In particular, it defines objects for managing RPR interfaces.

Management information is viewed as a collection of managed objects, residing in a virtual information store, termed the *MIB*. Collections of related objects are defined in MIB modules. These modules are written using an adapted subset of the International Organization for Standardization (ISO) Abstract Syntax Notation One, ASN.1 (1988), defined in the Structure of Management Information (SMI).

D.2 The SNMP management framework

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of IETF RFC 3410 [B12].

Managed objects are accessed via a virtual information store, termed the *management information base* or *MIB*. Objects in the MIB are defined using the mechanisms defined in the SMI.

This memo specifies a MIB module that is compliant to the SMIV2, which is described in STD 58, RFC 2578; STD 58, RFC 2579; and STD 58, RFC 2580. A MIB conforming to the SMIV1 can be produced through the appropriate translations, although this is not recommended. The resulting translated MIB would be semantically equivalent, except where objects or events are omitted because no translation is possible (e.g., use of Counter64, use of BITS). Some machine-readable information in SMIV2 will be converted into textual descriptions in SMIV1 during the translation process. However, this loss of machine-readable information is not considered to change the semantics of the MIB. The loss of objects and events, though, would be significant.

D.3 Security considerations

There are managed objects defined in the RPR MIB that have a MAX-ACCESS class of read-write. Such objects may be considered sensitive or vulnerable in some network environments. The support for SET operations in a non-secure environment without proper protection can have a negative effect on network operations.

The following are the tables of the RPR MIB and their sensitivity/vulnerability:

- a) rprIfTable contains read-write parameters that are used for RPR MAC configurations. If the table is not protected, an attack may result in substantial vulnerabilities, or in improper RPR operation.
- b) rprIfStatsControlTable controls the clearing of interval statistics counters. Set operations on rprIfStatsControlTable will cause some or all of the interval counters to be cleared.
- c) rprSpanProtectionTable controls the RPR protection protocol. Improper setting of the table parameters may result in wrong protection behaviors, or in false protection events.

- d) rprFairnessTable controls the RPR fairness protocol performance. Improper setting of the table parameters may result in poor fairness performance on the RPR ring.
- e) rprOamTable controls the RPR OAM operations. Improper setting of the table parameters may result in undesired or flooded OAM frames on the ring.
- f) rprSasSummaryTable contains read-write parameters that are used for RPR MAC SAS configuration. If the table is not protected, an attack may result in substantial vulnerabilities or in improper RPR operation. In particular:
 - Setting rprSasSummaryAgingTime changes the rate at which SDB entries are removed.
 - Setting rprSasSummarySdbPurgeCmd (and either rprSasSummarySdbPurgeScopeSingle or rprSasSummarySdbPurgeScopeMap) will cause some subset of entries to be removed from the SDB.
- g) rprSasStaticUcastTable controls the SAS table containing static association entries for unicast MAC addresses configured by management. Improper setting of the table parameters may result in incorrect target addresses being applied to client frames.
- h) rprSasStaticMcastTable controls the SAS table containing static association entries for multicast MAC addresses configured by management. Improper setting of the table parameters may result in incorrect multicast scoping applied to client frames.

Table D.1 lists the writeable attributes for the SAS optional sublayer of the MAC.

Table D.1—RPR MIB SAS read-write and read-create attributes

Table	Attribute	Access
rprSasSummaryTable	rprSasSummaryAgingTime	Read-write
	rprSasSummarySdbPurgeCmd	Read-write
	rprSasSummarySdbPurgeScopeSingle	Read-write
	rprSasSummarySdbPurgeScopeMap	Read-write
rprSasStaticUcastTable	rprSasStaticUcastVidOnly	Read-create
	rprSasStaticUcastTargetAddr	Read-create
	rprSasStaticUcastRowStatus	Read-create
rprSasStaticMcastTable	rprSasStaticMcastHopsRinglet0	Read-create
	rprSasStaticMcastHopsRinglet1	Read-create
	rprSasStaticMcastExtend	Read-create
	rprSasStaticMcastRowStatus	Read-create

Some of the readable objects in this MIB module (i.e., objects with a MAX-ACCESS other than not-accessible) may be considered sensitive or vulnerable in some network environments. It is thus important to control even GET and/or NOTIFY access to these objects and possibly to even encrypt the values of these objects when sending them over the network via SNMP.

SNMP versions prior to SNMPv3 did not include adequate security. Even if the network is secure (for example, by using IPSec), there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB module.

It is RECOMMENDED that implementers consider the security features as provided by the SNMPv3 framework (see [B12], section 8), including full support for the SNMPv3 cryptographic mechanisms (for authentication and privacy).

Further, deployment of SNMP versions prior to SNMPv3 is NOT RECOMMENDED. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security. It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.

D.4 MIB Structure

This subclause describes the structure of the RPR MIB.

Attributes described in the MIB contain a “REFERENCE” marker identifying the subclause number and the RPR variable name.

D.4.1 Structure of the MIB

The MIB provides objects to configure and manage the RPR MAC, as defined in this standard. The structure is described in Table D.2.

Table D.2—Structure of the MIB

Grouping	Table	Contents
Basic tables	rprIfTable	Contains the configuration and status information for the RPR interface. The table is indexed by ifIndex.
Span tables	rprSpanTable	Contains the configuration and status information for each span interface of the RPR interface. The table is indexed by ifIndex and the span ID (East/West).
	rprSpanProtectionTable	The RPR interface Span protection management and status table.
Protocol tables	rprTopoImageTable	Reports the topology and protection database. The table is indexed by ifIndex and the MAC address of the station in the topology
	rprFairnessTable	Contains the configuration parameters of RPR fairness. The table is indexed by ifIndex and ringlet ID.
	rprOamTable	Allows for basic OAM test functions (echo and flush). This table is indexed by ifIndex.
Statistics control tables	rprIfStatsControlTable	The RPR statistics management table, controls the collection, duration, clearance, and status of the MAC statistics. This table is indexed by ifIndex

Table D.2—Structure of the MIB (continued)

Grouping	Table	Contents
Running counters tables	rprSpanCountersStatsTable	Span frame and byte counters
	rprClientCountersStatsTable	Client frame and byte counters
	rprSpanErrorCountersStatsTable	Span error counters
Current interval tables	rprSpanCountersCurrentTable	Span frame and byte counters
	rprClientCountersCurrentTable	Client frame and byte counters
	rprSpanErrorCountersCurrentTable	Span error counters
Previous intervals tables	rprSpanCountersIntervalTable	Span frame and byte counters
	rprClientCountersIntervalTable	Client frame and byte counters
	rprSpanErrorCountersIntervalTable	Span error counters
Intervals total tables	rprSpanCountersDayTable	Span frame and byte counters
	rprClientCountersDayTable	Client frame and byte counters
	rprSpanErrorCountersDayTable	Span error counters
Sas tables	rprSasSummaryTable	Contains the count of the dynamic and static (including unicast and multicast) entries along with the aging time.
	rprSasDbTable	Contains the entry information of the SDB.
	rprSasStaticUcastTable	Contains the statically configured SDB unicast entries.
	rprSasStaticMcastTable	Contains the statically configured multicast MAC address entries.
	rprSasCountersStatsTable	Contains the SAS counters.

D.5 Relationship to other MIBs

A system implementing the RPR MIB shall also implement (at least) the “interfaces” group defined in IETF RFC 2863.

The “interfaces” group defines a portion of the management information base (MIB) for use with network management protocols in the Internet community. In particular, it describes managed objects used for managing Network Interfaces.

D.5.1 Relationship to the Interfaces MIB

The Interfaces MIB (IETF RFC 2863) requires that any MIB that is an adjunct of the Interface MIB clarify specific areas within the Interface MIB. These areas were intentionally left vague in the Interfaces MIB to avoid over-constraining the MIB, thereby precluding management of certain media types.

Implicit in this MIB is the notion of an RPR MAC interface. Each RPR MAC interface is associated with one interface of the “interfaces” group (one row in the ifTable). Each RPR MAC interface is uniquely identified by an interface number (ifIndex). This interface is layered over the interfaces representing the RS

attachments to the PHYs. Note that the PHY may contain multiple interfaces (e.g., GFP, SONET VT, SONET PATH, SONET Section/Line).

D.5.1.1 Layering model

This annex assumes the interpretation of the Interfaces Group to be in accordance with IETF RFC 2863, which states that the ifTable contains information on the managed resource's interfaces and that each sublayer below the internetwork layer of a network interface is considered an interface.

D.5.1.2 IfStackTable

Each RPR interface represents the “top” interface of an RPR MAC. Each RPR interface will be layered on top of two interface stacks, one for each of the east and west spans.

D.5.1.3 Specific Interface MIB Objects

Table D.3 provides specific implementation guidelines for applying the interface group objects to RPR media. See 13.3.1 for specifics on interface state.

Table D.3—Specific interface MIB objects

Object	Guideline
ifIndex	Each RPR interface is represented by an ifEntry. The rprIfTable in this MIB module is indexed by rprIfIndex. The interface identified by a particular value of rprIfIndex is the same interface as identified by the same value of ifIndex.
ifDescr	The RPR interface description is “RPR interface”.
ifType	The RPR ifType shall be used. To be assigned by IANA.
ifMtu	The value of this object MUST reflect the actual MTU in use on the interface whether it matches the standard MTU or not.
ifSpeed ifHighSpeed	Represents the current operational speed of the interface in bits (or millions of bits for high) per second. Refer to 13.3.1.
ifPhysAddress	The RPR MAC address.
ifAdminStatus	Write access is not required. Support for “testing” is not required. Refer to 13.3.1.1
ifOperStatus	The operational state of the interface. Support for “testing” is not required. The value “dormant” has no meaning for an RPR interface.
ifLastChange	Refer to [RFC2863].
ifInOctets HCInOctets	rprClientStatsInUcastClassAOctets + rprClientStatsInUcastClassBCirOctets + rprClientStatsInUcastClassBEirOctets + rprClientStatsInUcastClassCOctets + rprClientStatsInMcastClassAOctets + rprClientStatsInMcastClassBCirOctets + rprClientStatsInMcastClassBEirOctets + rprClientStatsInMcastClassCOctets
ifInUcastPkts ifHCInUcastPkts	rprClientStatsInUcastClassAFrames + rprClientStatsInUcastClassBCirFrames+ rprClientStatsInUcastClassBEirFrames+ rprClientStatsInUcastClassCFrames

Table D.3—Specific interface MIB objects (continued)

Object	Guideline
ifInDiscards	The number of frames discarded on transmission to the client. Not mapped to any statistics.
ifInErrors	The number of frames to the client discarded due to transmission errors. Not mapped to any statistics.
ifInUnknownProtos	Not used, replaced by this MIB module.
ifOutOctets ifHCOutOctets	rprClientStatsOutUcastClassAOctets + rprClientStatsOutUcastClassBCirOctets + rprClientStatsOutUcastClassBEirOctets + rprClientStatsOutUcastClassCOctets + rprClientStatsOutMcastClassAOctets + rprClientStatsOutMcastClassBCirOctets + rprClientStatsOutMcastClassBEirOctets + rprClientStatsOutMcastClassCOctets
ifOutUcastPkts ifHCOutUcastPkts	rprClientStatsOutUcastClassAFrames + rprClientStatsOutUcastClassBCirFrames+ rprClientStatsOutUcastClassBEirFrames+ rprClientStatsOutUcastClassCFrames
ifOutDiscards	The number of frames transmitted by the client that were chosen to be discarded without them being in error. Not mapped to any statistics.
ifOutErrors	The number of frames transmitted by the client that were discarded because of transmission errors. Not mapped to any statistics.
ifName	Locally significant textual name for the interface.
ifInMulticastPkts ifHCInMulticastPkts	rprClientStatsInMcastClassAFrames + rprClientStatsInMcastClassBCirFrames + rprClientStatsInMcastClassBEirFrames + rprClientStatsInMcastClassCFrames - rprClientStatsInBcastFrames
ifInBroadcastPkts ifHCInBroadcastPkts	rprClientStatsInBcastFrames
ifOutMulticastPkts ifHCOutMulticastPkts	rprClientStatsOutMcastClassAFrames + rprClientStatsOutMcastClassBCirFrames + rprClientStatsOutMcastClassBEirFrames + rprClientStatsOutMcastClassCFrames - rprClientStatsOutBcastFrames
ifOutBroadcastPkts ifHCOutBroadcastPkts	rprClientStatsOutBcastFrames
ifLinkUpDownTrapEnable	Indicates whether linkUp/linkDown traps should be generated for this interface. Default is “disabled.” Refer to [RFC2863].
ifSpeed ifHighSpeed	Represents the current operational speed in millions of bits per second.

Table D.3—Specific interface MIB objects (continued)

Object	Guideline
ifPromiscuousMode	Indicates if this interface only accepts packets/frames that are addressed to this station, or accepts all packets/frames transmitted on the media. In this standard, this is reported based on the value of the <i>myRxfilter</i> parameter, defined in 7.2.2. It has values of RX_BASIC and RX_FLOOD. If <i>myRxfilter</i> is set to flood, then ifPromiscuousMode is reported as true, otherwise false. Refer to [RFC2863].
ifConnectorPresent	Always false.
ifAlias	Allows a network manager to give interface its own unique name, irrespective of any interface-stack relationship. Further, the ifAlias name is nonvolatile, and thus an interface must retain its assigned ifAlias value across reboots, even if an agent chooses a new ifIndex value for the interface
ifCounterDiscontinuityTime	The value of sysUpTime on the most recent occasion at which any one or more of the Client Stats suffered a discontinuity. If no such discontinuities have occurred since the last reinitialization of the local management subsystem, then this object contains a zero value.
ifStackHigherLayer ifStackLowerLayer ifStackStatus	Refer to D.5.1.1.
ifRcvAddressAddress	Indicates the addresses for which the system will accept packets/frames Refer to [RFC2863].
ifRcvAddressStatus	Used to create and delete rows in the ifRcvAddressTable Refer to [RFC2863].
ifRcvAddressType	Indicates the storage type (volatile or non-volatile) of the entry Refer to [RFC2863].

D.5.2 Relationship to PHY MIBs

In addition to the RPR MAC MIB, there are different MIBs for each of the different types of PHYs defined for RPR in Clause 8. For example, the SONET MIB is defined in IETF RFC 2558 [B8].

There is no defined relationship between these MIBs.

D.6 Definitions for the RPR MIB

```
IEEE-802DOT17-RPR-MIB DEFINITIONS ::= BEGIN
IMPORTS

    MODULE-IDENTITY, OBJECT-TYPE, Integer32, Counter32,
    Counter64, Unsigned32
        FROM SNMPv2-SMI
    MODULE-COMPLIANCE, OBJECT-GROUP
        FROM SNMPv2-CONF
    MacAddress, TimeStamp, TEXTUAL-CONVENTION, TruthValue, DateAndTime,
    RowStatus
        FROM SNMPv2-TC
    InterfaceIndex, InterfaceIndexOrZero
        FROM IF-MIB
```

```
SnmpAdminString
    FROM SNMP-FRAMEWORK-MIB
InetAddressType, InetAddress
    FROM INET-ADDRESS-MIB
HCPerfTotalCount, HCPerfIntervalCount, HCPerfCurrentCount
    FROM HC-PerfHist-TC-MIB
VlanIdOrNone
    FROM Q-BRIDGE-MIB
;
```

```
ieee802dot17rprMIB MODULE-IDENTITY
LAST-UPDATED "201010091200Z" -- 9 October 2010 12:00:00 UTC
ORGANIZATION "IEEE 802.17 Working Group"
CONTACT-INFO "stds-802-17@ieee.org"
DESCRIPTION
    "The resilient packet ring MIB for IEEE 802.17.

    Copyright (C) IEEE (2010).
    This version of this MIB module
    is published as Annex D.6 of IEEE Std 802.17.
    See the standard itself for full legal notices."
```

-- Revision history

```
REVISION
    "201010091200Z" -- 9 October 2010 12:00:00 UTC
DESCRIPTION
    "Fourth published version.

    Incorporates editorial corrections and minor MIB specification
    issues including incorrect choices for MIB only definitions
    (e.g., the range for the various TimeElapsed variables),
    inconsistencies between the MIB and main document clauses
    (e.g., the units for rprIfAtdTimer), and adding or correcting
    references.

    Deprecates rprIfProtectionWTR and replaces it with
        rprSpanProtectionWTR and rprSpanProtectionKeepaliveTimeout.
    Deprecates rprIfKeepaliveTimeout and replaces it with
        rprSpanProtectionKeepaliveTimeout.

    Added rprIfTopoStabilityTimer.
    Added additional defect values for rprIfCurrentStatus.
    Added rprSpanStatsInOamEchoReqFrames, rprSpanStatsInOamEchoRspFrames,
        rprSpanStatsOutOamEchoReqFrames, rprSpanStatsOutOamEchoRspFrames."
```

```
REVISION
    "200903171200Z" -- 17 March 2009 12:00:00 UTC
DESCRIPTION
    "Third published version.

    Includes 802.17c Protected Inter-ring Connected (PIRC).
```

This adds:

```
TEXTUAL-CONVENTIONS - RprPircProtectionStatus.
Tables - rprPircConfTable, rprPircSummaryTable.
Module Compliances - rprModulePircCompliance
BITS values - rprIfMacOperModes (pircEnabled, pircPrimary,
    pircRevertive, pircLoadbalancing),
    rprTopoImageStatus (receivedPircUser).
```

```
Objects - rprSpanStatsInOamPircStatusFrames,
          rprSpanStatsOutOamPircStatusFrames."
```

REVISION

"200611141200Z" -- 14 November 2006 12:00:00 UTC

DESCRIPTION

"Second published version.
Includes 802.17b Spatially Aware Sublayer (SAS)."

This adds:

```
TEXTUAL-CONVENTIONS - RprSasDbIdValue
Tables - rprSasSummaryTable, rprSasDbTable,
          rprSasStaticUcastTable, rprSasStaticMcastTable,
          rprSasCountersStatsTable.
Module Compliances - rprModuleSasCompliance
BITS values - rprIfMacOperModes (sasEnabled, relaxedTransmission,
                  permissiveTransmission and sasVlanAware),
                  rprTopoImageStatus (receivedSasUser)
Objects - rprSpanStatsInOamSasNotifyFrames,
          rprSpanStatsOutOamSasNotifyFrames."
```

REVISION

"200404211200Z" -- 21 Apr 2004 12:00:00 UTC

DESCRIPTION

"First published version."

```
::= {iso std(0) iso8802(8802) ieee802dot17(17) ieee802dot17mibs(1) 1}
```

```
--  
-- Textual Conventions used in this MIB  
--
```

RprSpan ::= TEXTUAL-CONVENTION

STATUS current

DESCRIPTION

"Indicates the span interface of the RPR MAC. Each RPR span is connected to both ringlet0 and ringlet1.
The east span receives from ringlet1 and transmits through ringlet0.
The west span receives from ringlet0 and transmits through ringlet1."

```
SYNTAX INTEGER {
    east      (1),
    west      (2)
}
```

RprProtectionStatus ::= TEXTUAL-CONVENTION

STATUS current

DESCRIPTION

"Indicates the current protection status of the RPR MAC span.
The status values are (arranged in ascending priority order) :

```
noRequest
No protection request on the span, the protection status is idle.
```

waitToRestore

The condition for an automatic protection was cleared and the span is engaged in a wait to restore period.

manualSwitch

A user initiated manual switch (via the rprSpanProtectionCommand) on the span.

signalDegraded

An automatically-detected status which causes protection on a span when a media signal degrade is detected due to excessive BER.

signalFailed

An automatically-detected status which causes protection on a span when a media signal failure is detected.

forcedSwitch

A user initiated forced switch (via the rprSpanProtectionCommand) on the span."

SYNTAX BITS {

noRequest	(0),
waitToRestore	(1),
manualSwitch	(2),
signalDegraded	(3),
signalFailed	(4),
forcedSwitch	(5)

}

RprOamRinglet ::= TEXTUAL-CONVENTION

STATUS current

DESCRIPTION

"Indicates the ringlet on which the OAM action request/response is sent/replied.

The valid values for this object are:

(1) Default (2) ringlet0 (3) ringlet1 (4) Reverse ringlet.

The value (4) is applicable only for OAM response action."

SYNTAX INTEGER {

default	(1),
ringlet0	(2),
ringlet1	(3),
reverseRinglet	(4)

}

RprRingHopCount ::= TEXTUAL-CONVENTION

DISPLAY-HINT "d"

STATUS current

DESCRIPTION

"A hop count value."

SYNTAX Integer32 (0..255)

RprSasDbIdValue ::= TEXTUAL-CONVENTION

DISPLAY-HINT "d"

STATUS current

DESCRIPTION

"An SDB ID.

This follows the syntax defined in Q-BRIDGE-MIB dot1qFdbId.
 If this MAC interface has a bridging client, this is the
 dot1qFdbId value used in the corresponding bridge relay
 FDB entry.
 If the MAC interface does not have a bridging client, then
 the value used is 1."
 SYNTAX Unsigned32 (1..4094)

```
RprPircProtectionStatus ::= TEXTUAL-CONVENTION
  STATUS current
  DESCRIPTION
    "Indicates the current PIRC protection status of the RPR MAC.
     The status values are (arranged in ascending priority
     order) :

    noRequest
      No PIRC protection request on the MAC, the PIRC protection
      status is idle.

    containment
      The PIRC state preceding protection group member activation
      for inter-ring connectivity. The PIRC containment timer
      prevents frame duplication during PIRC transient events.

    waitToRestore
      The condition for an automatic PIRC protection was cleared
      and the MAC is engaged in a wait to restore period.

    manualSwitch
      A user initiated PIRC manual switch (via the
      rprPircConfProtectionCommand) on the MAC.

    protecting
      The MAC PIRC protection state while the PIRC mate station
      deactivates inter-ring connectivity.

    failure
      The MAC PIRC state during PIRC misconfiguration condition,
      or loss of ring interface (i/f operStatus not up).

    forcedSwitch
      A user initiated PIRC forced switch (via the
      rprPircConfProtectionCommand) on the MAC."
SYNTAX BITS {
  noRequest      (0),
  containment    (1),
  waitToRestore  (2),
  manualSwitch   (3),
  protecting     (4),
  failure        (5),
  forcedSwitch   (6)
}

-- Object groups --
rprObjects          OBJECT IDENTIFIER ::= { ieee802dot17rprMIB 1 }
```

```

rprGeneral          OBJECT IDENTIFIER ::= { rprObjects 1 }
rprProtocols        OBJECT IDENTIFIER ::= { rprObjects 2 }
rprSpanCounters    OBJECT IDENTIFIER ::= { rprObjects 3 }
rprClientCounters  OBJECT IDENTIFIER ::= { rprObjects 4 }
rprSpanErrorCounters OBJECT IDENTIFIER ::= { rprObjects 5 }
rprSas              OBJECT IDENTIFIER ::= { rprObjects 6 }
rprSasObjects       OBJECT IDENTIFIER ::= { rprSas 1 }
rprSasCounters     OBJECT IDENTIFIER ::= { rprSas 2 }
rprPirc             OBJECT IDENTIFIER ::= { rprObjects 7 }
rprPircObjects     OBJECT IDENTIFIER ::= { rprPirc 1 }

-- Conformance

rprConformance      OBJECT IDENTIFIER ::= { ieee802dot17rprMIB 2 }

-- RPR interface table

rprIfTable OBJECT-TYPE
  SYNTAX      SEQUENCE OF RprIfEntry
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION
    "The RPR interface table, extension to the ifTable.

    All read-write attributes in this table are non-volatile,
    i.e., will be retained across system reset."
  ::= { rprGeneral 1 }

rprIfEntry OBJECT-TYPE
  SYNTAX      RprIfEntry
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION
    "One such entry for every interface in the ifTable which
    has an ifType of RPR interface, i.e., rpr(225)."
  INDEX { rprIfIndex }
  ::= { rprIfTable 1 }

RprIfEntry ::= SEQUENCE {
  rprIfIndex           InterfaceIndex,
  rprIfStationsOnRing Unsigned32,
  rprIfReversionMode  TruthValue,
  rprIfProtectionWTR  Unsigned32,
  rprIfProtectionFastTimer Unsigned32,
  rprIfProtectionSlowTimer Unsigned32,
  rprIfAtdTimer        Unsigned32,
  rprIfKeepaliveTimeout Unsigned32,
  rprIfFairnessAggressive TruthValue,
  rprIfPtqSize          Unsigned32,
  rprIfStqSize          Unsigned32,
  rprIfSTQFullThreshold Unsigned32,
  rprIfIdleThreshold   Unsigned32,
  rprIfSesThreshold    Unsigned32,
}

```

```

rprIfWrapConfig          TruthValue,
rprIfJumboFramePreferred TruthValue,
rprIfMacOperModes        BITS,
rprIfRingOperModes       BITS,
rprIfCurrentStatus      BITS,
rprIfLastChange          TimeStamp,
rprIfChanges             Counter32,
rprIfTopoStabilityTimer Unsigned32
}

rprIfIndex OBJECT-TYPE
SYNTAX      InterfaceIndex
MAX-ACCESS   not-accessible
STATUS      current
DESCRIPTION
"The ifIndex of this RPR interface."
REFERENCE
"RFC 2863, ifIndex"
::= { rprIfEntry 1 }

rprIfStationsOnRing OBJECT-TYPE
SYNTAX      Unsigned32 (1..255)
MAX-ACCESS   read-only
STATUS      current
DESCRIPTION
"The number of stations on the RPR ring.
When the operStatus of the interface is down the value is 1."
REFERENCE
"IEEE 802.17 Subclause 11.2.4, numStations"
::= { rprIfEntry 2 }

rprIfReversionMode OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS   read-write
STATUS      current
DESCRIPTION
"The reversion mode of the MAC,
False for non-revertive
True for revertive.

Revertive station will return to idle state after
WTR interval expires.

Default value for reversion mode is true."
REFERENCE
"IEEE 802.17 Subclause 11.2.3, revertive"
DEFVAL { true }
::= { rprIfEntry 3 }

rprIfProtectionWTR OBJECT-TYPE
SYNTAX      Unsigned32 (0..1440)
UNITS       "Seconds"
MAX-ACCESS   read-write
STATUS      deprecated
DESCRIPTION
"Indicates the length of time in seconds, to remain in the
protection state, after the cause of an automatic
protection is removed. This mechanism prevents protection
switch oscillations.

```

Default value for WTR is 10 seconds.

This is deprecated. rprSpanProtectionWTR should be used instead."

REFERENCE

"IEEE 802.17 Subclause 11.2.3, wtr"
DEFVAL { 10 }
 ::= { rprIfEntry 4 }

rprIfProtectionFastTimer OBJECT-TYPE

SYNTAX Unsigned32 (1..20)
UNITS "milliseconds"
MAX-ACCESS read-write
STATUS current

DESCRIPTION

"Indicates the protection messages fast timer value in 1 ms units.
The fast timer is used for protection protocols.

Default value for fast timer protection messages is 10 ms."

REFERENCE

"IEEE 802.17 Subclause 11.2.3, txFastTimeout"
DEFVAL { 10 }
 ::= { rprIfEntry 5 }

rprIfProtectionSlowTimer OBJECT-TYPE

SYNTAX Unsigned32 (1..10)
UNITS "100 milliseconds"
MAX-ACCESS read-write
STATUS current

DESCRIPTION

"Indicates the protection slow timer value in 100 ms units.
The slow timer is used for topology and protection protocols.

Default value for slow timer protection and topology messages
is 1 units of 100 ms, i.e., 100 ms."

REFERENCE

"IEEE 802.17 Subclause 11.2.3, txSlowTimeout"
DEFVAL { 1 }
 ::= { rprIfEntry 6 }

rprIfAtdTimer OBJECT-TYPE

SYNTAX Unsigned32 (1..10)
UNITS "seconds"
MAX-ACCESS read-write
STATUS current

DESCRIPTION

"Indicates the timer period for ATD message transmissions.

Default value is 1 second."

REFERENCE

"IEEE 802.17 Subclause 11.2.3, atdTimerTimeout"
DEFVAL { 1 }
 ::= { rprIfEntry 7 }

rprIfKeepaliveTimeout OBJECT-TYPE

SYNTAX Unsigned32 (2..200)
UNITS "milliseconds"
MAX-ACCESS read-write
STATUS deprecated

DESCRIPTION
 "Indicates the timer to declare keepalive timeout in multiples of 1 millisecond.

The default value is 3 ms.

This is deprecated.
 rprSpanProtectionKeepaliveTimeout should be used instead."

REFERENCE
 "IEEE 802.17 Subclause 11.6.2.2, keepaliveTimeout"

DEFVAL { 3 }
 ::= { rprIfEntry 8 }

rprIfFairnessAggressive OBJECT-TYPE

SYNTAX	TruthValue
MAX-ACCESS	read-write
STATUS	current

DESCRIPTION
 "Indicates whether the selected RPR fairness algorithm is aggressive or conservative.

If true, the selected fairness mode is aggressive.
 If false, the selected fairness mode is conservative.

The default value for fairness aggressive is true."

REFERENCE
 "IEEE 802.17 Subclause 11.2.5, conservativeMode"

DEFVAL { true }
 ::= { rprIfEntry 9 }

rprIfPtqSize OBJECT-TYPE

SYNTAX	Unsigned32
UNITS	"Bytes"
MAX-ACCESS	read-only
STATUS	current

DESCRIPTION
 "The size in bytes of the Primary Transit Queue per ringlet supported by this RPR MAC."

REFERENCE
 "IEEE 802.17 Subclause 7.2.2, sizePtq"
 ::= { rprIfEntry 10 }

rprIfStqSize OBJECT-TYPE

SYNTAX	Unsigned32
UNITS	"Bytes"
MAX-ACCESS	read-only
STATUS	current

DESCRIPTION
 "The size in bytes of the Secondary Transit Queue per ringlet supported by this RPR MAC."

REFERENCE
 "IEEE 802.17 Subclause 7.2.2, sizeStq"
 ::= { rprIfEntry 11 }

rprIfSTQFullThreshold OBJECT-TYPE

SYNTAX	Unsigned32
UNITS	"MTUs"
MAX-ACCESS	read-write
STATUS	current

DESCRIPTION

"A level of STQ occupancy at or above which the STQ is almost full.

This attribute specifies the full threshold location in MTU units below the STQ size.

The range is [stqHighThreshold + mtuSize, sizeSTQ - mtuSize].

The default value for stqFullThreshold is sizeSTQ - 2*mtuSize."

REFERENCE

"IEEE 802.17 Subclause 7.2.2, stqFullThreshold"

DEFVAL { 2 }

::= { rprIfEntry 12 }

rprIfIdleThreshold OBJECT-TYPE

SYNTAX Unsigned32

UNITS "MTUs"

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"Indicates that a MAC is experiencing an adverse rate mismatch. When the free space in the PTQ becomes less than or equal to this value, the incoming link is considered adversely rate mismatched.

This attribute specifies the idle-threshold location from the top of the PTQ buffer, in MTU size.

Default value should be 1, which means (sizePTQ - mtuSize)."

REFERENCE

"IEEE 802.17 Subclause 7.5.3.2, idleThreshold"

DEFVAL { 1 }

::= { rprIfEntry 13 }

rprIfSesThreshold OBJECT-TYPE

SYNTAX Unsigned32 (1..512)

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"Indicates the number of E-RPR errors to declare a second as SES-RPR

The default value is the rounded integer of 0.000001 * (lineRate * advertisementRatio)."

REFERENCE

"IEEE 802.17 Subclause 12.2.2, sesThreshold"

::= { rprIfEntry 14 }

rprIfWrapConfig OBJECT-TYPE

SYNTAX TruthValue

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"Indicates the configured protection mode, steering or wrapping, in a MAC that supports both modes.

If rprIfWrapConfig is 'true', the station will perform wrap protection during ring failures.

If the station detects another station on the ring that

has a protection configuration that conflicts with its own,
the station will generate a defect to higher layers.

Default value for wrap configured is false for steering
only stations, and true for stations that implement wrapping."

REFERENCE

"IEEE 802.17 Subclause 11.2.5, protConfig"
 ::= { rprIfEntry 15 }

rprIfJumboFramePreferred OBJECT-TYPE

SYNTAX TruthValue

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"Indicates the preference of the manager to support jumbo frame
in a MAC that supports jumbo frame.

The station advertises the 'logical and' of rprIfJumboFramePreferred
and jumbo capable bit in the rprIfRingOperModes.

Default value for jumbo frame preferred is true. I.e., station
supports jumbo frame."

REFERENCE

"IEEE 802.17 Subclause 11.2.5, jumboPrefer"
DEFVAL { true }
 ::= { rprIfEntry 16 }

rprIfMacOperModes OBJECT-TYPE

SYNTAX BITS {

strictOrder	(0),
dropBadFcs	(1),
sasEnabled	(2),
relaxedTransmission	(3),
permissiveTransmission	(4),
sasVlanAware	(5),
pircEnabled	(6),
pircPrimary	(7),
pircRevertive	(8),
pircLoadbalancing	(9)

}

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The summary of the Mac operational modes.
if strictOrder is set, the MAC operates in strict order mode.
if dropBadFcs is set, frames with bad FCS are dropped.
if sasEnabled is set, then the SAS layer is present and enabled.
if relaxedTransmission is set, then the MAC is providing
relaxed transmission service.
if permissiveTransmission is set, then the MAC is providing
permissive transmission service.
if sasVlanAware is set, then SAS is VLAN aware.
if pircEnabled is set, then the PIRC layer is present and enabled.
if pircPrimary is set, then this MAC is the primary PIRC protection
group member.
if pircRevertive is set, then this MAC operates in PIRC protection
reversion mode is enabled.
if pircLoadbalancing is set, then this MAC operates in PIRC load-
balancing mode.

NOTE: Only one of the strictOrder, relaxedTransmission, or permissiveTransmission values is reported at any given time."

REFERENCE

"IEEE 802.17 Subclause 6.2, copyBadFcs, enableSAS, and transmissionControl, Subclause 14.2.2 optionSasVlanAware, Subclause 6.2.1 enablePirc, Subclause 15.3.1 pircProv.pircPrimary, pircProv.pircRevertive, pircProv.pircLoadBalancing."
 $::= \{ rprIfEntry 17 \}$

rprIfRingOperModes OBJECT-TYPE

SYNTAX BITS {
 jumboFrames (0),
 wrapProtection (1),
 openRing (2)
 }

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The summary of the ring operational modes collected through the topology discovery protocol.

If at least one station does not support jumbo frames, the jumboFrame bit in this attribute is set to false.
 Only if all stations support jumbo frames, the bit is true.

If at least one station was not configured to wrap, the wrap bit in this attribute is set to false.
 Only if all stations configured to wrap, the bit is true.

If the ring does not complete full loop, the ring is considered openRing, with at least one detected edge."

REFERENCE

"IEEE 802.17 Subclause 11.2.4, jumboType and topoType"
 $::= \{ rprIfEntry 18 \}$

rprIfCurrentStatus OBJECT-TYPE

SYNTAX BITS {
 neighborInconsistency (0),
 duplicateMac (1),
 exceedMaxStations (2),
 excessReservedRate (3),
 lrttIncomplete (4),
 exceedMaxSecMacAddress (5),
 miscabling (6),
 pircLbInconsistency (7),
 pircPrInconsistency (8),
 protectionInconsistency (9),
 topoEntryInvalid (10),
 topoInstability (11)
 }

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"Indicates the current defect status of the RPR interface.

This attribute is used to generate alarms to the management system."

REFERENCE

"IEEE 802.17 Subclause 11.2.9, topoInconsistencyDefect, duplicateSecMacAddressDefect, maxStationsDefect, excessReservedRateDefect, lrttIncompleteDefect,

```

        maxSecMacAddressDefect, miscablingDefect, pircLbDefect,
        pircPrDefect, protMisconfigDefect, topoEntryInvalidDefect,
        and topoInstabilityDefect, respectively."
 ::= { rprIfEntry 19 }

rprIfLastChange OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The value of sysUpTime at the time when any of the following
         attributes changes:
         rprIfStationsOnRing, rprIfReversionMode, rprIfProtectionWTR,
         rprIfProtectionFastTimer, rprIfProtectionSlowTimer, rprIfAtdTimer,
         rprIfKeepaliveTimeout, rprIfFairnessAggressive,
         rprIfSTQFullThreshold, rprIfIdleThreshold,
         rprIfSesThreshold, rprIfWrapConfig, rprIfJumboFramePreferred,
         rprIfMacOperModes, rprIfRingOperModes, rprIfCurrentStatus
         contents of the rprTopoImageEntry."
 ::= { rprIfEntry 20 }

rprIfChanges OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates number of times rprIfLastChange changed.

        The discontinued counter value is indicated
        by the ifCounterDiscontinuityTime value."
 ::= { rprIfEntry 21 }

rprIfTopologyTimer OBJECT-TYPE
    SYNTAX      Unsigned32 (10..100)
    UNITS      "milliseconds"
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "Indicates the timer to wait for stability.

        The default value is 40 ms."
    REFERENCE
        "IEEE 802.17 Subclause 11.6.6.3, stabilityTimeout"
    DEFVAL { 40 }
 ::= { rprIfEntry 22 }

-- RPR statistics management table
--

rprIfStatsControlTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprIfStatsControlEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The RPR statistics management table,
         controls the collection, duration, clearance, and status
         of the MAC statistics.

```

The usage of this table is as follows:

1. In order to clear period of counters (or all periods) of specific counting point (or all counting points) of the RPR MAC, the user should first set the rprIfStatsControlPeriodClear item to specify the period that the user would like to clear, and the rprIfStatsControlCountPointClear item to specify the counting point on which the user would like to perform the statistics clearance.
In case that the user specified 'clearSpecificInterval' in the rprIfStatsControlPeriodClear item, the user should also set the specific interval to clear by setting rprIfStatsControlIntervalClear.
After setting these items, the user can activate the clear operation by setting rprIfStatsControlCommitClear item to 'commit'.
The status of the clear operation (done or failed) is indicated by the returned value of rprIfStatsControlCommitClear item.
2. Setting rprIfStatsControlPeriodClear to 'clearAllIntervals' and rprIfStatsControlCountPointClear to 'clearWest' (for example) will clear all current and previous intervals, and day counters of the west side of the RPR MAC.
3. The IntervalValidData of cleared interval should be set to false.

All read-write attributes in this table are volatile,
i.e., will be cleared on system reset."

`::= { rprGeneral 2 }`

```
rprIfStatsControlEntry OBJECT-TYPE
    SYNTAX          RprIfStatsControlEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "One such entry for every interface in the ifTable which
         has an ifType of RPR interface, i.e., rpr(225)."
    INDEX { rprIfStatsControlIfIndex }
    ::= { rprIfStatsControlTable 1 }
```

```
RprIfStatsControlEntry ::= SEQUENCE {
    rprIfStatsControlIfIndex           InterfaceIndex,
    rprIfStatsControlPeriodClear      INTEGER,
    rprIfStatsControlCountPointClear  INTEGER,
    rprIfStatsControlIntervalClear    Unsigned32,
    rprIfStatsControlCommitClear      INTEGER,
    rprIfStatsControlTimeElapsed     Unsigned32,
    rprIfStatsControlValidIntervals  Unsigned32
}
```

```
rprIfStatsControlIfIndex OBJECT-TYPE
    SYNTAX          InterfaceIndex
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    REFERENCE
        "RFC 2863, ifIndex"
    ::= { rprIfStatsControlEntry 1 }
```

```

rprIfStatsControlPeriodClear OBJECT-TYPE
    SYNTAX      INTEGER {
        idle          (1),
        clearAllIntervals (2),
        clearCurrent (3),
        clearIntervals (4),
        clearSpecificInterval (5)
    }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "This attribute allows for some or all of the
         interval statistics for this RPR MAC to be cleared.
         It does not affect the values of the running counters
         or the counter values reported through the interface MIB.

        clearSpecificInterval clears the interval indicated by
        rprIfStatsControlIntervalClear.

        Default value for period clear is idle."
    DEFVAL { idle }
    ::= { rprIfStatsControlEntry 2 }

rprIfStatsControlCountPointClear OBJECT-TYPE
    SYNTAX      INTEGER {
        clearAll          (1),
        clearWest         (2),
        clearEast         (3),
        clearClient       (4)
    }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "This attribute allows for a specific counting point of the
         RPR MAC or all of the RPR MAC counting points to be cleared
         on rprIfStatsControlPeriodClear request.

        Default value for Interface to clear is clearAll,
         i.e., clear statistics of all interfaces."
    DEFVAL { clearAll }
    ::= { rprIfStatsControlEntry 3 }

rprIfStatsControlIntervalClear OBJECT-TYPE
    SYNTAX      Unsigned32 (1..96)
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The statistics interval number to clear.
         The interval identified by 1 is the most recently completed
         15 minute interval, and interval identified by N is the
         interval immediately preceding the one identified by N-1.

        Setting rprIfStatsControlPeriodClear to clearSpecificInterval will
        clear the interval that is indicated by this object."
    ::= { rprIfStatsControlEntry 4 }

rprIfStatsControlCommitClear OBJECT-TYPE
    SYNTAX  INTEGER {

```

```
        commit      (1), -- write only
        commitDone  (2), -- read only
        commitFailed (3) -- read only
    }
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION
    "Sends a commit with input parameters to specify the
     Period (rprIfStatsControlPeriodClear),
     count point (rprIfStatsControlCountPointClear) and
     specific interval (rprIfStatsControlIntervalClear) to clear.

    If the clear operation succeeds this attribute will have
    the value commitDone(2) when read.
    If the clear operation fails this attribute will have
    the value commitFailed(3) when read."
 ::= { rprIfStatsControlEntry 5 }

rprIfStatsControlTimeElapsed OBJECT-TYPE
    SYNTAX      Unsigned32 (0..910)
    UNITS      "Seconds"
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of seconds, including partial seconds, that
         have elapsed since the beginning of the current
         measurement interval. If, for some reason, such as an
         adjustment in the system's time-of-day clock, the current
         interval exceeds the maximum value, the agent will return
         the maximum value."
 ::= { rprIfStatsControlEntry 6 }

rprIfStatsControlValidIntervals OBJECT-TYPE
    SYNTAX      Unsigned32 (0..96)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of previous 15-minute intervals for
         which data was collected.
        An RPR interface must be capable of supporting at least n
         intervals.
        The minimum value of n is 4. The default of n is 32.
        The maximum value of n is 96.
        The value will be <n> unless the measurement was
        (re-)started within the last (<n>*15) minutes, in which
        case the value will be the number of complete 15
        minute intervals for which the agent has at least
        some data. In certain cases (e.g., in the case
        where the agent is a proxy) it is possible that some
        intervals are unavailable. In this case, this
        interval is the maximum interval number for
        which data is available."
 ::= { rprIfStatsControlEntry 7 }

-- RPR span table
--

rprSpanTable OBJECT-TYPE
```

```

SYNTAX      SEQUENCE OF RprSpanEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "The RPR interface Span table.

    All read-write attributes in this table are non-volatile,
    i.e., will be retained across system reset."
 ::= { rprGeneral 3 }

rprSpanEntry OBJECT-TYPE
SYNTAX      RprSpanEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "One such entry for every span of an RPR interface."
INDEX { rprSpanIfIndex,
        rprSpanId      }
 ::= { rprSpanTable 1 }

RprSpanEntry ::= SEQUENCE {
    rprSpanIfIndex          InterfaceIndex,
    rprSpanId                RprSpan,
    rprSpanLowerLayerIfIndex InterfaceIndexOrZero,
    rprSpanTotalRingletReservedRate Unsigned32,
    rprSpanCurrentStatus     BITS,
    rprSpanLastChange       TimeStamp,
    rprSpanChanges           Counter32
}

rprSpanIfIndex OBJECT-TYPE
SYNTAX      InterfaceIndex
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "The ifIndex of this RPR interface."
REFERENCE
    "RFC 2863, ifIndex"
 ::= { rprSpanEntry 1 }

rprSpanId OBJECT-TYPE
SYNTAX      RprSpan
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "Indicates the span interface of the RPR MAC. Each RPR
    span is connected to both ringlet0 and ringlet1.
    The east span receives from ringlet1 and transmits through
    ringlet0.
    The west span receives from ringlet0 and transmits through
    ringlet1."
REFERENCE
    "IEEE 802.17 Subclause 7.2.2, myRi"
 ::= { rprSpanEntry 2 }

rprSpanLowerLayerIfIndex OBJECT-TYPE
SYNTAX      InterfaceIndexOrZero
MAX-ACCESS  read-only
STATUS      current

```

DESCRIPTION

"The ifIndex of interface which is below the RPR layer in this span. A value of zero indicates an interface index that has yet to be determined."

REFERENCE

"RFC 2863, ifIndex"
 ::= { rprSpanEntry 3 }

rprSpanTotalRingletReservedRate OBJECT-TYPE

SYNTAX Unsigned32
UNITS "Mb/s"
MAX-ACCESS read-only
STATUS current

DESCRIPTION

"The total reserved subclassA0 bandwidth on the ringlet.
This variable is used by the fairness module
to determine the total reclaimable bandwidth."

REFERENCE

"IEEE 802.17 Subclause 11.2.4, unreservedRate"
 ::= { rprSpanEntry 4 }

rprSpanCurrentStatus OBJECT-TYPE

SYNTAX BITS {
keepAliveTimeout (0),
muscabling (1),
phyLinkDegrade (2),
phyLinkFail (3)
}
MAX-ACCESS read-only
STATUS current

DESCRIPTION

"Indicates the current status of the RPR span.
this attribute is used to generate alarm to the management system."

REFERENCE

"IEEE 802.17 Subclause 11.2.3, keepaliveTime, 11.2.9,
muscablingDefect, and 8.2.3.2, LINK_STATUS"
 ::= { rprSpanEntry 5 }

rprSpanLastChange OBJECT-TYPE

SYNTAX TimeStamp
MAX-ACCESS read-only
STATUS current

DESCRIPTION

"The value of sysUpTime at the time when any of the following attributes changes:
rprSpanCurrentStatus,
rprSpanProtectionNeighborValid, rprSpanProtectionHoldOffTimer,
rprSpanProtectionCommand, rprSpanProtectionCount."

::= { rprSpanEntry 6 }

rprSpanChanges OBJECT-TYPE

SYNTAX Counter32
MAX-ACCESS read-only
STATUS current

DESCRIPTION

"Indicates number of times rprSpanLastChange changed.

The discontinued counter value is indicated

```

        by the ifCounterDiscontinuityTime value."
 ::= { rprSpanEntry 7 }

--  

-- RPR span protection table  

--  

rprSpanProtectionTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprSpanProtectionEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The RPR interface Span protection management table.

        All read-write attributes in this table are non-volatile,
        i.e., will be retained across system reset."
 ::= { rprGeneral 4 }

rprSpanProtectionEntry OBJECT-TYPE
    SYNTAX      RprSpanProtectionEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "One such entry for every span of an RPR interface."
    INDEX { rprSpanProtectionIfIndex,
            rprSpanProtectionSpan }
    ::= { rprSpanProtectionTable 1 }

RprSpanProtectionEntry ::= SEQUENCE {
    rprSpanProtectionIfIndex          InterfaceIndex,
    rprSpanProtectionSpan           RprSpan,
    rprSpanProtectionNeighborValid   TruthValue,
    rprSpanProtectionHoldOffTimer    Unsigned32,
    rprSpanProtectionCommand         INTEGER,
    rprSpanProtectionCount          Counter32,
    rprSpanProtectionDuration        Counter32,
    rprSpanProtectionLastActivationTime TimeStamp,
    rprSpanProtectionWTR             Unsigned32,
    rprSpanProtectionKeepaliveTimeout Unsigned32
}

rprSpanProtectionIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    REFERENCE
        "RFC 2863, ifIndex"
    ::= { rprSpanProtectionEntry 1 }

rprSpanProtectionSpan OBJECT-TYPE
    SYNTAX      RprSpan
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The Span for this entry."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.2. myRi"

```

```

 ::= { rprSpanProtectionEntry 2 }

rprSpanProtectionNeighborValid OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Indicates whether the neighbor address is active or former.
         true value for active, false for former."
    REFERENCE
        "IEEE 802.17 Subclause 11.2.5, lastNeighborMac"
    ::= { rprSpanProtectionEntry 3 }

rprSpanProtectionHoldOffTimer OBJECT-TYPE
    SYNTAX      Unsigned32 (0..20)
    UNITS      "10 milliseconds"
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The period that RPR gives to a lower layer to perform
         protection before RPR activates its protection mechanism.

        Indicates the protection holdoff timer in 10 ms units.

        The default value is 0 milliseconds (no holdoff)."
    REFERENCE
        "IEEE 802.17 Subclause 11.2.3, holdOffTimeout"
    DEFVAL { 0 }
    ::= { rprSpanProtectionEntry 4 }

rprSpanProtectionCommand OBJECT-TYPE
    SYNTAX      INTEGER {
        idle          (1),
        manualSwitch (2),
        forcedSwitch (3)
    }
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
        "The protection mode requested by management for the local
         station that can affect the protection status of the RPR
         station, according to the set of rules describing the RPR
         protection.

        When read, this object returns the last command written
        unless it has been preempted, or idle if no command has been
        written to this interface span since initialization.

        There is no pending of commands, that is if a command has
        been preempted by a failure, when the failure clears the
        command is not executed.

        If the command cannot be executed because an equal or
        higher priority request is in effect, an error is returned.

        Writing idle to a span that has no pending protection
        command, has no affect. An idle clears an active WTR state."

```

The protection commands (arranged in ascending priority order) are:

idle

This command clears the protection for the specified interface span.

This value should be returned by a read request when no protection command has been written to the object.

manualSwitch

A protection command on each end of a specified span. This command does not have precedence over automatic protection, and therefore it cannot preempt an existing automatic protection request.

forcedSwitch

A command on each end of a specified span. This command has precedence over automatic protection, and therefore it can preempt an existing automatic protection request.

Default value of protection command is idle."

REFERENCE

"IEEE 802.17 Subclause 11.2.3, adminRequestProtection"

DEFVAL { idle }

::= { rprSpanProtectionEntry 5 }

rprSpanProtectionCount OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The number of transitions from idle state to active protection state.

The discontinued counter value is indicated by the ifCounterDiscontinuityTime value."

REFERENCE

"IEEE 802.17 Subclause 11.6.5.5, IndicateEdgeState()"

::= { rprSpanProtectionEntry 6 }

rprSpanProtectionDuration OBJECT-TYPE

SYNTAX Counter32

UNITS "seconds"

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The total amount of time protection was active on the span interface.

The discontinued counter value is indicated by the ifCounterDiscontinuityTime value."

REFERENCE

"IEEE 802.17 Subclause 11.6.5.5, IndicateEdgeState()"

::= { rprSpanProtectionEntry 7 }

rprSpanProtectionLastActivationTime OBJECT-TYPE

SYNTAX TimeStamp

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The value of sysUpTime at the time of the last protection activation."

REFERENCE

"IEEE 802.17 Subclause 11.6.5.5, IndicateEdgeState()
 ::= { rprSpanProtectionEntry 8 }

rprSpanProtectionWTR OBJECT-TYPE

SYNTAX Unsigned32 (0..1440)
UNITS "Seconds"
MAX-ACCESS read-write
STATUS current

DESCRIPTION

"Indicates the length of time in seconds, to remain in the protection state, after the cause of an automatic protection is removed. This mechanism prevents protection switch oscillations.

Default value for WTR is 10 seconds.

rprSpanProtectionWTR should be used instead instead of rprIfProtectionWTR, which has been deprecated."

REFERENCE

"IEEE 802.17 Subclause 11.2.3, wtr"
DEFVAL { 10 }
 ::= { rprSpanProtectionEntry 9 }

rprSpanProtectionKeepaliveTimeout OBJECT-TYPE

SYNTAX Unsigned32 (2..200)
UNITS "milliseconds"
MAX-ACCESS read-write
STATUS current

DESCRIPTION

"Indicates the timer to declare keepalive timeout in multiples of 1 millisecond.

The default value is 3 ms."

rprSpanProtectionKeepaliveTimeout should be used instead of rprIfKeepaliveTimeout, which has been deprecated."

REFERENCE

"IEEE 802.17 Subclause 11.6.2.2, keepaliveTimeout"
DEFVAL { 3 }
 ::= { rprSpanProtectionEntry 10 }

--

-- RPR topology table

--

rprTopoImageTable OBJECT-TYPE

SYNTAX SEQUENCE OF RprTopoImageEntry
MAX-ACCESS not-accessible
STATUS current

DESCRIPTION

"A topology map that details the list of stations on the RPR ringlets.

```

        All attributes in this table are volatile,
        i.e., will be cleared on system reset."
 ::= { rprProtocols 1 }

rprTopoImageEntry OBJECT-TYPE
    SYNTAX      RprTopoImageEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Each entry contains information specific to a particular
         station on the ring.

        The table has at least one entry for the station itself,
        this entry indicates zero hops on each ringlet."
INDEX { rprTopoImageIfIndex,
        rprTopoImageMacAddress }
 ::= { rprTopoImageTable 1 }

RprTopoImageEntry ::= SEQUENCE {
    rprTopoImageIfIndex           InterfaceIndex,
    rprTopoImageMacAddress        MacAddress,
    rprTopoImageSecMacAddress1   MacAddress,
    rprTopoImageSecMacAddress2   MacAddress,
    rprTopoImageStationIfIndex   InterfaceIndexOrZero,
    rprTopoImageStationName      SnmpAdminString,
    rprTopoImageInetAddressType  InetAddressType,
    rprTopoImageInetAddress     InetAddress,
    rprTopoImageCapability       BITS,
    rprTopoImageRinglet0Hops     Integer32,
    rprTopoImageRinglet0ReservedRate Unsigned32,
    rprTopoImageRinglet1Hops     Integer32,
    rprTopoImageRinglet1ReservedRate Unsigned32,
    rprTopoImageWestProtectionStatus RprProtectionStatus,
    rprTopoImageWestWeight       Unsigned32,
    rprTopoImageEastProtectionStatus RprProtectionStatus,
    rprTopoImageEastWeight       Unsigned32,
    rprTopoImageStatus          BITS
}

rprTopoImageIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The ifIndex of this RPR interface."
REFERENCE
    "RFC 2863, ifIndex"
 ::= { rprTopoImageEntry 1 }

rprTopoImageMacAddress OBJECT-TYPE
    SYNTAX      MacAddress
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The 48-bit MAC address of the station."
REFERENCE
    "IEEE 802.17 Subclause 11.2.6, macAddress"
 ::= { rprTopoImageEntry 2 }

```

```
rprTopoImageSecMacAddress1 OBJECT-TYPE
    SYNTAX      MacAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The 48-bit first secondary MAC address of the station.

        Default and unused value is FF:FF:FF:FF:FF:FF."
    REFERENCE
        "IEEE 802.17 Subclause 11.2.6, secMac"
    ::= { rprTopoImageEntry 3 }
```

```
rprTopoImageSecMacAddress2 OBJECT-TYPE
    SYNTAX      MacAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The 48-bit second secondary MAC address of the station.

        Default and unused value is FF:FF:FF:FF:FF:FF."
    REFERENCE
        "IEEE 802.17 Subclause 11.2.6, secMac"
    ::= { rprTopoImageEntry 4 }
```

```
rprTopoImageStationIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndexOrZero
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The station ifIndex. A value of zero indicates an
         interface index that has yet to be determined."
    REFERENCE
        "IEEE 802.17 Subclause 11.2.6, interfaceIndex"
    ::= { rprTopoImageEntry 5 }
```

```
rprTopoImageStationName OBJECT-TYPE
    SYNTAX      SnmpAdminString
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The operator assigned station name."
    REFERENCE
        "IEEE 802.17 Subclause 11.2.6, stationName"
    ::= { rprTopoImageEntry 6 }
```

```
rprTopoImageInetAddressType OBJECT-TYPE
    SYNTAX      InetAddressType
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Denotes the address type of the station INET address
         It should be set to 'unknown' if station INET address
         is not known."
    REFERENCE
        "IEEE 802.17 Subclause 11.2.6, managementAddressType"
    ::= { rprTopoImageEntry 7 }
```

```
rprTopoImageInetAddress OBJECT-TYPE
    SYNTAX      InetAddress
```

```

MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "This object contains the value of of the station
  IP (v4 or V6) address.
  The format of this address is specified by the value
  of the corresponding rprTopoImageInetAddressType object.

  If the address type is unknown, then this object contains the
  zero-length OCTET-STRING."
REFERENCE
  "IEEE 802.17 Subclause 11.2.6, managementIpAddr"
::= { rprTopoImageEntry 8 }

rprTopoImageCapability OBJECT-TYPE
SYNTAX        BITS {
  jumboFrames          (0),
  wrapProtection       (1),
  supportsConservativeFairness (2)

}
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "Indicates the capabilities that the MAC supports.

  jumboFrames - Indicates whether the MAC capable of receiving,
  transmitting and transiting jumbo frames.

  wrapProtection - Indicates whether the MAC configured to wrap
  protection.

  supportsConservativeFairness - indicates the MAC supports
  conservative fairness algorithm."
REFERENCE
  "IEEE 802.17 Subclause 11.2.6, jumboPrefer, protConfig,
  and conservativeMode"
::= { rprTopoImageEntry 9 }

rprTopoImageRinglet0Hops OBJECT-TYPE
SYNTAX        Integer32
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "The number of hops to this station through ringlet0.

  The self station is indicated with zero hops.
  A non reachable station is indicated with -1 value."
REFERENCE
  "IEEE 802.17 Subclause 11.2.6, hops"
::= { rprTopoImageEntry 10 }

rprTopoImageRinglet0ReservedRate OBJECT-TYPE
SYNTAX        Unsigned32
UNITS         "Mb/s"
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "The amount of A0 traffic that this station adds on ringlet0,

```

in Mb/s units."

REFERENCE

"IEEE 802.17 Subclause 11.2.6, reservedRate"
 ::= { rprTopoImageEntry 11 }

rprTopoImageRinglet1Hops OBJECT-TYPE

SYNTAX	Integer32
MAX-ACCESS	read-only
STATUS	current

DESCRIPTION

"The number of hops to this station through ringlet1.

The self station is indicated with zero hops.
A non reachable station is indicated with -1 value."

REFERENCE

"IEEE 802.17 Subclause 11.2.6, hops"
 ::= { rprTopoImageEntry 12 }

rprTopoImageRinglet1ReservedRate OBJECT-TYPE

SYNTAX	Unsigned32
UNITS	"Mb/s"
MAX-ACCESS	read-only
STATUS	current

DESCRIPTION

"The amount of A0 traffic that this station adds on ringlet1,
in Mb/s units."

REFERENCE

"IEEE 802.17 Subclause 11.2.6, reservedRate"
 ::= { rprTopoImageEntry 13 }

rprTopoImageWestProtectionStatus OBJECT-TYPE

SYNTAX	RprProtectionStatus
MAX-ACCESS	read-only
STATUS	current

DESCRIPTION

"The current protection status of the West span.
At least one of rprTopoImageEastProtectionStatus and
rprTopoImageWestProtectionStatus must have a value of noRequest."

REFERENCE

"IEEE 802.17 Subclause 11.2.6, spanProtState"
 ::= { rprTopoImageEntry 14 }

rprTopoImageWestWeight OBJECT-TYPE

SYNTAX	Unsigned32 (1..255)
MAX-ACCESS	read-only
STATUS	current

DESCRIPTION

"The weight of the station on the west span.
used for weighted fairness."

REFERENCE

"IEEE 802.17 Subclause 11.2.6, weight"
 ::= { rprTopoImageEntry 15 }

rprTopoImageEastProtectionStatus OBJECT-TYPE

SYNTAX	RprProtectionStatus
MAX-ACCESS	read-only
STATUS	current

DESCRIPTION

"The current protection status of the east span."

```

At least one of rprTopoImageEastProtectionStatus and
rprTopoImageWestProtectionStatus must have a value of noRequest."
REFERENCE
    "IEEE 802.17 Subclause 11.2.6, spanProtState"
::= { rprTopoImageEntry 16 }

rprTopoImageEastWeight OBJECT-TYPE
    SYNTAX      Unsigned32 (1..255)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The weight of the station on the east span.
         Used for weighted fairness."
REFERENCE
    "IEEE 802.17 Subclause 11.2.6, weight"
::= { rprTopoImageEntry 17 }

rprTopoImageStatus OBJECT-TYPE
    SYNTAX      BITS {
                    reachableRinglet0          (0),
                    reachableRinglet1          (1),
                    edgeWest                  (2),
                    edgeEast                  (3),
                    badFcsUser                (4),
                    multichokeUser            (5),
                    sasUser                   (6),
                    pircUser                  (7)
                }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Status of the station:
         the reachability of station through ringlet0 and ringlet1,
         edge status on the east and west spans,
         willingness to receive frames with bad FCS,
         willingness to receive multichoke fairness frames,
         SAS user,
         PIRC user."
REFERENCE
    "IEEE 802.17 Subclause 11.2.6, reachable, protConfig, spanEdge,
     badFcsUser, and multichokeUser, sasUser, pircGroupMember."
::= { rprTopoImageEntry 18 }

-- The RPR Fairness table

rprFairnessTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprFairnessEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A table of RPR Fairness per RPR span.

        All read-write attributes in this table are non-volatile,
        i.e., will be retained across system reset."
::= { rprProtocols 2 }

rprFairnessEntry OBJECT-TYPE

```

```

SYNTAX      RprFairnessEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "A fairness parameters for a particular ringlet of the
     an RPR interface."
INDEX { rprFairnessIfIndex,
         rprFairnessRinglet }
 ::= { rprFairnessTable 1 }

RprFairnessEntry ::= SEQUENCE {
    rprFairnessIfIndex           InterfaceIndex,
    rprFairnessRinglet          INTEGER,
    rprFairnessRingletWeight    Unsigned32,
    rprFairnessReservedRate     Unsigned32,
    rprFairnessMaxAllowed      Unsigned32,
    rprFairnessAgeCoef         Unsigned32,
    rprFairnessRampCoef        Unsigned32,
    rprFairnessLpCoef          Unsigned32,
    rprFairnessAdvertisementRatio Unsigned32,
    rprFairnessMcffReportCoef  Unsigned32,
    rprFairnessActiveWeightsCoef Unsigned32,
    rprFairnessSTQHighThreshold Unsigned32,
    rprFairnessSTQMedThreshold Unsigned32,
    rprFairnessSTQLowThreshold Unsigned32,
    rprFairnessRateHighThreshold Unsigned32,
    rprFairnessRateLowThreshold Unsigned32,
    rprFairnessResetWaterMarks INTEGER,
    rprFairnessSTQHighWaterMark Unsigned32,
    rprFairnessSTQLowWaterMark Unsigned32,
    rprFairnessLastChange       TimeStamp,
    rprFairnessChanges          Counter32
}

rprFairnessIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    REFERENCE
        "RFC 2863, ifIndex"
    ::= { rprFairnessEntry 1 }

rprFairnessRinglet OBJECT-TYPE
    SYNTAX      INTEGER {
        ringlet0    (1),
        ringlet1    (2)
    }
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The ringlet for which this row contains information.

        ringlet0 is the transmission of the east span and the
        reception of the west span,
        ringlet1 is the transmission of the west span and the
        reception of the east span."
    REFERENCE

```

```

    "IEEE 802.17 Subclause 7.2.2, myRi"
 ::= { rprFairnessEntry 2 }

rprFairnessRingletWeight OBJECT-TYPE
  SYNTAX      Unsigned32 (1..255)
  MAX-ACCESS  read-write
  STATUS      current
  DESCRIPTION
    "Weight assigned to fairness to permit the scaling of
     fair rate values among stations on the ringlet.
     This allows one station to use a larger share of available
     capacity than another station without violating fairness
     principles.

The allowed range is [1, 255]. The default value is 1.

A station may constrain the value to a power of two."
REFERENCE
  "IEEE 802.17 Subclause 10.2.2, localWeight"
DEFVAL { 1 }
 ::= { rprFairnessEntry 3 }

rprFairnessReservedRate OBJECT-TYPE
  SYNTAX      Unsigned32
  UNITS      "Mb/s"
  MAX-ACCESS  read-write
  STATUS      current
  DESCRIPTION
    "The amount of A0 traffic that this station adds that ringlet,
     in Mb/s units.

Default value for A0 reserved rate is 0 Mb/s."
REFERENCE
  "IEEE 802.17 Subclause 11.2.5, reservedRate"
DEFVAL { 0 }
 ::= { rprFairnessEntry 4 }

rprFairnessMaxAllowed OBJECT-TYPE
  SYNTAX      Unsigned32
  UNITS      "Mb/s"
  MAX-ACCESS  read-write
  STATUS      current
  DESCRIPTION
    "The maximum value that the station is allowed to transmit
     local excess traffic to the ringlet. The default value
     is the physical ring rate."
REFERENCE
  "IEEE 802.17 Subclause 10.2.2, maxAllowedRate"
 ::= { rprFairnessEntry 5 }

rprFairnessAgeCoef OBJECT-TYPE
  SYNTAX      Unsigned32 (0..4)
  MAX-ACCESS  read-write
  STATUS      current
  DESCRIPTION
    "The coefficient used by the aging procedure to specify
     the relative weights assigned to
     (a) the change in the value of a rate-counter during
         the most recent agingInterval and
     (b) the value of the rate-counter at the end of the previous

```

agingInterval.

The value is interpreted as 2 raised to the power specified by this attribute.

The default value is 2, giving an aging coefficient of 2^{**2} , or 4."

REFERENCE

"IEEE 802.17 Subclause 10.2.2, ageCoef"

DEFVAL { 2 }

::= { rprFairnessEntry 6 }

rprFairnessRampCoef OBJECT-TYPE

SYNTAX Unsigned32 (4..9)

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"The coefficient used for ramping a rate.

The value is interpreted as 2 raised to the power specified by this attribute.

The default value is 6, giving a low-pass coefficient of 2^{**6} , or 64."

REFERENCE

"IEEE 802.17 Subclause 10.2.2, rampUpCoef"

DEFVAL { 6 }

::= { rprFairnessEntry 7 }

rprFairnessLpCoef OBJECT-TYPE

SYNTAX Unsigned32 (4..9)

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"The coefficient used by the low-pass filter procedure to specify the relative weights applied to
(a) the increase in the rate-count value during the most recent agingInterval and
(b) the previous low-pass filtered rate.

The former is assigned a weight of 1 and the latter a weight of (lpCoef-1).

The value is interpreted as 2 raised to the power specified by this attribute.

The default value is 6, giving a low-pass coefficient of 2^{**6} , or 64."

REFERENCE

"IEEE 802.17 Subclause 10.2.2, lpCoef"

DEFVAL { 6 }

::= { rprFairnessEntry 8 }

rprFairnessAdvertisementRatio OBJECT-TYPE

SYNTAX Unsigned32 (1..40)

UNITS "0.00025"

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"The ratio between the link capacity reserved for fairness control messages and the total link capacity.

Default value of fairness advertisement ratio is 5 units of 0.00025, i.e., 0.00125."

REFERENCE

"IEEE 802.17 Subclause 10.2.2, advertisementRatio"

DEFVAL { 5 }

::= { rprFairnessEntry 9 }

rprFairnessMcffReportCoef OBJECT-TYPE

SYNTAX Unsigned32 (8..512)

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"Indicates the number of SCFF advertisingIntervals that elapse between the sending of successive MCFFs. This value allows the interval between sending MCFFs to be established as MCFF_reportingInterval = rprFairnessMcffReportCoef * SCFF_advertisingInterval.

The default value is 10."

REFERENCE

"IEEE 802.17 Subclause 10.2.2, reportCoef"

DEFVAL { 10 }

::= { rprFairnessEntry 10 }

rprFairnessActiveWeightsCoef OBJECT-TYPE

SYNTAX Unsigned32 (8..512)

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"Indicates the number of agingIntervals that elapse between successive computations of activeWeights. This value allows the interval between computations of activeWeights to be established as activeWeightsInterval = activeWeightsCoef * agingInterval.

The default value is 64."

REFERENCE

"IEEE 802.17 Subclause 10.2.2, activeWeightsCoef"

DEFVAL { 64 }

::= { rprFairnessEntry 11 }

rprFairnessSTQHighThreshold OBJECT-TYPE

SYNTAX Unsigned32 (0..1000)

UNITS "Tenth of percent"

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"A level of STQ occupancy at or above which CSFE frames are no longer admitted. Defined only for a dual transit-queue implementation.

This attribute specifies the high threshold location in percentage of the STQ size.

The allowed range is [2*mtuSize, stqFullThreshold -mtuSize].

The default value is 0.25*stqFullThreshold."

REFERENCE

"IEEE 802.17 Subclause 10.2.2, stqHighThreshold"
DEFVAL { 250 }
::= { rprFairnessEntry 12 }

rprFairnessSTQMedThreshold OBJECT-TYPE
SYNTAX Unsigned32 (0..1000)
UNITS "Tenth of percent"
MAX-ACCESS read-write
STATUS current
DESCRIPTION
"A level of buffer occupancy in a dual-queue deployment,
at or above which congestion on the outbound link is declared.

This attribute specifies the medium threshold location
in percentage of the STQ size.
The allowed range is [stqLowThreshold + mtuSize,
stqHighThreshold - mtuSize].
The default value is 0.5 * (stqHighThreshold + stqLowThreshold)."
REFERENCE
"IEEE 802.17 Subclause 10.2.2, stqMedThreshold"
DEFVAL { 187 }
::= { rprFairnessEntry 13 }

rprFairnessSTQLowThreshold OBJECT-TYPE
SYNTAX Unsigned32 (0..1000)
UNITS "Tenth of percent"
MAX-ACCESS read-write
STATUS current
DESCRIPTION
"A level of STQ occupancy at or above which congestion on
the outbound link is imminent.
Defined only for dual transit-queue implementations.

This attribute specifies the low threshold location
in percentage of the STQ size.
The range is [mtuSize, stqHighThreshold -mtuSize].
The default value is 0.5*stqHighThreshold."
REFERENCE
"IEEE 802.17 Subclause 10.2.2, stqLowThreshold"
DEFVAL { 125 }
::= { rprFairnessEntry 14 }

rprFairnessRateHighThreshold OBJECT-TYPE
SYNTAX Unsigned32 (400..990)
UNITS "Tenth of percent"
MAX-ACCESS read-write
STATUS current
DESCRIPTION
"Rate at or above which congestion on the outbound link
is declared.

The range is [0.4 * unreservedRate, 0.99 * unreservedRate].
The default value is 0.95 * unreservedRate."
REFERENCE
"IEEE 802.17 Subclause 10.2.2, rateHighThreshold"
DEFVAL { 950 }
::= { rprFairnessEntry 15 }

rprFairnessRateLowThreshold OBJECT-TYPE

```

SYNTAX      Unsigned32 (500..990)
UNITS      "Tenth of percent"
MAX-ACCESS  read-write
STATUS     current
DESCRIPTION
    "Rate at or above which congestion on the outbound link
     is imminent.

    The range is [0.5 * rateHighThreshold, 0.99 * rateHighThreshold].
    The default value is 0.9 * rateHighThreshold."
REFERENCE
    "IEEE 802.17 Subclause 10.2.2, rateLowThreshold"
DEFVAL { 900 }
 ::= { rprFairnessEntry 16 }

rprFairnessResetWaterMarks OBJECT-TYPE
SYNTAX      INTEGER {
                idle          (1),
                resetWaterMarks (2)
            }
MAX-ACCESS  read-write
STATUS     current
DESCRIPTION
    "Write resetWaterMarks to this attribute to reset
     the STQ water marks to the current occupancy.

    Default value is idle."
DEFVAL { idle }
 ::= { rprFairnessEntry 17 }

rprFairnessSTQHighWaterMark OBJECT-TYPE
SYNTAX      Unsigned32 (0..1000)
UNITS      "Tenth of percent"
MAX-ACCESS  read-only
STATUS     current
DESCRIPTION
    "The highest level of STQ occupancy since the last reset
     of this value, in percentage of the STQ size."
REFERENCE
    "IEEE 802.17 Subclause 10.2.2, stqHighWatermark"
 ::= { rprFairnessEntry 18 }

rprFairnessSTQLowWaterMark OBJECT-TYPE
SYNTAX      Unsigned32 (0..1000)
UNITS      "Tenth of percent"
MAX-ACCESS  read-only
STATUS     current
DESCRIPTION
    "The lowest level of STQ occupancy since the last reset of
     this value, in percentage of the STQ size."
REFERENCE
    "IEEE 802.17 Subclause 10.2.2, stqLowWatermark"
 ::= { rprFairnessEntry 19 }

rprFairnessLastChange OBJECT-TYPE
SYNTAX      TimeStamp
MAX-ACCESS  read-only
STATUS     current
DESCRIPTION

```

```
"The value of sysUpTime at the time when any of the following
attributes changes:
rprFairnessRingletWeight, rprFairnessReservedRate,
rprFairnessMaxAllowed, rprFairnessAgeCoef, rprFairnessRampCoef,
rprFairnessLpCoef, rprFairnessAdvertisementRatio,
rprFairnessSTQHighThreshold, rprFairnessSTQLowThreshold
rprFairnessResetWaterMarks."
 ::= { rprFairnessEntry 20 }

rprFairnessChanges OBJECT-TYPE
SYNTAX      Counter32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "Indicates number of times rprFairnessLastChange changed.

    The discontinued counter value is indicated
    by the ifCounterDiscontinuityTime value."
 ::= { rprFairnessEntry 21 }

-- 
-- The RPR OAM actions table
--

rprOamTable OBJECT-TYPE
SYNTAX      SEQUENCE OF RprOamEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "A table of RPR OAM actions.
    The table is designed to support a one action at a time
    on an RPR interface.

    All read-write attributes in this table are volatile,
    i.e., will be cleared on system reset."
 ::= { rprProtocols 3 }

rprOamEntry OBJECT-TYPE
SYNTAX      RprOamEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "OAM Echo and Flush control for an RPR interface."
INDEX { rprOamIfIndex }
 ::= { rprOamTable 1 }

RprOamEntry ::= SEQUENCE {
    rprOamIfIndex          InterfaceIndex,
    rprOamActionType        INTEGER,
    rprOamDestAddress       MacAddress,
    rprOamRequestRinglet    RprOamRinglet,
    rprOamResponseRinglet   RprOamRinglet,
    rprOamClassOfService    INTEGER,
    rprOamUserData          OCTET STRING,
    rprOamProtected         TruthValue,
    rprOamRequestCount      Unsigned32,
    rprOamTimeout           Unsigned32,
    rprOamControl           INTEGER,
```

```

rprOamResponseCount      Unsigned32,
rprOamAvResponseTime    Unsigned32,
rprOamResponseStatus     INTEGER
}

rprOamIfIndex   OBJECT-TYPE
  SYNTAX          InterfaceIndex
  MAX-ACCESS     not-accessible
  STATUS         current
  DESCRIPTION
    "The ifIndex of this RPR interface."
  REFERENCE
    "RFC 2863, ifIndex"
  ::= { rprOamEntry 1 }

rprOamActionType OBJECT-TYPE
  SYNTAX          INTEGER {
    echo  (1),
    flush (2)
  }
  MAX-ACCESS     read-write
  STATUS         current
  DESCRIPTION
    "The type of OAM action.
     The valid values for this object are:
     (1) echo (2) flush."
  REFERENCE
    "IEEE 802.17 Subclause 12.4.1, OAM_ECHO_REQ, and
     12.4.3, OAM_FLUSH_REQ"
  DEFVAL { echo }
  ::= { rprOamEntry 2 }

rprOamDestAddress OBJECT-TYPE
  SYNTAX          MacAddress
  MAX-ACCESS     read-write
  STATUS         current
  DESCRIPTION
    "The 48-bit MAC address of the destination station of OAM
     session."
  REFERENCE
    "IEEE 802.17 Subclause 12.4.1 and 12.4.3, destination_address"
  ::= { rprOamEntry 3 }

rprOamRequestRinglet OBJECT-TYPE
  SYNTAX          RprOamRinglet
  MAX-ACCESS     read-write
  STATUS         current
  DESCRIPTION
    "The ringlet ID on which the OAM request should be sent."
  REFERENCE
    "IEEE 802.17 Subclause 12.4.1 and 12.4.3, ringlet_id"
  DEFVAL { default }
  ::= { rprOamEntry 4 }

rprOamResponseRinglet OBJECT-TYPE
  SYNTAX          RprOamRinglet
  MAX-ACCESS     read-write
  STATUS         current
  DESCRIPTION

```

```

        "The ringlet ID on which the OAM response should be replied."
REFERENCE
    "IEEE 802.17 Subclause 12.4.1, response_ringlet"
DEFVAL { default }
 ::= { rprOamEntry 5 }

rprOamClassOfService OBJECT-TYPE
SYNTAX      INTEGER {
            classA   (1),
            classB   (2),
            classC   (3)
        }
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION
    "The class-of-service of OAM session frames."
REFERENCE
    "IEEE 802.17 Subclause 12.4.1 and 12.4.3, service_class"
DEFVAL { classC }
 ::= { rprOamEntry 6 }

rprOamUserData OBJECT-TYPE
SYNTAX      OCTET STRING (SIZE (0..1024))
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION
    "The operator assigned user specific data."
REFERENCE
    "IEEE 802.17 Subclause 12.4.1 and 12.4.3, user_data"
DEFVAL { "" }
 ::= { rprOamEntry 7 }

rprOamProtected OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION
    "Indicates whether the OAM action should be protected.

    In a wrapping ring,
    If true, the WE (wrap eligible) bit of the OAM action
    frame is set to 1.
    Otherwise, the WE bit is set to 0.

    In steering ring,
    If true, in case of failure on the requested ringlet between
    the source and the destination stations, the OAM action frame
    will be steered to the alternative ringlet.
    Otherwise, the OAM action frame will be sent through the requested
    ringlet regardless of its protection state."
REFERENCE
    "IEEE 802.17 Subclause 12.4.1 and 12.4.3, mac_protection"
DEFVAL { false }
 ::= { rprOamEntry 8 }

rprOamRequestCount OBJECT-TYPE
SYNTAX      Unsigned32 (0..65535)
MAX-ACCESS  read-write
STATUS      deprecated

```

```

DESCRIPTION
    "Indicates the number of OAM requests to send.

    Default value of OAM request action is 1.

    This has no correspondence with any internal variable
    and is now deprecated."
DEFVAL { 1 }
 ::= { rprOamEntry 9 }

rprOamTimeout OBJECT-TYPE
    SYNTAX      Unsigned32 (1..10000)
    UNITS       "10 usec"
    MAX-ACCESS   read-write
    STATUS       current
    DESCRIPTION
        "Indicates the timer to declare OAM action timeout,
         in 10 usec units.

        Default value of OAM timeout is 500 units of 10 usec (i.e., 5 ms)."
DEFVAL { 500 }
 ::= { rprOamEntry 10 }

rprOamControl OBJECT-TYPE
    SYNTAX      INTEGER {
        idle      (1),
        active    (2),
        abort     (3)
    }
    MAX-ACCESS   read-write
    STATUS       deprecated
    DESCRIPTION
        "Control of an OAM action.
        The valid values for this object are:
        (1) idle (2) active (3) abort.

        Default value of OAM control is idle.

        This has no correspondence with any internal variable
        and is now deprecated."
DEFVAL { idle }
 ::= { rprOamEntry 11 }

rprOamResponseCount OBJECT-TYPE
    SYNTAX      Unsigned32 (0..65535)
    MAX-ACCESS   read-only
    STATUS       deprecated
    DESCRIPTION
        "Indicates the number of OAM actions responses received.

        This has no correspondence with any internal variable
        and is now deprecated."
 ::= { rprOamEntry 12 }

rprOamAvResponseTime OBJECT-TYPE
    SYNTAX      Unsigned32 (0..65535)
    UNITS       "10 usec"
    MAX-ACCESS   read-only
    STATUS       deprecated

```

DESCRIPTION

"Average response time to receive the OAM reply.

This has no correspondence with any internal variable
and is now deprecated."

::= { rprOamEntry 13 }

rprOamResponseStatus OBJECT-TYPE

SYNTAX INTEGER {
 unknown (1),
 inProcess (2),
 error (3),
 success (4)
 }
MAX-ACCESS read-only
STATUS deprecated

DESCRIPTION

"Status of an OAM action.

The valid values for this object are:

(1) unknown (2) inProcess (3) error (4) success.

The action status is success if at least 90% of the responses
were received till timeout.

This has no correspondence with any internal variable
and is now deprecated."

::= { rprOamEntry 14 }

--
-- RPR changes summary group
--

rprIfChangeSummaryObject OBJECT IDENTIFIER ::= { rprGeneral 5 }

rprIfChangeSummaryNumInterfaces OBJECT-TYPE

SYNTAX Unsigned32
MAX-ACCESS read-only
STATUS current

DESCRIPTION

"Number of RPR interfaces on this device."

::= { rprIfChangeSummaryObject 1 }

rprIfChangeSummaryIfLastChange OBJECT-TYPE

SYNTAX TimeStamp
MAX-ACCESS read-only
STATUS current

DESCRIPTION

"Latest timestamp when any instance of rprIfLastChange changed."

::= { rprIfChangeSummaryObject 2 }

rprIfChangeSummaryIfChanges OBJECT-TYPE

SYNTAX Counter32
MAX-ACCESS read-only
STATUS current

DESCRIPTION

"Number of times any instance of rprIfChanges changed.

The discontinued counter value is indicated

```

        by the ifCounterDiscontinuityTime value."
 ::= { rprIfChangeSummaryObject 3 }

rprIfChangeSummarySpanLastChange OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Latest timestamp when any instance of rprSpanLastChange changed."
 ::= { rprIfChangeSummaryObject 4 }

rprIfChangeSummarySpanChanges OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Number of times any instance of rprSpanChanges changed.

        The discontinued counter value is indicated
        by the ifCounterDiscontinuityTime value."
 ::= { rprIfChangeSummaryObject 5 }

rprIfChangeSummaryFairnessLastChange OBJECT-TYPE
    SYNTAX      TimeStamp
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Latest timestamp when any instance of rprFairnessLastChange changed."
 ::= { rprIfChangeSummaryObject 6 }

rprIfChangeSummaryFairnessChanges OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "Number of times any instance of rprFairnessChanges changed.

        The discontinued counter value is indicated
        by the ifCounterDiscontinuityTime value."
 ::= { rprIfChangeSummaryObject 7 }

-- RPR ring interface current counters table

rprSpanCountersCurrentTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprSpanCountersCurrentEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The RPR MAC Span interface current counters table."
 ::= { rprSpanCounters 1 }

rprSpanCountersCurrentEntry OBJECT-TYPE
    SYNTAX      RprSpanCountersCurrentEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION

```

```
"Frames and octets statistics for the current interval for
the RPR MAC Span interface.

The corresponding instance of rprIfStatsControlTimeElapsed
indicates the number of seconds which have elapsed
so far in the current interval."
INDEX { rprSpanCurrentIfIndex,
          rprSpanCurrentSpan }
::= { rprSpanCountersCurrentTable 1 }

RprSpanCountersCurrentEntry ::= SEQUENCE {
    rprSpanCurrentIfIndex           InterfaceIndex,
    rprSpanCurrentSpan             RprSpan,
    rprSpanCurrentInUcastClassAFrames      HCPerfCurrentCount,
    rprSpanCurrentInUcastClassAOctets     HCPerfCurrentCount,
    rprSpanCurrentInUcastClassBCirFrames   HCPerfCurrentCount,
    rprSpanCurrentInUcastClassBCirOctets   HCPerfCurrentCount,
    rprSpanCurrentInUcastClassBEirFrames   HCPerfCurrentCount,
    rprSpanCurrentInUcastClassBEirOctets   HCPerfCurrentCount,
    rprSpanCurrentInUcastClassCFrames     HCPerfCurrentCount,
    rprSpanCurrentInUcastClassCOctets     HCPerfCurrentCount,
    rprSpanCurrentInMcastClassAFrames     HCPerfCurrentCount,
    rprSpanCurrentInMcastClassAOctets    HCPerfCurrentCount,
    rprSpanCurrentInMcastClassBCirFrames  HCPerfCurrentCount,
    rprSpanCurrentInMcastClassBCirOctets  HCPerfCurrentCount,
    rprSpanCurrentInMcastClassBEirFrames  HCPerfCurrentCount,
    rprSpanCurrentInMcastClassBEirOctets  HCPerfCurrentCount,
    rprSpanCurrentInMcastClassCFrames    HCPerfCurrentCount,
    rprSpanCurrentInMcastClassCOctets    HCPerfCurrentCount,
    rprSpanCurrentOutUcastClassAFrames   HCPerfCurrentCount,
    rprSpanCurrentOutUcastClassAOctets  HCPerfCurrentCount,
    rprSpanCurrentOutUcastClassBCirFrames HCPerfCurrentCount,
    rprSpanCurrentOutUcastClassBCirOctets HCPerfCurrentCount,
    rprSpanCurrentOutUcastClassBEirFrames HCPerfCurrentCount,
    rprSpanCurrentOutUcastClassBEirOctets HCPerfCurrentCount,
    rprSpanCurrentOutUcastClassCFrames  HCPerfCurrentCount,
    rprSpanCurrentOutUcastClassCOctets  HCPerfCurrentCount,
    rprSpanCurrentOutMcastClassAFrames   HCPerfCurrentCount,
    rprSpanCurrentOutMcastClassAOctets  HCPerfCurrentCount,
    rprSpanCurrentOutMcastClassBCirFrames HCPerfCurrentCount,
    rprSpanCurrentOutMcastClassBCirOctets HCPerfCurrentCount,
    rprSpanCurrentOutMcastClassBEirFrames HCPerfCurrentCount,
    rprSpanCurrentOutMcastClassBEirOctets HCPerfCurrentCount,
    rprSpanCurrentOutMcastClassCFrames  HCPerfCurrentCount,
    rprSpanCurrentOutMcastClassCOctets  HCPerfCurrentCount
}

rprSpanCurrentIfIndex OBJECT-TYPE
SYNTAX           InterfaceIndex
MAX-ACCESS       not-accessible
STATUS           current
DESCRIPTION
    "The ifIndex of this RPR interface."
REFERENCE
    "RFC 2863, ifIndex"
::= { rprSpanCountersCurrentEntry 1 }
```

```

rprSpanCurrentSpan OBJECT-TYPE
    SYNTAX      RprSpan
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An indication of the span of the interface for which this
         row contains information."
    ::= { rprSpanCountersCurrentEntry 2 }

rprSpanCurrentInUcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classA unicast frames
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 3 }

rprSpanCurrentInUcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classA unicast octets,
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 4 }

rprSpanCurrentInUcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB CIR unicast frames
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 5 }

rprSpanCurrentInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB CIR unicast octets
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 6 }

rprSpanCurrentInUcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB EIR unicast frames
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 7 }

rprSpanCurrentInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current

```

DESCRIPTION
"The number of received (PHY to MAC) classB EIR unicast octets
in the current interval."
::= { rprSpanCountersCurrentEntry 8 }

rprSpanCurrentInUcastClassCFrames OBJECT-TYPE
SYNTAX HCPfCurrentCount
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) classC unicast frames
in the current interval."
::= { rprSpanCountersCurrentEntry 9 }

rprSpanCurrentInUcastClassCOctets OBJECT-TYPE
SYNTAX HCPfCurrentCount
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) classC unicast octets
in the current interval."
::= { rprSpanCountersCurrentEntry 10 }

rprSpanCurrentInMcastClassAFrames OBJECT-TYPE
SYNTAX HCPfCurrentCount
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) classA multicast and
broadcast frames in the current interval."
::= { rprSpanCountersCurrentEntry 11 }

rprSpanCurrentInMcastClassAOctets OBJECT-TYPE
SYNTAX HCPfCurrentCount
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) classA multicast and
broadcast octets in the current interval."
::= { rprSpanCountersCurrentEntry 12 }

rprSpanCurrentInMcastClassBCirFrames OBJECT-TYPE
SYNTAX HCPfCurrentCount
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) classB CIR multicast and
broadcast frames in the current interval."
::= { rprSpanCountersCurrentEntry 13 }

rprSpanCurrentInMcastClassBCirOctets OBJECT-TYPE
SYNTAX HCPfCurrentCount
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) classB CIR multicast and
broadcast octets in the current interval."
::= { rprSpanCountersCurrentEntry 14 }

```

rprSpanCurrentInMcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB EIR multicast and
         broadcast frames in the current interval."
    ::= { rprSpanCountersCurrentEntry 15 }

rprSpanCurrentInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB EIR multicast and
         broadcast octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 16 }

rprSpanCurrentInMcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classC multicast and
         broadcast frames in the current interval."
    ::= { rprSpanCountersCurrentEntry 17 }

rprSpanCurrentInMcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classC multicast and
         broadcast octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 18 }

rprSpanCurrentOutUcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classA unicast frames
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 19 }

rprSpanCurrentOutUcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classA unicast octets
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 20 }

rprSpanCurrentOutUcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION

```

```
"The number of transmitted (MAC to PHY) classB CIR unicast frames
in the current interval."
 ::= { rprSpanCountersCurrentEntry 21 }

rprSpanCurrentOutUcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of transmitted (MAC to PHY) classB CIR unicast octets
in the current interval."
 ::= { rprSpanCountersCurrentEntry 22 }

rprSpanCurrentOutUcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of transmitted (MAC to PHY) classB EIR unicast frames
in the current interval."
 ::= { rprSpanCountersCurrentEntry 23 }

rprSpanCurrentOutUcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of transmitted (MAC to PHY) classB EIR unicast octets
in the current interval."
 ::= { rprSpanCountersCurrentEntry 24 }

rprSpanCurrentOutUcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of transmitted (MAC to PHY) classC unicast frames
in the current interval."
 ::= { rprSpanCountersCurrentEntry 25 }

rprSpanCurrentOutUcastClassCOctets OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of transmitted (MAC to PHY) classC unicast octets
in the current interval."
 ::= { rprSpanCountersCurrentEntry 26 }

rprSpanCurrentOutMcastClassAFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of transmitted (MAC to PHY) classA multicast and
broadcast frames in the current interval."
 ::= { rprSpanCountersCurrentEntry 27 }

rprSpanCurrentOutMcastClassAOctets OBJECT-TYPE
```

```

SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classA multicast and
    broadcast octets in the current interval."
 ::= { rprSpanCountersCurrentEntry 28 }

rprSpanCurrentOutMcastClassBCirFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classB CIR multicast and
    broadcast frames in the current interval."
 ::= { rprSpanCountersCurrentEntry 29 }

rprSpanCurrentOutMcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classB CIR multicast and
    broadcast octets in the current interval."
 ::= { rprSpanCountersCurrentEntry 30 }

rprSpanCurrentOutMcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classB EIR multicast and
    broadcast frames in the current interval."
 ::= { rprSpanCountersCurrentEntry 31 }

rprSpanCurrentOutMcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classB EIR multicast and
    broadcast octets in the current interval."
 ::= { rprSpanCountersCurrentEntry 32 }

rprSpanCurrentOutMcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classC multicast and
    broadcast frames in the current interval."
 ::= { rprSpanCountersCurrentEntry 33 }

rprSpanCurrentOutMcastClassCOctets OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classC multicast and
    broadcast octets in the current interval."

```

```
        broadcast octets in the current interval."
 ::= { rprSpanCountersCurrentEntry 34 }

--  
-- RPR ring interface interval counters  
--  
  
rprSpanCountersIntervalTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprSpanCountersIntervalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The RPR MAC Span interface interval counters table."
 ::= { rprSpanCounters 2 }

rprSpanCountersIntervalEntry OBJECT-TYPE
    SYNTAX      RprSpanCountersIntervalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Frames and octets statistics collected for a particular
         interval for the RPR MAC Span interface of a
         particular span of a particular RPR interface.
         The corresponding instance of rprIfValidIntervals
         indicates the number of intervals for which the set of
         statistics is available."
    INDEX { rprSpanIntervalIfIndex,
            rprSpanIntervalSpan,
            rprSpanIntervalNumber }
 ::= { rprSpanCountersIntervalTable 1 }

RprSpanCountersIntervalEntry ::= SEQUENCE {
    rprSpanIntervalIfIndex           InterfaceIndex,
    rprSpanIntervalSpan             RprSpan,
    rprSpanIntervalNumber           Unsigned32,
    rprSpanIntervalValidData        TruthValue,
    rprSpanIntervalTimeElapsed      Unsigned32,
    rprSpanIntervalStartTime        DateAndTime,
    rprSpanIntervalInUcastClassAFrames HCPerfIntervalCount,
    rprSpanIntervalInUcastClassAOctets HCPerfIntervalCount,
    rprSpanIntervalInUcastClassBCirFrames HCPerfIntervalCount,
    rprSpanIntervalInUcastClassBCirOctets HCPerfIntervalCount,
    rprSpanIntervalInUcastClassBEirFrames HCPerfIntervalCount,
    rprSpanIntervalInUcastClassBEirOctets HCPerfIntervalCount,
    rprSpanIntervalInUcastClassCFrames HCPerfIntervalCount,
    rprSpanIntervalInUcastClassCOctets HCPerfIntervalCount,
    rprSpanIntervalInMcastClassAFrames HCPerfIntervalCount,
    rprSpanIntervalInMcastClassAOctets HCPerfIntervalCount,
    rprSpanIntervalInMcastClassBCirFrames HCPerfIntervalCount,
    rprSpanIntervalInMcastClassBCirOctets HCPerfIntervalCount,
    rprSpanIntervalInMcastClassBEirFrames HCPerfIntervalCount,
    rprSpanIntervalInMcastClassBEirOctets HCPerfIntervalCount,
    rprSpanIntervalInMcastClassCFrames HCPerfIntervalCount,
    rprSpanIntervalInMcastClassCOctets HCPerfIntervalCount,
    rprSpanIntervalOutUcastClassAFrames HCPerfIntervalCount,
```

```

rprSpanIntervalOutUcastClassAOctets      HCPerfIntervalCount,
rprSpanIntervalOutUcastClassBCirFrames   HCPerfIntervalCount,
rprSpanIntervalOutUcastClassBCirOctets   HCPerfIntervalCount,
rprSpanIntervalOutUcastClassBEirFrames   HCPerfIntervalCount,
rprSpanIntervalOutUcastClassBEirOctets   HCPerfIntervalCount,
rprSpanIntervalOutUcastClassCFrames     HCPerfIntervalCount,
rprSpanIntervalOutUcastClassCOctets     HCPerfIntervalCount,
rprSpanIntervalOutMcastClassAFrames    HCPerfIntervalCount,
rprSpanIntervalOutMcastClassAOctets   HCPerfIntervalCount,
rprSpanIntervalOutMcastClassBCirFrames HCPerfIntervalCount,
rprSpanIntervalOutMcastClassBCirOctets HCPerfIntervalCount,
rprSpanIntervalOutMcastClassBEirFrames HCPerfIntervalCount,
rprSpanIntervalOutMcastClassBEirOctets HCPerfIntervalCount,
rprSpanIntervalOutMcastClassCFrames   HCPerfIntervalCount,
rprSpanIntervalOutMcastClassCOctets   HCPerfIntervalCount
}

rprSpanIntervalIfIndex OBJECT-TYPE
  SYNTAX      InterfaceIndex
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION
    "The ifIndex of this RPR interface."
  ::= { rprSpanCountersIntervalEntry 1 }

rprSpanIntervalSpan OBJECT-TYPE
  SYNTAX      RprSpan
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION
    "An indication of the span of the interface for which this
     row contains information."
  ::= { rprSpanCountersIntervalEntry 2 }

rprSpanIntervalNumber OBJECT-TYPE
  SYNTAX      Unsigned32 (1..96)
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION
    "A number between 1 and 96, which identifies the intervals
     for which the set of statistics is available. The interval
     identified by 1 is the most recently completed 15 minute
     interval, and interval identified by N is the interval
     immediately preceding the one identified by N-1."
  ::= { rprSpanCountersIntervalEntry 3 }

rprSpanIntervalValidData OBJECT-TYPE
  SYNTAX      TruthValue
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "This variable indicates if the data for this interval
     is valid.
     It will be valid if it contains data for 900 seconds
     plus or minus 10 seconds."
  ::= { rprSpanCountersIntervalEntry 4 }

rprSpanIntervalTimeElapsed OBJECT-TYPE

```

```
SYNTAX      Unsigned32 (0..910)
UNITS      "Seconds"
MAX-ACCESS  read-only
STATUS     current
DESCRIPTION
  "The duration of a particular interval in seconds.
  If, for some reason, such as an adjustment in the system's
  time-of-day clock, the current interval exceeds the maximum
  value, the agent will return the maximum value."
 ::= { rprSpanCountersIntervalEntry 5 }

rprSpanIntervalStartTime OBJECT-TYPE
SYNTAX      DateAndTime
MAX-ACCESS  read-only
STATUS     current
DESCRIPTION
  "Indicates the wall clock time that this interval started."
 ::= { rprSpanCountersIntervalEntry 6 }

rprSpanIntervalInUcastClassAFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS     current
DESCRIPTION
  "The number of received (PHY to MAC) classA unicast frames
  in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 7 }

rprSpanIntervalInUcastClassAOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS     current
DESCRIPTION
  "The number of received (PHY to MAC) classA unicast octets
  in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 8 }

rprSpanIntervalInUcastClassBCirFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS     current
DESCRIPTION
  "The number of received (PHY to MAC) classB CIR unicast frames
  in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 9 }

rprSpanIntervalInUcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS     current
DESCRIPTION
  "The number of received (PHY to MAC) classB CIR unicast octets
  in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 10 }

rprSpanIntervalInUcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS     current
```

```

DESCRIPTION
    "The number of received (PHY to MAC) classB EIR unicast
    frames in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 11 }

rprSpanIntervalInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
DESCRIPTION
    "The number of received (PHY to MAC) classB EIR unicast
    octets in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 12 }

rprSpanIntervalInUcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
DESCRIPTION
    "The number of received (PHY to MAC) classC unicast
    frames in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 13 }

rprSpanIntervalInUcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
DESCRIPTION
    "The number of received (PHY to MAC) classC unicast
    octets in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 14 }

rprSpanIntervalInMcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
DESCRIPTION
    "The number of received (PHY to MAC) classA multicast and
    broadcast frames in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 15 }

rprSpanIntervalInMcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
DESCRIPTION
    "The number of received (PHY to MAC) classA multicast and
    broadcast octets in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 16 }

rprSpanIntervalInMcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
DESCRIPTION
    "The number of received (PHY to MAC) classB CIR multicast and
    broadcast frames in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 17 }

```

```
rprSpanIntervalInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB CIR multicast and
         broadcast octets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 18 }

rprSpanIntervalInMcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB EIR multicast and
         broadcast frames in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 19 }

rprSpanIntervalInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB EIR multicast and
         broadcast octets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 20 }

rprSpanIntervalInMcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classC multicast and
         broadcast frames in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 21 }

rprSpanIntervalInMcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classC multicast and
         broadcast octets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 22 }

rprSpanIntervalOutUcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classA unicast frames
         in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 23 }

rprSpanIntervalOutUcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
```

```

        "The number of transmitted (MAC to PHY) classA unicast octets
        in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 24 }

rprSpanIntervalOutUcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classB CIR unicast frames,
        in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 25 }

rprSpanIntervalOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classB CIR unicast octets
        in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 26 }

rprSpanIntervalOutUcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classB EIR unicast frames
        in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 27 }

rprSpanIntervalOutUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classB EIR unicast octets
        in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 28 }

rprSpanIntervalOutUcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classC unicast frames
        in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 29 }

rprSpanIntervalOutUcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classC unicast octets
        in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 30 }

rprSpanIntervalOutMcastClassAFrames OBJECT-TYPE

```

```
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classA multicast and
     broadcast frames in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 31 }

rprSpanIntervalOutMcastClassAOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classA multicast and
     broadcast octets in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 32 }

rprSpanIntervalOutMcastClassBCirFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classB CIR multicast and
     broadcast frames in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 33 }

rprSpanIntervalOutMcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classB CIR multicast and
     broadcast octets in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 34 }

rprSpanIntervalOutMcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classB EIR multicast and
     broadcast frames in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 35 }

rprSpanIntervalOutMcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classB EIR multicast and
     broadcast octets in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 36 }

rprSpanIntervalOutMcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of transmitted (MAC to PHY) classC multicast and
```

```

        broadcast frames in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 37 }

rprSpanIntervalOutMcastClassCOctets OBJECT-TYPE
    SYNTAX          HCPerfIntervalCount
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classC multicast and
        broadcast octets in a particular interval in the past 24 hours."
 ::= { rprSpanCountersIntervalEntry 38 }

-- 
-- RPR ring interface day (24 hour summaries) counters
--

rprSpanCountersDayTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF RprSpanCountersDayEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "The RPR Mac Span Day Table contains the cumulative sum
        of the various statistics for the 24 hour period
        preceding the current interval."
 ::= { rprSpanCounters 3 }

rprSpanCountersDayEntry OBJECT-TYPE
    SYNTAX          RprSpanCountersDayEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "An entry in the RPR Span Day table."
 INDEX { rprSpanDayIfIndex,
          rprSpanDaySpan }
 ::= { rprSpanCountersDayTable 1 }

RprSpanCountersDayEntry ::= SEQUENCE {
    rprSpanDayIfIndex                  InterfaceIndex,
    rprSpanDaySpan                    RprSpan,
    rprSpanDayInUcastClassAFrames    HCPerfTotalCount,
    rprSpanDayInUcastClassAOctets    HCPerfTotalCount,
    rprSpanDayInUcastClassBCirFrames HCPerfTotalCount,
    rprSpanDayInUcastClassBCirOctets HCPerfTotalCount,
    rprSpanDayInUcastClassBEirFrames HCPerfTotalCount,
    rprSpanDayInUcastClassBEirOctets HCPerfTotalCount,
    rprSpanDayInUcastClassCFFrames   HCPerfTotalCount,
    rprSpanDayInUcastClassCOctets    HCPerfTotalCount,
    rprSpanDayInMcastClassAFrames    HCPerfTotalCount,
    rprSpanDayInMcastClassAOctets    HCPerfTotalCount,
    rprSpanDayInMcastClassBCirFrames HCPerfTotalCount,
    rprSpanDayInMcastClassBCirOctets HCPerfTotalCount,
    rprSpanDayInMcastClassBEirFrames HCPerfTotalCount,
    rprSpanDayInMcastClassBEirOctets HCPerfTotalCount,
    rprSpanDayInMcastClassCFFrames   HCPerfTotalCount,
    rprSpanDayInMcastClassCOctets    HCPerfTotalCount,
    rprSpanDayOutUcastClassAFrames   HCPerfTotalCount,
}

```

```
rprSpanDayOutUcastClassAOctets    HCPerfTotalCount,
rprSpanDayOutUcastClassBCirFrames HCPerfTotalCount,
rprSpanDayOutUcastClassBCirOctets HCPerfTotalCount,
rprSpanDayOutUcastClassBEirFrames HCPerfTotalCount,
rprSpanDayOutUcastClassBEirOctets HCPerfTotalCount,
rprSpanDayOutUcastClassCFrames   HCPerfTotalCount,
rprSpanDayOutUcastClassCOctets   HCPerfTotalCount,

rprSpanDayOutMcastClassAFrames   HCPerfTotalCount,
rprSpanDayOutMcastClassAOctets  HCPerfTotalCount,
rprSpanDayOutMcastClassBCirFrames HCPerfTotalCount,
rprSpanDayOutMcastClassBCirOctets HCPerfTotalCount,
rprSpanDayOutMcastClassBEirFrames HCPerfTotalCount,
rprSpanDayOutMcastClassBEirOctets HCPerfTotalCount,
rprSpanDayOutMcastClassCFrames  HCPerfTotalCount,
rprSpanDayOutMcastClassCOctets  HCPerfTotalCount
}

rprSpanDayIfIndex OBJECT-TYPE
  SYNTAX      InterfaceIndex
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION
    "The ifIndex of this RPR interface."
  ::= { rprSpanCountersDayEntry 1 }

rprSpanDaySpan OBJECT-TYPE
  SYNTAX      RprSpan
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION
    "An indication of the span of the interface for which this
     row contains information."
  ::= { rprSpanCountersDayEntry 2 }

rprSpanDayInUcastClassAFrames OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of received (PHY to MAC) classA unicast frames."
  ::= { rprSpanCountersDayEntry 3 }

rprSpanDayInUcastClassAOctets OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of received (PHY to MAC) classA unicast octets."
  ::= { rprSpanCountersDayEntry 4 }

rprSpanDayInUcastClassBCirFrames OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of received (PHY to MAC) classB CIR unicast frames."
  ::= { rprSpanCountersDayEntry 5 }
```

```

rprSpanDayInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB CIR unicast octets."
    ::= { rprSpanCountersDayEntry 6 }

rprSpanDayInUcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB EIR unicast frames."
    ::= { rprSpanCountersDayEntry 7 }

rprSpanDayInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB EIR unicast octets."
    ::= { rprSpanCountersDayEntry 8 }

rprSpanDayInUcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classC unicast frames."
    ::= { rprSpanCountersDayEntry 9 }

rprSpanDayInUcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classC unicast octets."
    ::= { rprSpanCountersDayEntry 10 }

rprSpanDayInMcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classA multicast
         and broadcast frames."
    ::= { rprSpanCountersDayEntry 11 }

rprSpanDayInMcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classA multicast
         and broadcast octets."
    ::= { rprSpanCountersDayEntry 12 }

rprSpanDayInMcastClassBCirFrames OBJECT-TYPE

```

```
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classB CIR multicast
  and broadcast frames."
 ::= { rprSpanCountersDayEntry 13 }

rprSpanDayInMcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classB CIR multicast
  and broadcast octets."
 ::= { rprSpanCountersDayEntry 14 }

rprSpanDayInMcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classB EIR multicast
  and broadcast frames."
 ::= { rprSpanCountersDayEntry 15 }

rprSpanDayInMcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classB EIR multicast
  and broadcast octets."
 ::= { rprSpanCountersDayEntry 16 }

rprSpanDayInMcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classC multicast
  and broadcast frames."
 ::= { rprSpanCountersDayEntry 17 }

rprSpanDayInMcastClassCOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classC multicast
  and broadcast octets."
 ::= { rprSpanCountersDayEntry 18 }

rprSpanDayOutUcastClassAFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of transmitted (MAC to PHY) classA unicast frames."
```

```

 ::= { rprSpanCountersDayEntry 19 }

rprSpanDayOutUcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classA unicast octets."
 ::= { rprSpanCountersDayEntry 20 }

rprSpanDayOutUcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classB CIR unicast frames."
 ::= { rprSpanCountersDayEntry 21 }

rprSpanDayOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classB CIR unicast octets."
 ::= { rprSpanCountersDayEntry 22 }

rprSpanDayOutUcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classB EIR unicast frames."
 ::= { rprSpanCountersDayEntry 23 }

rprSpanDayOutUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classB EIR unicast octets."
 ::= { rprSpanCountersDayEntry 24 }

rprSpanDayOutUcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classC unicast frames."
 ::= { rprSpanCountersDayEntry 25 }

rprSpanDayOutUcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classC unicast octets."
 ::= { rprSpanCountersDayEntry 26 }

rprSpanDayOutMcastClassAFrames OBJECT-TYPE

```

```
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of transmitted (MAC to PHY) classA multicast and broadcast
  frames."
 ::= { rprSpanCountersDayEntry 27 }
```

```
rprSpanDayOutMcastClassAOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of transmitted (MAC to PHY) classA multicast and broadcast
  octets."
 ::= { rprSpanCountersDayEntry 28 }
```

```
rprSpanDayOutMcastClassBCirFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of transmitted (MAC to PHY) classB CIR
  multicast and broadcast frames."
 ::= { rprSpanCountersDayEntry 29 }
```

```
rprSpanDayOutMcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of transmitted (MAC to PHY) classB CIR
  multicast and broadcast octets."
 ::= { rprSpanCountersDayEntry 30 }
```

```
rprSpanDayOutMcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of transmitted (MAC to PHY) classB EIR
  multicast and broadcast frames."
 ::= { rprSpanCountersDayEntry 31 }
```

```
rprSpanDayOutMcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of transmitted (MAC to PHY) classB EIR
  multicast and broadcast octets."
 ::= { rprSpanCountersDayEntry 32 }
```

```
rprSpanDayOutMcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of transmitted (MAC to PHY) classC multicast and broadcast
```

```

frames."
 ::= { rprSpanCountersDayEntry 33 }

rprSpanDayOutMcastClassCOctets OBJECT-TYPE
    SYNTAX          HCPerfTotalCount
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classC multicast and broadcast
         octets."
 ::= { rprSpanCountersDayEntry 34 }

--
-- RPR ring interface continuously running counters
--

rprSpanCountersStatsTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF RprSpanCountersStatsEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "The RPR Mac Span interface total counters table.

        The DiscontinuityTime for this table is indicated by
        ifCounterDiscontinuityTime defined in ifXTable."
 ::= { rprSpanCounters 4 }

rprSpanCountersStatsEntry OBJECT-TYPE
    SYNTAX          RprSpanCountersStatsEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "An entry in the span stats table."
    INDEX { rprSpanStatsIfIndex,
            rprSpanStatsSpan }
 ::= { rprSpanCountersStatsTable 1 }

RprSpanCountersStatsEntry ::= SEQUENCE {
    rprSpanStatsIfIndex           InterfaceIndex,
    rprSpanStatsSpan             RprSpan,
    rprSpanStatsInUcastClassAFrames Counter64,
    rprSpanStatsInUcastClassAOctets Counter64,
    rprSpanStatsInUcastClassBCirFrames Counter64,
    rprSpanStatsInUcastClassBCirOctets Counter64,
    rprSpanStatsInUcastClassBEirFrames Counter64,
    rprSpanStatsInUcastClassBEirOctets Counter64,
    rprSpanStatsInUcastClassCFrames Counter64,
    rprSpanStatsInUcastClassCOctets Counter64,
    rprSpanStatsInMcastClassAFrames Counter64,
    rprSpanStatsInMcastClassAOctets Counter64,
    rprSpanStatsInMcastClassBCirFrames Counter64,
    rprSpanStatsInMcastClassBCirOctets Counter64,
    rprSpanStatsInMcastClassBEirFrames Counter64,
    rprSpanStatsInMcastClassBEirOctets Counter64,
    rprSpanStatsInMcastClassCFrames Counter64,
    rprSpanStatsInMcastClassCOctets Counter64,
}

```

```

rprSpanStatsInCtrlFrames          Counter64,
rprSpanStatsInOamEchoFrames      Counter64,
rprSpanStatsInOamFlushFrames     Counter64,
rprSpanStatsInOamOrgFrames       Counter64,
rprSpanStatsInTopoAtdFrames      Counter64,
rprSpanStatsInTopoChkSumFrames   Counter64,
rprSpanStatsInTopoTpFrames       Counter64,

rprSpanStatsOutUcastClassAFrames Counter64,
rprSpanStatsOutUcastClassAOctets Counter64,
rprSpanStatsOutUcastClassBCirFrames Counter64,
rprSpanStatsOutUcastClassBCirOctets Counter64,
rprSpanStatsOutUcastClassBEirFrames Counter64,
rprSpanStatsOutUcastClassBEirOctets Counter64,
rprSpanStatsOutUcastClassCFFrames Counter64,
rprSpanStatsOutUcastClassCOctets Counter64,

rprSpanStatsOutMcastClassAFrames Counter64,
rprSpanStatsOutMcastClassAOctets Counter64,
rprSpanStatsOutMcastClassBCirFrames Counter64,
rprSpanStatsOutMcastClassBCirOctets Counter64,
rprSpanStatsOutMcastClassBEirFrames Counter64,
rprSpanStatsOutMcastClassBEirOctets Counter64,
rprSpanStatsOutMcastClassCFFrames Counter64,
rprSpanStatsOutMcastClassCOctets Counter64,

rprSpanStatsOutCtrlFrames        Counter64,
rprSpanStatsOutOamEchoFrames    Counter64,
rprSpanStatsOutOamFlushFrames   Counter64,
rprSpanStatsOutOamOrgFrames     Counter64,
rprSpanStatsOutTopoAtdFrames   Counter64,
rprSpanStatsOutTopoChkSumFrames Counter64,
rprSpanStatsOutTopoTpFrames     Counter64,

rprSpanStatsInOamSasNotifyFrames Counter64,
rprSpanStatsOutOamSasNotifyFrames Counter64,

rprSpanStatsInOamPircStatusFrames Counter64,
rprSpanStatsOutOamPircStatusFrames Counter64,

rprSpanStatsInOamEchoReqFrames  Counter64,
rprSpanStatsInOamEchoRspFrames  Counter64,
rprSpanStatsOutOamEchoReqFrames Counter64,
rprSpanStatsOutOamEchoRspFrames Counter64
}

rprSpanStatsIfIndex OBJECT-TYPE
  SYNTAX      InterfaceIndex
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION
    "The ifIndex of this RPR interface."
  ::= { rprSpanCountersStatsEntry 1 }

rprSpanStatsSpan OBJECT-TYPE
  SYNTAX      RprSpan
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION

```

"An indication of the span of the interface for which this row contains information."

REFERENCE

"IEEE 802.17 Subclause 7.2.2, myRi"
 $\text{ ::= } \{ \text{rprSpanCountersStatsEntry} \ 2 \}$

rprSpanStatsInUcastClassAFrames OBJECT-TYPE

SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current

DESCRIPTION

"The number of received (PHY to MAC) classA unicast frames."

REFERENCE

"IEEE 802.17 Subclause 7.2.5, rxUcastClassAFrames"
 $\text{ ::= } \{ \text{rprSpanCountersStatsEntry} \ 3 \}$

rprSpanStatsInUcastClassAOctets OBJECT-TYPE

SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current

DESCRIPTION

"The number of received (PHY to MAC) classA unicast octets."

REFERENCE

"IEEE 802.17 Subclause 7.2.5, rxUcastClassABytes"
 $\text{ ::= } \{ \text{rprSpanCountersStatsEntry} \ 4 \}$

rprSpanStatsInUcastClassBCirFrames OBJECT-TYPE

SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current

DESCRIPTION

"The number of received (PHY to MAC) classB CIR unicast frames."

REFERENCE

"IEEE 802.17 Subclause 7.2.5, rxUcastClassBCirFrames"
 $\text{ ::= } \{ \text{rprSpanCountersStatsEntry} \ 5 \}$

rprSpanStatsInUcastClassBCirOctets OBJECT-TYPE

SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current

DESCRIPTION

"The number of received (PHY to MAC) classB CIR unicast octets."

REFERENCE

"IEEE 802.17 Subclause 7.2.5, rxUcastClassBCirBytes"
 $\text{ ::= } \{ \text{rprSpanCountersStatsEntry} \ 6 \}$

rprSpanStatsInUcastClassBEirFrames OBJECT-TYPE

SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current

DESCRIPTION

"The number of received (PHY to MAC) classB EIR unicast frames."

REFERENCE

"IEEE 802.17 Subclause 7.2.5, rxUcastClassBEirFrames"
 $\text{ ::= } \{ \text{rprSpanCountersStatsEntry} \ 7 \}$

rprSpanStatsInUcastClassBEirOctets OBJECT-TYPE

SYNTAX Counter64
 MAX-ACCESS read-only

```
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classB EIR unicast octets."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, rxUcastClassBEirBytes"
 ::= { rprSpanCountersStatsEntry 8 }

rprSpanStatsInUcastClassCFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classC unicast frames."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, rxUcastClassCFrames"
 ::= { rprSpanCountersStatsEntry 9 }

rprSpanStatsInUcastClassCOctets OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classC unicast octets."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, rxUcastClassCBytes"
 ::= { rprSpanCountersStatsEntry 10 }

rprSpanStatsInMcastClassAFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classA multicast and
  broadcast frames."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, rxMcastClassAFrames"
 ::= { rprSpanCountersStatsEntry 11 }

rprSpanStatsInMcastClassAOctets OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classA multicast and
  broadcast octets."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, rxMcastClassABytes"
 ::= { rprSpanCountersStatsEntry 12 }

rprSpanStatsInMcastClassBCirFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The number of received (PHY to MAC) classB CIR multicast
  and broadcast frames."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, rxMcastClassBCirFrames"
 ::= { rprSpanCountersStatsEntry 13 }
```

```

rprSpanStatsInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB CIR multicast
         and broadcast octets."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, rxMcastClassBCirBytes"
    ::= { rprSpanCountersStatsEntry 14 }

rprSpanStatsInMcastClassBEirFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB EIR multicast
         and broadcast frames."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, rxMcastClassBEirFrames"
    ::= { rprSpanCountersStatsEntry 15 }

rprSpanStatsInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classB EIR multicast
         and broadcast octets."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, rxMcastClassBEirBytes"
    ::= { rprSpanCountersStatsEntry 16 }

rprSpanStatsInMcastClassCFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classC multicast and
         broadcast frames."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, rxMcastClassCFrames"
    ::= { rprSpanCountersStatsEntry 17 }

rprSpanStatsInMcastClassCOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) classC multicast and
         broadcast octets."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, rxMcastClassCBytes"
    ::= { rprSpanCountersStatsEntry 18 }

rprSpanStatsInCtrlFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only

```

STATUS current
DESCRIPTION
"The number of received (PHY to MAC) control frames processed by this MAC.
This does not include control frames in transit, i.e., a multicast control frame received from a ringlet will be counted as In but not Out.
This does not include Fairness or idle frames."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, toCtrlFrames"
::= { rprSpanCountersStatsEntry 19 }

rprSpanStatsInOamEchoFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) OAM echo frames processed by this MAC.

See also rprSpanStatsInOamEchoReqFrames and rprSpanStatsInOamEchoRspFrames."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, toCtrlOamEchoReqFrames and toCtrlOamEchoRspFrames"
::= { rprSpanCountersStatsEntry 20 }

rprSpanStatsInOamFlushFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) OAM flush frames processed by this MAC."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, toCtrlOamFlushFrames"
::= { rprSpanCountersStatsEntry 21 }

rprSpanStatsInOamOrgFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) OAM Org frames processed by this MAC."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, toCtrlOamOrgFrames"
::= { rprSpanCountersStatsEntry 22 }

rprSpanStatsInTopoAtdFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of received (PHY to MAC) Topology ATD frames processed by this MAC."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, toCtrlTopoATDFrames"
::= { rprSpanCountersStatsEntry 23 }

```

rprSpanStatsInTopoChkSumFrames OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of received (PHY to MAC) topology
     checksum frames processed by this MAC."
  REFERENCE
    "IEEE 802.17 Subclause 7.2.5, toCtrlTopoSumFrames"
    ::= { rprSpanCountersStatsEntry 24 }

rprSpanStatsInTopoTpFrames OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of received (PHY to MAC) topology TP
     frames processed by this MAC."
  REFERENCE
    "IEEE 802.17 Subclause 7.2.5, toCtrlTopoTPFrames"
    ::= { rprSpanCountersStatsEntry 25 }

rprSpanStatsOutUcastClassAFrames OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of transmitted (MAC to PHY) classA unicast frames."
  REFERENCE
    "IEEE 802.17 Subclause 7.2.5, txUcastClassAFrames"
    ::= { rprSpanCountersStatsEntry 26 }

rprSpanStatsOutUcastClassAOctets OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of transmitted (MAC to PHY) classA unicast octets."
  REFERENCE
    "IEEE 802.17 Subclause 7.2.5, txUcastClassABytes"
    ::= { rprSpanCountersStatsEntry 27 }

rprSpanStatsOutUcastClassBCirFrames OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of transmitted (MAC to PHY) classB CIR unicast frames."
  REFERENCE
    "IEEE 802.17 Subclause 7.2.5, txUcastClassBCirFrames"
    ::= { rprSpanCountersStatsEntry 28 }

rprSpanStatsOutUcastClassBCirOctets OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of transmitted (MAC to PHY) classB CIR unicast octets."

```

REFERENCE
"IEEE 802.17 Subclause 7.2.5, txUcastClassBCirBytes"
::= { rprSpanCountersStatsEntry 29 }

rprSpanStatsOutUcastClassBEirFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of transmitted (MAC to PHY) classB EIR unicast frames."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, txUcastClassBEirFrames"
::= { rprSpanCountersStatsEntry 30 }

rprSpanStatsOutUcastClassBEirOctets OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of transmitted (MAC to PHY) classB EIR unicast octets."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, txUcastClassBEirBytes"
::= { rprSpanCountersStatsEntry 31 }

rprSpanStatsOutUcastClassCFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of transmitted (MAC to PHY) classC unicast frames."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, txUcastClassCFrames"
::= { rprSpanCountersStatsEntry 32 }

rprSpanStatsOutUcastClassCOctets OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of transmitted (MAC to PHY) classC unicast octets."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, txUcastClassCBytes"
::= { rprSpanCountersStatsEntry 33 }

rprSpanStatsOutMcastClassAFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of transmitted (MAC to PHY) classA multicast and broadcast frames."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, txMcastClassAFrames"
::= { rprSpanCountersStatsEntry 34 }

rprSpanStatsOutMcastClassAOctets OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current

```

DESCRIPTION
  "The number of transmitted (MAC to PHY) classA multicast and
  broadcast octets."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, txMcastClassABytes"
  ::= { rprSpanCountersStatsEntry 35 }

rprSpanStatsOutMcastClassBCirFrames OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of transmitted (MAC to PHY) classB CIR
     multicast and broadcast frames."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, txMcastClassBCirFrames"
  ::= { rprSpanCountersStatsEntry 36 }

rprSpanStatsOutMcastClassBCirOctets OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of transmitted (MAC to PHY) classB CIR
     multicast and broadcast octets."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, txMcastClassBCirBytes"
  ::= { rprSpanCountersStatsEntry 37 }

rprSpanStatsOutMcastClassBEirFrames OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of transmitted (MAC to PHY) classB EIR
     multicast and broadcast frames."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, txMcastClassBEirFrames"
  ::= { rprSpanCountersStatsEntry 38 }

rprSpanStatsOutMcastClassBEirOctets OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of transmitted (MAC to PHY) classB EIR
     multicast and broadcast octets."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, txMcastClassBEirBytes"
  ::= { rprSpanCountersStatsEntry 39 }

rprSpanStatsOutMcastClassCFrames OBJECT-TYPE
  SYNTAX      Counter64
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of transmitted (MAC to PHY) classC multicast and
     broadcast frames."
REFERENCE

```

```
"IEEE 802.17 Subclause 7.2.5, txMcastClassCFrames"
 ::= { rprSpanCountersStatsEntry 40 }

rprSpanStatsOutMcastClassCOctets OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) classC multicast and
         broadcast octets."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, txMcastClassCBytes"
        ::= { rprSpanCountersStatsEntry 41 }

rprSpanStatsOutCtrlFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) control frames
         generated by this MAC.
         This does not include control frames in transit,
         i.e., a multicast control frame received from a ringlet
         will be counted as In but not Out.
         This does not include Fairness or idle frames."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, fromCtrlFrames"
        ::= { rprSpanCountersStatsEntry 42 }

rprSpanStatsOutOamEchoFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) OAM echo
         frames generated by this MAC.

         See also rprSpanStatsOutOamEchoReqFrames and
         rprSpanStatsOutOamEchoRspFrames."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5,
         fromCtrlOamEchoReqFrames and fromCtrlOamEchoRspFrames"
        ::= { rprSpanCountersStatsEntry 43 }

rprSpanStatsOutOamFlushFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) OAM flush
         frames generated by this MAC."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, fromCtrlOamFlushFrames"
        ::= { rprSpanCountersStatsEntry 44 }

rprSpanStatsOutOamOrgFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
```

DESCRIPTION
 "The number of transmitted (MAC to PHY) OAM Org frames generated by this MAC."

REFERENCE
 "IEEE 802.17 Subclause 7.2.5, fromCtrlOamOrgFrames"
 ::= { rprSpanCountersStatsEntry 45 }

rprSpanStatsOutTopoAtdFrames OBJECT-TYPE
 SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of transmitted (MAC to PHY) topology ATD frames generated by this MAC."
 REFERENCE
 "IEEE 802.17 Subclause 7.2.5, fromCtrlTopoATDFrames"
 ::= { rprSpanCountersStatsEntry 46 }

rprSpanStatsOutTopoChkSumFrames OBJECT-TYPE
 SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of transmitted (MAC to PHY) topology checksum frames generated by this MAC."
 REFERENCE
 "IEEE 802.17 Subclause 7.2.5, fromCtrlTopoSumFrames"
 ::= { rprSpanCountersStatsEntry 47 }

rprSpanStatsOutTopoTpFrames OBJECT-TYPE
 SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of transmitted (MAC to PHY) topology TP frames generated by this MAC."
 REFERENCE
 "IEEE 802.17 Subclause 7.2.5, fromCtrlTopoTPFrames"
 ::= { rprSpanCountersStatsEntry 48 }

rprSpanStatsInOamSasNotifyFrames OBJECT-TYPE
 SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of received (PHY to MAC) OAM SAS Notify frames generated by this MAC."
 REFERENCE
 "IEEE 802.17 Subclause 7.2.5, toCtrlOamSasNotifyFrames"
 ::= { rprSpanCountersStatsEntry 49 }

rprSpanStatsOutOamSasNotifyFrames OBJECT-TYPE
 SYNTAX Counter64
 MAX-ACCESS read-only
 STATUS current
 DESCRIPTION
 "The number of transmitted (MAC to PHY) OAM SAS Notify frames generated by this MAC."
 REFERENCE

```
"IEEE 802.17 Subclause 7.2.5, fromCtrlOamSasNotifyFrames"
 ::= { rprSpanCountersStatsEntry 50 }

rprSpanStatsInOamPircStatusFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of received (PHY to MAC) OAM PIRC status
         frames generated by this MAC."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, toCtrlOamPircStatusFrames"
    ::= { rprSpanCountersStatsEntry 51 }

rprSpanStatsOutOamPircStatusFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of transmitted (MAC to PHY) OAM PIRC status
         frames generated by this MAC."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, fromCtrlOamPircStatusFrames"
    ::= { rprSpanCountersStatsEntry 52 }

rprSpanStatsInOamEchoReqFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of received (PHY to MAC) OAM echo
         request frames processed by this MAC."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, fromCtrlOamEchoReqFrames"
    ::= { rprSpanCountersStatsEntry 53 }

rprSpanStatsInOamEchoRspFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of received (PHY to MAC) OAM echo
         response frames processed by this MAC."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, fromCtrlOamEchoRspFrames"
    ::= { rprSpanCountersStatsEntry 54 }

rprSpanStatsOutOamEchoReqFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of sent (MAC to PHY) OAM echo
         request frames processed by this MAC."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, toCtrlOamEchoReqFrames"
    ::= { rprSpanCountersStatsEntry 55 }

rprSpanStatsOutOamEchoRspFrames OBJECT-TYPE
```

```

SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of sent (MAC to PHY) OAM echo
     response frames processed by this MAC."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, toCtrlOamEchoRspFrames"
::= { rprSpanCountersStatsEntry 56 }

-- 
-- RPR Client interface current counters table
-- 

rprClientCountersCurrentTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprClientCountersCurrentEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The local station traffic current counters table."
    ::= { rprClientCounters 1 }

rprClientCountersCurrentEntry OBJECT-TYPE
    SYNTAX      RprClientCountersCurrentEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Frames and octets statistics for the current interval for
         the local station traffic of a particular RPR client interface.
         The corresponding instance of rprIfTimeElapsed indicates
         the number of seconds which have elapsed so far in the
         current interval."
    INDEX { rprClientCurrentIfIndex }
    ::= { rprClientCountersCurrentTable 1 }

RprClientCountersCurrentEntry ::= SEQUENCE {
    rprClientCurrentIfIndex           InterfaceIndex,
    rprClientCurrentInUcastClassAFrames   HCPerfCurrentCount,
    rprClientCurrentInUcastClassAOctets   HCPerfCurrentCount,
    rprClientCurrentInUcastClassBCirFrames HCPerfCurrentCount,
    rprClientCurrentInUcastClassBCirOctets HCPerfCurrentCount,
    rprClientCurrentInUcastClassBEirFrames HCPerfCurrentCount,
    rprClientCurrentInUcastClassBEirOctets HCPerfCurrentCount,
    rprClientCurrentInUcastClassCFrames   HCPerfCurrentCount,
    rprClientCurrentInUcastClassCOctets   HCPerfCurrentCount,
    rprClientCurrentInMcastClassAFrames   HCPerfCurrentCount,
    rprClientCurrentInMcastClassAOctets   HCPerfCurrentCount,
    rprClientCurrentInMcastClassBCirFrames HCPerfCurrentCount,
    rprClientCurrentInMcastClassBCirOctets HCPerfCurrentCount,
    rprClientCurrentInMcastClassBEirFrames HCPerfCurrentCount,
    rprClientCurrentInMcastClassBEirOctets HCPerfCurrentCount,
    rprClientCurrentInMcastClassCFrames   HCPerfCurrentCount,
    rprClientCurrentInMcastClassCOctets   HCPerfCurrentCount,
    rprClientCurrentOutUcastClassAFrames  HCPerfCurrentCount,
    rprClientCurrentOutUcastClassAOctets  HCPerfCurrentCount,
    rprClientCurrentOutUcastClassBCirFrames HCPerfCurrentCount,

```

```

rprClientCurrentOutUcastClassBCirOctets HCPerfCurrentCount,
rprClientCurrentOutUcastClassBEirFrames HCPerfCurrentCount,
rprClientCurrentOutUcastClassBEirOctets HCPerfCurrentCount,
rprClientCurrentOutUcastClassCFrames HCPerfCurrentCount,
rprClientCurrentOutUcastClassCOctets HCPerfCurrentCount,

rprClientCurrentOutMcastClassAFrames HCPerfCurrentCount,
rprClientCurrentOutMcastClassAOctets HCPerfCurrentCount,
rprClientCurrentOutMcastClassBCirFrames HCPerfCurrentCount,
rprClientCurrentOutMcastClassBCirOctets HCPerfCurrentCount,
rprClientCurrentOutMcastClassBEirFrames HCPerfCurrentCount,
rprClientCurrentOutMcastClassBEirOctets HCPerfCurrentCount,
rprClientCurrentOutMcastClassCFrames HCPerfCurrentCount,
rprClientCurrentOutMcastClassCOctets HCPerfCurrentCount
}

rprClientCurrentIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    REFERENCE
        "RFC 2863, ifIndex"
    ::= { rprClientCountersCurrentEntry 1 }

rprClientCurrentInUcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classA unicast frames in
         the current interval."
    ::= { rprClientCountersCurrentEntry 2 }

rprClientCurrentInUcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classA unicast octets in
         the current interval."
    ::= { rprClientCountersCurrentEntry 3 }

rprClientCurrentInUcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classB CIR unicast frames in
         the current interval."
    ::= { rprClientCountersCurrentEntry 4 }

rprClientCurrentInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classB CIR unicast octets

```

```

        in the current interval."
 ::= { rprClientCountersCurrentEntry 5 }

rprClientCurrentInUcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classB EIR unicast
         frames in the current interval."
 ::= { rprClientCountersCurrentEntry 6 }

rprClientCurrentInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classB EIR unicast
         octets in the current interval."
 ::= { rprClientCountersCurrentEntry 7 }

rprClientCurrentInUcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classC unicast frames in the
         current interval."
 ::= { rprClientCountersCurrentEntry 8 }

rprClientCurrentInUcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classC unicast octets in the
         current interval."
 ::= { rprClientCountersCurrentEntry 9 }

rprClientCurrentInMcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classA multicast and broadcast
         frames in the current interval."
 ::= { rprClientCountersCurrentEntry 10 }

rprClientCurrentInMcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classA multicast and broadcast
         octets in the current interval."
 ::= { rprClientCountersCurrentEntry 11 }

rprClientCurrentInMcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount

```

```

MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of MAC to client classB CIR
     multicast and broadcast frames in the current interval."
::= { rprClientCountersCurrentEntry 12 }

rprClientCurrentInMcastClassBCirOctets OBJECT-TYPE
SYNTAX        HCPerfCurrentCount
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of MAC to client classB CIR
     multicast and broadcast octets in the current interval."
::= { rprClientCountersCurrentEntry 13 }

rprClientCurrentInMcastClassBEirFrames OBJECT-TYPE
SYNTAX        HCPerfCurrentCount
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of MAC to client classB EIR
     multicast and broadcast frames in the current interval."
::= { rprClientCountersCurrentEntry 14 }

rprClientCurrentInMcastClassBEirOctets OBJECT-TYPE
SYNTAX        HCPerfCurrentCount
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of MAC to client classB EIR
     multicast and broadcast octets in the current interval."
::= { rprClientCountersCurrentEntry 15 }

rprClientCurrentInMcastClassCFrames OBJECT-TYPE
SYNTAX        HCPerfCurrentCount
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of MAC to client classC multicast and broadcast
     frames in the current interval."
::= { rprClientCountersCurrentEntry 16 }

rprClientCurrentInMcastClassCOctets OBJECT-TYPE
SYNTAX        HCPerfCurrentCount
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of MAC to client classC multicast and broadcast
     octets in the current interval."
::= { rprClientCountersCurrentEntry 17 }

rprClientCurrentOutUcastClassAFrames OBJECT-TYPE
SYNTAX        HCPerfCurrentCount
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of client to MAC classA unicast frames
     in the current interval."

```

```

 ::= { rprClientCountersCurrentEntry 18 }

rprClientCurrentOutUcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classA unicast octets
         in the current interval."
 ::= { rprClientCountersCurrentEntry 19 }

rprClientCurrentOutUcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classB CIR unicast
         frames in the current interval."
 ::= { rprClientCountersCurrentEntry 20 }

rprClientCurrentOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classB CIR unicast
         octets in the current interval."
 ::= { rprClientCountersCurrentEntry 21 }

rprClientCurrentOutUcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classB EIR unicast
         frames in the current interval."
 ::= { rprClientCountersCurrentEntry 22 }

rprClientCurrentOutUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classB EIR unicast
         octets in the current interval."
 ::= { rprClientCountersCurrentEntry 23 }

rprClientCurrentOutUcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classC unicast frames
         in the current interval."
 ::= { rprClientCountersCurrentEntry 24 }

rprClientCurrentOutUcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only

```

```
STATUS      current
DESCRIPTION
    "The number of client to MAC classC unicast octets
     in the current interval."
 ::= { rprClientCountersCurrentEntry 25 }

rprClientCurrentOutMcastClassAFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classA multicast and broadcast
     frames in the current interval."
 ::= { rprClientCountersCurrentEntry 26 }

rprClientCurrentOutMcastClassAOctets OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classA multicast and broadcast
     octets in the current interval."
 ::= { rprClientCountersCurrentEntry 27 }

rprClientCurrentOutMcastClassBCirFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB CIR
     multicast and broadcast frames in the current interval."
 ::= { rprClientCountersCurrentEntry 28 }

rprClientCurrentOutMcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB CIR
     multicast and broadcast octets in the current interval."
 ::= { rprClientCountersCurrentEntry 29 }

rprClientCurrentOutMcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB EIR
     multicast and broadcast frames in the current interval."
 ::= { rprClientCountersCurrentEntry 30 }

rprClientCurrentOutMcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB EIR
     multicast and broadcast octets in the current interval."
 ::= { rprClientCountersCurrentEntry 31 }
```

```

rprClientCurrentOutMcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classC multicast and broadcast
         frames in the current interval."
    ::= { rprClientCountersCurrentEntry 32 }

rprClientCurrentOutMcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classC multicast and broadcast
         octets in the current interval."
    ::= { rprClientCountersCurrentEntry 33 }

-- RPR client interface interval counters table

rprClientCountersIntervalTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprClientCountersIntervalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The local station traffic interval counters table."
    ::= { rprClientCounters 2 }

rprClientCountersIntervalEntry OBJECT-TYPE
    SYNTAX      RprClientCountersIntervalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Frames and octets statistics collected for a particular
         interval for local station traffic of a particular RPR
         interface.
        The corresponding instance of rprIfValidIntervals indicates
         the number of intervals for which the set of statistics is
         available."
    INDEX { rprClientIntervalIfIndex,
            rprClientIntervalNumber }
    ::= { rprClientCountersIntervalTable 1 }

RprClientCountersIntervalEntry ::= SEQUENCE {
    rprClientIntervalIfIndex          InterfaceIndex,
    rprClientIntervalNumber          Unsigned32,
    rprClientIntervalValidData       TruthValue,
    rprClientIntervalTimeElapsed     Unsigned32,

    rprClientIntervalInUcastClassAFrames   HCPerfIntervalCount,
    rprClientIntervalInUcastClassAOctets  HCPerfIntervalCount,
    rprClientIntervalInUcastClassBCirFrames HCPerfIntervalCount,
    rprClientIntervalInUcastClassBCirOctets HCPerfIntervalCount,
    rprClientIntervalInUcastClassBEirFrames HCPerfIntervalCount,
    rprClientIntervalInUcastClassBEirOctets HCPerfIntervalCount,
}

```

```

rprClientIntervalInUcastClassCFrames      HCPerfIntervalCount,
rprClientIntervalInUcastClassCOctets      HCPerfIntervalCount,

rprClientIntervalInMcastClassAFrames      HCPerfIntervalCount,
rprClientIntervalInMcastClassAOctets      HCPerfIntervalCount,
rprClientIntervalInMcastClassBCirFrames   HCPerfIntervalCount,
rprClientIntervalInMcastClassBCirOctets   HCPerfIntervalCount,
rprClientIntervalInMcastClassBEirFrames   HCPerfIntervalCount,
rprClientIntervalInMcastClassBEirOctets   HCPerfIntervalCount,
rprClientIntervalInMcastClassCFrames     HCPerfIntervalCount,
rprClientIntervalInMcastClassCOctets     HCPerfIntervalCount,

rprClientIntervalOutUcastClassAFrames    HCPerfIntervalCount,
rprClientIntervalOutUcastClassAOctets    HCPerfIntervalCount,
rprClientIntervalOutUcastClassBCirFrames HCPerfIntervalCount,
rprClientIntervalOutUcastClassBCirOctets HCPerfIntervalCount,
rprClientIntervalOutUcastClassBEirFrames HCPerfIntervalCount,
rprClientIntervalOutUcastClassBEirOctets HCPerfIntervalCount,
rprClientIntervalOutUcastClassCFrames   HCPerfIntervalCount,
rprClientIntervalOutUcastClassCOctets   HCPerfIntervalCount,

rprClientIntervalOutMcastClassAFrames    HCPerfIntervalCount,
rprClientIntervalOutMcastClassAOctets    HCPerfIntervalCount,
rprClientIntervalOutMcastClassBCirFrames HCPerfIntervalCount,
rprClientIntervalOutMcastClassBCirOctets HCPerfIntervalCount,
rprClientIntervalOutMcastClassBEirFrames HCPerfIntervalCount,
rprClientIntervalOutMcastClassBEirOctets HCPerfIntervalCount,
rprClientIntervalOutMcastClassCFrames   HCPerfIntervalCount,
rprClientIntervalOutMcastClassCOctets   HCPerfIntervalCount
}

rprClientIntervalIfIndex OBJECT-TYPE
SYNTAX      InterfaceIndex
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"The ifIndex of this RPR interface."
::= { rprClientCountersIntervalEntry 1 }

rprClientIntervalNumber OBJECT-TYPE
SYNTAX      Unsigned32 (1..96)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"A number between 1 and 96, which identifies the interval
for which the set of statistics is available. The interval
identified by 1 is the most recently completed 15 minute
interval, and interval identified by N is the interval
immediately preceding the one identified by N-1."
::= { rprClientCountersIntervalEntry 2 }

rprClientIntervalValidData OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"This variable indicates if the data for this interval
is valid.
It will be valid if it contains data for 900 seconds"

```

```

        plus or minus 10 seconds."
 ::= { rprClientCountersIntervalEntry 3 }

rprClientIntervalTimeElapsed OBJECT-TYPE
    SYNTAX      Unsigned32 (0..910)
    UNITS      "Seconds"
    MAX-ACCESS  read-only
    STATUS     current
    DESCRIPTION
        "The duration of a particular interval in seconds.
        If, for some reason, such as an adjustment in the system's
        time-of-day clock, the current interval exceeds the maximum
        value, the agent will return the maximum value."
 ::= { rprClientCountersIntervalEntry 4 }

rprClientIntervalInUcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS     current
    DESCRIPTION
        "The number of MAC to client classA unicast frames
        in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 5 }

rprClientIntervalInUcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS     current
    DESCRIPTION
        "The number of MAC to client classA unicast octets
        in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 6 }

rprClientIntervalInUcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS     current
    DESCRIPTION
        "The number of MAC to client classB CIR unicast frames
        in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 7 }

rprClientIntervalInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS     current
    DESCRIPTION
        "The number of MAC to client classB CIR unicast octets
        in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 8 }

rprClientIntervalInUcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS     current
    DESCRIPTION
        "The number of MAC to client classB EIR unicast
        frames in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 9 }

```

```
rprClientIntervalInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classB EIR unicast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 10 }

rprClientIntervalInUcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classC unicast
         frames in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 11 }

rprClientIntervalInUcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classC unicast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 12 }

rprClientIntervalInMcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classA multicast and broadcast
         frames in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 13 }

rprClientIntervalInMcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classA multicast and broadcast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 14 }

rprClientIntervalInMcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classB CIR multicast and
         broadcast frames in a particular interval in the past
         24 hours."
    ::= { rprClientCountersIntervalEntry 15 }

rprClientIntervalInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
```

```

STATUS      current
DESCRIPTION
    "The number of MAC to client classB CIR multicast and
    broadcast octets in a particular interval in the past
    24 hours."
 ::= { rprClientCountersIntervalEntry 16 }

rprClientIntervalInMcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classB EIR multicast and
    broadcast frames in a particular interval in the past
    24 hours."
 ::= { rprClientCountersIntervalEntry 17 }

rprClientIntervalInMcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classB EIR multicast and
    broadcast octets in a particular interval in the past
    24 hours."
 ::= { rprClientCountersIntervalEntry 18 }

rprClientIntervalInMcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classC multicast and broadcast
    frames in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 19 }

rprClientIntervalInMcastClassCOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classC multicast and broadcast
    octets in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 20 }

rprClientIntervalOutUcastClassAFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classA unicast frames
    in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 21 }

rprClientIntervalOutUcastClassAOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION

```

```
"The number of client to MAC classA unicast octets
in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 22 }

rprClientIntervalOutUcastClassBCirFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of client to MAC classB CIR unicast frames
in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 23 }

rprClientIntervalOutUcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of client to MAC classB CIR unicast octets
in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 24 }

rprClientIntervalOutUcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of client to MAC classB EIR unicast
frames in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 25 }

rprClientIntervalOutUcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of client to MAC classB EIR unicast
octets in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 26 }

rprClientIntervalOutUcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of client to MAC classC unicast
frames in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 27 }

rprClientIntervalOutUcastClassCOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of client to MAC classC unicast
octets in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 28 }

rprClientIntervalOutMcastClassAFrames OBJECT-TYPE
```

```

SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classA multicast and broadcast
     frames in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 29 }

rprClientIntervalOutMcastClassAOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classA multicast and broadcast
     octets in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 30 }

rprClientIntervalOutMcastClassBCirFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB CIR multicast
     and broadcast frames in a particular interval in the
     past 24 hours."
 ::= { rprClientCountersIntervalEntry 31 }

rprClientIntervalOutMcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB CIR multicast
     and broadcast octets in a particular interval in the
     past 24 hours."
 ::= { rprClientCountersIntervalEntry 32 }

rprClientIntervalOutMcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB EIR multicast
     and broadcast frames in a particular interval in the
     past 24 hours."
 ::= { rprClientCountersIntervalEntry 33 }

rprClientIntervalOutMcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfIntervalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB EIR multicast
     and broadcast octets in a particular interval in the
     past 24 hours."
 ::= { rprClientCountersIntervalEntry 34 }

rprClientIntervalOutMcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfIntervalCount

```

```

MAX-ACCESS      read-only
STATUS         current
DESCRIPTION
    "The number of client to MAC classC multicast and broadcast
     frames in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 35 }

rprClientIntervalOutMcastClassCOctets OBJECT-TYPE
    SYNTAX          HCPerfIntervalCount
    MAX-ACCESS      read-only
    STATUS          current
    DESCRIPTION
        "The number of client to MAC classC multicast and broadcast
         octets in a particular interval in the past 24 hours."
 ::= { rprClientCountersIntervalEntry 36 }

--
-- RPR client interface day (24 hour summaries) counters table
--

rprClientCountersDayTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF RprClientCountersDayEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION
        "The RPR Mac Client Day Table contains the cumulative sum
         of the various statistics for the 24 hour period
         preceding the current interval."
 ::= { rprClientCounters 3 }

rprClientCountersDayEntry OBJECT-TYPE
    SYNTAX          RprClientCountersDayEntry
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION
        "An entry in the RPR Client Day table."
 INDEX { rprClientDayIfIndex }
 ::= { rprClientCountersDayTable 1 }

RprClientCountersDayEntry ::= SEQUENCE {
    rprClientDayIfIndex           InterfaceIndex,
    rprClientDayInUcastClassAFrames   HCPerfTotalCount,
    rprClientDayInUcastClassAOctets   HCPerfTotalCount,
    rprClientDayInUcastClassBCirFrames  HCPerfTotalCount,
    rprClientDayInUcastClassBCirOctets  HCPerfTotalCount,
    rprClientDayInUcastClassBEirFrames  HCPerfTotalCount,
    rprClientDayInUcastClassBEirOctets  HCPerfTotalCount,
    rprClientDayInUcastClassCFrames   HCPerfTotalCount,
    rprClientDayInUcastClassCOctets   HCPerfTotalCount,
    rprClientDayInMcastClassAFrames   HCPerfTotalCount,
    rprClientDayInMcastClassAOctets   HCPerfTotalCount,
    rprClientDayInMcastClassBCirFrames  HCPerfTotalCount,
    rprClientDayInMcastClassBCirOctets  HCPerfTotalCount,
    rprClientDayInMcastClassBEirFrames  HCPerfTotalCount,
    rprClientDayInMcastClassBEirOctets  HCPerfTotalCount,
    rprClientDayInMcastClassCFrames   HCPerfTotalCount,
    rprClientDayInMcastClassCOctets   HCPerfTotalCount,
}

```

```

rprClientDayOutUcastClassAFrames      HCPerfTotalCount,
rprClientDayOutUcastClassAOctets     HCPerfTotalCount,
rprClientDayOutUcastClassBCirFrames  HCPerfTotalCount,
rprClientDayOutUcastClassBCirOctets  HCPerfTotalCount,
rprClientDayOutUcastClassBEirFrames  HCPerfTotalCount,
rprClientDayOutUcastClassBEirOctets  HCPerfTotalCount,
rprClientDayOutUcastClassCFrames    HCPerfTotalCount,
rprClientDayOutUcastClassCOctets   HCPerfTotalCount,

rprClientDayOutMcastClassAFrames     HCPerfTotalCount,
rprClientDayOutMcastClassAOctets   HCPerfTotalCount,
rprClientDayOutMcastClassBCirFrames HCPerfTotalCount,
rprClientDayOutMcastClassBCirOctets HCPerfTotalCount,
rprClientDayOutMcastClassBEirFrames HCPerfTotalCount,
rprClientDayOutMcastClassBEirOctets HCPerfTotalCount,
rprClientDayOutMcastClassCFrames   HCPerfTotalCount,
rprClientDayOutMcastClassCOctets  HCPerfTotalCount
}

rprClientDayIfIndex OBJECT-TYPE
  SYNTAX      InterfaceIndex
  MAX-ACCESS  not-accessible
  STATUS      current
  DESCRIPTION
    "The ifIndex of this RPR interface."
  ::= { rprClientCountersDayEntry 1 }

rprClientDayInUcastClassAFrames OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of MAC to client classA unicast frames."
  ::= { rprClientCountersDayEntry 2 }

rprClientDayInUcastClassAOctets OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of MAC to client classA unicast octets."
  ::= { rprClientCountersDayEntry 3 }

rprClientDayInUcastClassBCirFrames OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of MAC to client classB CIR unicast frames."
  ::= { rprClientCountersDayEntry 4 }

rprClientDayInUcastClassBCirOctets OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of MAC to client classB CIR unicast octets."
  ::= { rprClientCountersDayEntry 5 }

```

```
rprClientDayInUcastClassBEirFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classB EIR unicast frames."
    ::= { rprClientCountersDayEntry 6 }

rprClientDayInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classB EIR unicast octets."
    ::= { rprClientCountersDayEntry 7 }

rprClientDayInUcastClassCFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classC unicast frames."
    ::= { rprClientCountersDayEntry 8 }

rprClientDayInUcastClassCOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classC unicast octets."
    ::= { rprClientCountersDayEntry 9 }

rprClientDayInMcastClassAFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classA multicast and broadcast frames."
    ::= { rprClientCountersDayEntry 10 }

rprClientDayInMcastClassAOctets OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classA multicast and broadcast octets."
    ::= { rprClientCountersDayEntry 11 }

rprClientDayInMcastClassBCirFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client classB CIR multicast
         and broadcast frames."
    ::= { rprClientCountersDayEntry 12 }

rprClientDayInMcastClassBCirOctets OBJECT-TYPE
```

```

SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classB CIR multicast
    and broadcast octets."
 ::= { rprClientCountersDayEntry 13 }

rprClientDayInMcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classB EIR multicast
    and broadcast frames."
 ::= { rprClientCountersDayEntry 14 }

rprClientDayInMcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classB EIR multicast
    and broadcast octets."
 ::= { rprClientCountersDayEntry 15 }

rprClientDayInMcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classC multicast and broadcast frames."
 ::= { rprClientCountersDayEntry 16 }

rprClientDayInMcastClassCOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classC multicast and broadcast octets."
 ::= { rprClientCountersDayEntry 17 }

rprClientDayOutUcastClassAFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classA unicast frames."
 ::= { rprClientCountersDayEntry 18 }

rprClientDayOutUcastClassAOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classA unicast octets."
 ::= { rprClientCountersDayEntry 19 }

rprClientDayOutUcastClassBCirFrames OBJECT-TYPE

```

```
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB CIR unicast frames."
::= { rprClientCountersDayEntry 20 }

rprClientDayOutUcastClassBCirOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB CIR unicast octets."
::= { rprClientCountersDayEntry 21 }

rprClientDayOutUcastClassBEirFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB EIR unicast frames."
::= { rprClientCountersDayEntry 22 }

rprClientDayOutUcastClassBEirOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB EIR unicast octets."
::= { rprClientCountersDayEntry 23 }

rprClientDayOutUcastClassCFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classC unicast frames."
::= { rprClientCountersDayEntry 24 }

rprClientDayOutUcastClassCOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classC unicast octets."
::= { rprClientCountersDayEntry 25 }

rprClientDayOutMcastClassAFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classA multicast and broadcast
     frames."
::= { rprClientCountersDayEntry 26 }

rprClientDayOutMcastClassAOctets OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
```

```

STATUS      current
DESCRIPTION
  "The number of client to MAC classA multicast and broadcast
  octets."
 ::= { rprClientCountersDayEntry 27 }

rprClientDayOutMcastClassBCirFrames OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of client to MAC classB CIR
    multicast and broadcast frames."
 ::= { rprClientCountersDayEntry 28 }

rprClientDayOutMcastClassBCirOctets OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of client to MAC classB CIR
    multicast and broadcast octets."
 ::= { rprClientCountersDayEntry 29 }

rprClientDayOutMcastClassBEirFrames OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of client to MAC classB EIR
    multicast and broadcast frames."
 ::= { rprClientCountersDayEntry 30 }

rprClientDayOutMcastClassBEirOctets OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of client to MAC classB EIR
    multicast and broadcast octets."
 ::= { rprClientCountersDayEntry 31 }

rprClientDayOutMcastClassCFrames OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of client to MAC classC multicast and broadcast frames."
 ::= { rprClientCountersDayEntry 32 }

rprClientDayOutMcastClassCOctets OBJECT-TYPE
  SYNTAX      HCPerfTotalCount
  MAX-ACCESS  read-only
  STATUS      current
  DESCRIPTION
    "The number of client to MAC classC multicast and broadcast octets."
 ::= { rprClientCountersDayEntry 33 }

-- 

```

```
-- RPR client interface continuously running counters table
--

rprClientCountersStatsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprClientCountersStatsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The local station traffic total counters table.

        The DiscontinuityTime for this table is indicated by
        ifCounterDiscontinuityTime defined in ifXTable."
 ::= { rprClientCounters 4 }

rprClientCountersStatsEntry OBJECT-TYPE
    SYNTAX      RprClientCountersStatsEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry in the span stats table."
    INDEX { rprClientStatsIfIndex }
 ::= { rprClientCountersStatsTable 1 }

RprClientCountersStatsEntry ::= SEQUENCE {
    rprClientStatsIfIndex           InterfaceIndex,
    rprClientStatsInUcastClassAFrames   Counter64,
    rprClientStatsInUcastClassAOctets   Counter64,
    rprClientStatsInUcastClassBCirFrames Counter64,
    rprClientStatsInUcastClassBCirOctets Counter64,
    rprClientStatsInUcastClassBEirFrames Counter64,
    rprClientStatsInUcastClassBEirOctets Counter64,
    rprClientStatsInUcastClassCFFrames Counter64,
    rprClientStatsInUcastClassCOctets   Counter64,
    rprClientStatsInMcastClassAFrames   Counter64,
    rprClientStatsInMcastClassAOctets   Counter64,
    rprClientStatsInMcastClassBCirFrames Counter64,
    rprClientStatsInMcastClassBCirOctets Counter64,
    rprClientStatsInMcastClassBEirFrames Counter64,
    rprClientStatsInMcastClassBEirOctets Counter64,
    rprClientStatsInMcastClassCFFrames Counter64,
    rprClientStatsInMcastClassCOctets   Counter64,
    rprClientStatsInBcastFrames        Counter64,
    rprClientStatsOutUcastClassAFrames  Counter64,
    rprClientStatsOutUcastClassAOctets  Counter64,
    rprClientStatsOutUcastClassBCirFrames Counter64,
    rprClientStatsOutUcastClassBCirOctets Counter64,
    rprClientStatsOutUcastClassBEirFrames Counter64,
    rprClientStatsOutUcastClassBEirOctets Counter64,
    rprClientStatsOutUcastClassCFFrames Counter64,
    rprClientStatsOutUcastClassCOctets  Counter64,
    rprClientStatsOutMcastClassAFrames  Counter64,
    rprClientStatsOutMcastClassAOctets  Counter64,
    rprClientStatsOutMcastClassBCirFrames Counter64,
    rprClientStatsOutMcastClassBCirOctets Counter64,
```

```

rprClientStatsOutMcastClassBEirFrames Counter64,
rprClientStatsOutMcastClassBEirOctets Counter64,
rprClientStatsOutMcastClassCFrames Counter64,
rprClientStatsOutMcastClassCOctets Counter64,

rprClientStatsOutBcastFrames Counter64
}

rprClientStatsIfIndex OBJECT-TYPE
SYNTAX InterfaceIndex
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The ifIndex of this RPR interface."
::= { rprClientCountersStatsEntry 1 }

rprClientStatsInUcastClassAFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of MAC to client classA unicast frames."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, toClientUcastClassAFrames"
::= { rprClientCountersStatsEntry 2 }

rprClientStatsInUcastClassAOctets OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of MAC to client classA unicast octets."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, toClientUcastClassABytes"
::= { rprClientCountersStatsEntry 3 }

rprClientStatsInUcastClassBCirFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of MAC to client classB CIR unicast frames."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, toClientUcastClassBCirFrames"
::= { rprClientCountersStatsEntry 4 }

rprClientStatsInUcastClassBCirOctets OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The number of MAC to client classB CIR unicast octets."
REFERENCE
"IEEE 802.17 Subclause 7.2.5, toClientUcastClassBCirBytes"
::= { rprClientCountersStatsEntry 5 }

rprClientStatsInUcastClassBEirFrames OBJECT-TYPE
SYNTAX Counter64
MAX-ACCESS read-only

```

```
STATUS      current
DESCRIPTION
    "The number of MAC to client classB EIR unicast frames."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, toClientUcastClassBEirFrames"
::= { rprClientCountersStatsEntry 6 }

rprClientStatsInUcastClassBEirOctets OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classB EIR unicast octets."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, toClientUcastClassBEirBytes"
::= { rprClientCountersStatsEntry 7 }

rprClientStatsInUcastClassCFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classC unicast frames."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, toClientUcastClassCFrames"
::= { rprClientCountersStatsEntry 8 }

rprClientStatsInUcastClassCOctets OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classC unicast octets."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, toClientUcastClassCBytes"
::= { rprClientCountersStatsEntry 9 }

rprClientStatsInMcastClassAFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classA multicast and broadcast frames."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, toClientMcastClassAFrames"
::= { rprClientCountersStatsEntry 10 }

rprClientStatsInMcastClassAOctets OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of MAC to client classA multicast and broadcast octets."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, toClientMcastClassABytes"
::= { rprClientCountersStatsEntry 11 }

rprClientStatsInMcastClassBCirFrames OBJECT-TYPE
SYNTAX      Counter64
```

```

MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "The number of MAC to client classB CIR multicast
  and broadcast frames."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, toClientMcastClassBCirFrames"
  ::= { rprClientCountersStatsEntry 12 }

rprClientStatsInMcastClassBCirOctets OBJECT-TYPE
SYNTAX        Counter64
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "The number of MAC to client classB CIR multicast
  and broadcast octets."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, toClientMcastClassBCirBytes"
  ::= { rprClientCountersStatsEntry 13 }

rprClientStatsInMcastClassBEirFrames OBJECT-TYPE
SYNTAX        Counter64
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "The number of MAC to client classB EIR multicast
  and broadcast frames."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, toClientMcastClassBEirFrames"
  ::= { rprClientCountersStatsEntry 14 }

rprClientStatsInMcastClassBEirOctets OBJECT-TYPE
SYNTAX        Counter64
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "The number of MAC to client classB EIR multicast
  and broadcast octets."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, toClientMcastClassBEirBytes"
  ::= { rprClientCountersStatsEntry 15 }

rprClientStatsInMcastClassCFrames OBJECT-TYPE
SYNTAX        Counter64
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "The number of MAC to client classC multicast and broadcast frames."
REFERENCE
  "IEEE 802.17 Subclause 7.2.5, toClientMcastClassCFrames"
  ::= { rprClientCountersStatsEntry 16 }

rprClientStatsInMcastClassCOctets OBJECT-TYPE
SYNTAX        Counter64
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
  "The number of MAC to client classC multicast and broadcast octets."
REFERENCE

```

```
"IEEE 802.17 Subclause 7.2.5, toClientMcastClassCBytes"
 ::= { rprClientCountersStatsEntry 17 }

rprClientStatsInBcastFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of MAC to client broadcast frames.
         This is used only when deriving the multicast
         and broadcast packet counters for the interface MIB."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, toClientBcastFrames"
        ::= { rprClientCountersStatsEntry 18 }

rprClientStatsOutUcastClassAFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classA unicast frames."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, fromClientUcastClassAFrames"
        ::= { rprClientCountersStatsEntry 19 }

rprClientStatsOutUcastClassAOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classA unicast octets."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, fromClientUcastClassABytes"
        ::= { rprClientCountersStatsEntry 20 }

rprClientStatsOutUcastClassBCirFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classB CIR unicast frames."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, fromClientUcastClassBCirFrames"
        ::= { rprClientCountersStatsEntry 21 }

rprClientStatsOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classB CIR unicast octets."
    REFERENCE
        "IEEE 802.17 Subclause 7.2.5, fromClientUcastClassBCirBytes"
        ::= { rprClientCountersStatsEntry 22 }

rprClientStatsOutUcastClassBEirFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
```

```

DESCRIPTION
    "The number of client to MAC classB EIR unicast frames."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientUcastClassBEirFrames"
    ::= { rprClientCountersStatsEntry 23 }

rprClientStatsOutUcastClassBEirOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classB EIR unicast octets."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientUcastClassBEirBytes"
    ::= { rprClientCountersStatsEntry 24 }

rprClientStatsOutUcastClassCFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classC unicast frames."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientUcastClassCFrames"
    ::= { rprClientCountersStatsEntry 25 }

rprClientStatsOutUcastClassCOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classC unicast octets."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientUcastClassCBytes"
    ::= { rprClientCountersStatsEntry 26 }

rprClientStatsOutMcastClassAFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classA multicast and broadcast
         frames."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientMcastClassAFrames"
    ::= { rprClientCountersStatsEntry 27 }

rprClientStatsOutMcastClassAOctets OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of client to MAC classA multicast and broadcast
         octets."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientMcastClassABytes"
    ::= { rprClientCountersStatsEntry 28 }

rprClientStatsOutMcastClassBCirFrames OBJECT-TYPE

```

```
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB CIR
     multicast and broadcast frames."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientMcastClassBCirFrames"
::= { rprClientCountersStatsEntry 29 }

rprClientStatsOutMcastClassBCirOctets OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB CIR
     multicast and broadcast octets."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientMcastClassBCirBytes"
::= { rprClientCountersStatsEntry 30 }

rprClientStatsOutMcastClassBEirFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB EIR
     multicast and broadcast frames."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientMcastClassBEirFrames"
::= { rprClientCountersStatsEntry 31 }

rprClientStatsOutMcastClassBEirOctets OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classB EIR
     multicast and broadcast octets."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientMcastClassBEirBytes"
::= { rprClientCountersStatsEntry 32 }

rprClientStatsOutMcastClassCFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of client to MAC classC multicast and broadcast
     frames."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientMcastClassCFrames"
::= { rprClientCountersStatsEntry 33 }

rprClientStatsOutMcastClassCOctets OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
```

```

    "The number of client to MAC classC multicast and broadcast
    octets."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientMcastClassCBytes"
::= { rprClientCountersStatsEntry 34 }

rprClientStatsOutBcastFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS   read-only
STATUS       current
DESCRIPTION
    "The number of client to MAC broadcast frames.
    This is used only when deriving the multicast
    and broadcast packet counters for the interface MIB."
REFERENCE
    "IEEE 802.17 Subclause 7.2.5, fromClientBcastFrames"
::= { rprClientCountersStatsEntry 35 }

-- RPR error current counters
--

rprSpanErrorCountersCurrentTable OBJECT-TYPE
SYNTAX      SEQUENCE OF RprSpanErrorCountersCurrentEntry
MAX-ACCESS   not-accessible
STATUS       current
DESCRIPTION
    "The RPR Errors Current counters table."
::= { rprSpanErrorCounters 1 }

rprSpanErrorCountersCurrentEntry OBJECT-TYPE
SYNTAX      RprSpanErrorCountersCurrentEntry
MAX-ACCESS   not-accessible
STATUS       current
DESCRIPTION
    "Errors statistics for the current interval of a particular
    span of a particular RPR interface.
    The corresponding instance of rprIfTimeElapsed indicates
    the number of seconds which have elapsed so far in the
    current interval."
INDEX { rprSpanErrorCurrentIfIndex,
         rprSpanErrorCurrentSpan }
::= { rprSpanErrorCountersCurrentTable 1 }

RprSpanErrorCountersCurrentEntry ::= SEQUENCE {
    rprSpanErrorCurrentIfIndex           InterfaceIndex,
    rprSpanErrorCurrentSpan            RprSpan,
    rprSpanErrorCurrentTtlExpFrames    HCPerfCurrentCount,
    rprSpanErrorCurrentTooLongFrames   HCPerfCurrentCount,
    rprSpanErrorCurrentTooShortFrames  HCPerfCurrentCount,
    rprSpanErrorCurrentBadHecFrames    HCPerfCurrentCount,
    rprSpanErrorCurrentBadFcsFrames   HCPerfCurrentCount,
    rprSpanErrorCurrentSelfSrcUcastFrames HCPerfCurrentCount,
    rprSpanErrorCurrentPmdAbortFrames  HCPerfCurrentCount,
    rprSpanErrorCurrentBadAddrFrames   HCPerfCurrentCount,
    rprSpanErrorCurrentBadParityFrames HCPerfCurrentCount,
    rprSpanErrorCurrentContainedFrames HCPerfCurrentCount,
}

```

```

rprSpanErrorCurrentScffErrors          HCPerfCurrentCount,
rprSpanErrorCurrentErroredSeconds     HCPerfCurrentCount,
rprSpanErrorCurrentSeverelyErroredSeconds HCPerfCurrentCount,
rprSpanErrorCurrentUnavailableSeconds HCPerfCurrentCount
}

rprSpanErrorCurrentIfIndex OBJECT-TYPE
SYNTAX      InterfaceIndex
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "The ifIndex of this RPR interface."
REFERENCE
    "RFC 2863, ifIndex"
::= { rprSpanErrorCountersCurrentEntry 1 }

rprSpanErrorCurrentSpan OBJECT-TYPE
SYNTAX      RprSpan
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "An indication of the span of the interface for which this
     row contains information."
REFERENCE
    "IEEE 802.17 Subclause 7.2.2, myRi"
::= { rprSpanErrorCountersCurrentEntry 2 }

rprSpanErrorCurrentTtlExpFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of received (PHY to MAC) data frames that were dropped
     due to zero Time To Live (TTL)."
REFERENCE
    "IEEE 802.17 Subclause 7.6.3.7.2 and 7.6.3.9.2, ttlExpiredFrames"
::= { rprSpanErrorCountersCurrentEntry 3 }

rprSpanErrorCurrentTooLongFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of received (PHY to MAC) frames that exceed the
     maximum permitted frame size."
REFERENCE
    "IEEE 802.17 Subclause 7.6.3.7.2, tooLongFrames"
::= { rprSpanErrorCountersCurrentEntry 4 }

rprSpanErrorCurrentTooShortFrames OBJECT-TYPE
SYNTAX      HCPerfCurrentCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of received (PHY to MAC) frames shortest than the
     minimum permitted frame size."
REFERENCE
    "IEEE 802.17 Subclause 7.6.3.7.2, tooShortFrames"
::= { rprSpanErrorCountersCurrentEntry 5 }

```

```

rprSpanErrorCurrentBadHecFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames with HEC error."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.7.2, badHecFrames"
        ::= { rprSpanErrorCountersCurrentEntry 6 }

rprSpanErrorCurrentBadFcsFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) data and control frames
         where the fcs value did not match the expected fcs value.

        This includes data frames passed to the client as a result of
         rprIfMacOperModes being not set to dropBadFcs."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.7.2 and 7.6.3.11.2, badFcsFrames"
        ::= { rprSpanErrorCountersCurrentEntry 7 }

rprSpanErrorCurrentSelfSrcUcastFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) unicast frames that were
         transmitted by the station itself.
         I.e., the source MAC is equal to the interface MAC."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.9.2, selfSourcedFrames"
        ::= { rprSpanErrorCountersCurrentEntry 8 }

rprSpanErrorCurrentPmdAbortFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames that were
         aborted by the PMD."
    REFERENCE
        "IEEE 802.17 Subclause 8.2.2.2, PHY_DATA.indication.STATUS"
        ::= { rprSpanErrorCountersCurrentEntry 9 }

rprSpanErrorCurrentBadAddrFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames
         with invalid SA value."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.7.2, badAddressFrames"
        ::= { rprSpanErrorCountersCurrentEntry 10 }

```

```
rprSpanErrorCurrentBadParityFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames
         parity value not matching the expected parity value."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.7.2, badParityFrames"
        ::= { rprSpanErrorCountersCurrentEntry 11 }

rprSpanErrorCurrentContainedFrames OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames
         that were removed due to context containment."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.7.2, 7.6.3.9.2,
         and 7.6.3.11.2, containedFrames"
        ::= { rprSpanErrorCountersCurrentEntry 12 }

rprSpanErrorCurrentScffErrors OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) errored SCFF,
         with bad parity, bad FCS, or both."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.7.2, scffErrors"
        ::= { rprSpanErrorCountersCurrentEntry 13 }

rprSpanErrorCurrentErroredSeconds OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of errored seconds."
    REFERENCE
        "IEEE 802.17 Subclause 12.6.1.3, erroredSeconds"
        ::= { rprSpanErrorCountersCurrentEntry 14 }

rprSpanErrorCurrentSeverelyErroredSeconds OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of severely errored seconds."
    REFERENCE
        "IEEE 802.17 Subclause 12.6.1.4, severelyErroredSeconds"
        ::= { rprSpanErrorCountersCurrentEntry 15 }

rprSpanErrorCurrentUnavailableSeconds OBJECT-TYPE
    SYNTAX      HCPerfCurrentCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
```

```

        "The number of unavailable seconds."
REFERENCE
        "IEEE 802.17 Subclause 12.6.2, unavailableSeconds"
::= { rprSpanErrorCountersCurrentEntry 16 }

-- 
-- RPR error interval counters table
--

rprSpanErrorCountersIntervalTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprSpanErrorCountersIntervalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The RPR Errors Interval counters table."
::= { rprSpanErrorCounters 2 }

rprSpanErrorCountersIntervalEntry OBJECT-TYPE
    SYNTAX      RprSpanErrorCountersIntervalEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Error statistics collected for a particular interval of a
        particular span of a particular RPR interface.
        The corresponding instance of rprIfValidIntervals indicates
        the number of intervals for which the set of statistics is
        available."
    INDEX { rprSpanErrorIntervalIfIndex,
            rprSpanErrorIntervalSpan,
            rprSpanErrorIntervalNumber }
::= { rprSpanErrorCountersIntervalTable 1 }

RprSpanErrorCountersIntervalEntry ::= SEQUENCE {
    rprSpanErrorIntervalIfIndex           InterfaceIndex,
    rprSpanErrorIntervalSpan             RprSpan,
    rprSpanErrorIntervalNumber           Unsigned32,
    rprSpanErrorIntervalValidData       TruthValue,
    rprSpanErrorIntervalTimeElapsed     Unsigned32,
    rprSpanErrorIntervalTtlExpFrames    HCPerfIntervalCount,
    rprSpanErrorIntervalTooLongFrames   HCPerfIntervalCount,
    rprSpanErrorIntervalTooShortFrames  HCPerfIntervalCount,
    rprSpanErrorIntervalBadHecFrames    HCPerfIntervalCount,
    rprSpanErrorIntervalBadFcsFrames   HCPerfIntervalCount,
    rprSpanErrorIntervalSelfSrcUcastFrames HCPerfIntervalCount,
    rprSpanErrorIntervalPmdAbortFrames  HCPerfIntervalCount,
    rprSpanErrorIntervalBadAddrFrames   HCPerfIntervalCount,
    rprSpanErrorIntervalBadParityFrames HCPerfIntervalCount,
    rprSpanErrorIntervalContainedFrames HCPerfIntervalCount,
    rprSpanErrorIntervalScffErrors      HCPerfIntervalCount,
    rprSpanErrorIntervalErroredSeconds  HCPerfIntervalCount,
    rprSpanErrorIntervalSeverelyErroredSeconds HCPerfIntervalCount,
    rprSpanErrorIntervalUnavailableSeconds HCPerfIntervalCount
}

rprSpanErrorIntervalIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex

```

```

MAX-ACCESS    not-accessible
STATUS        current
DESCRIPTION
    "The ifIndex of this RPR interface."
::= { rprSpanErrorCountersIntervalEntry 1 }

rprSpanErrorIntervalSpan OBJECT-TYPE
SYNTAX        RprSpan
MAX-ACCESS    not-accessible
STATUS        current
DESCRIPTION
    "An indication of the span of the interface for which this
    row contains information."
::= { rprSpanErrorCountersIntervalEntry 2 }

rprSpanErrorIntervalNumber OBJECT-TYPE
SYNTAX        Unsigned32 (1..96)
MAX-ACCESS    not-accessible
STATUS        current
DESCRIPTION
    "A number between 1 and 96, which identifies the interval
    for which the set of statistics is available. The interval
    identified by 1 is the most recently completed 15 minute
    interval, and interval identified by N is the interval
    immediately preceding the one identified by N-1."
::= { rprSpanErrorCountersIntervalEntry 3 }

rprSpanErrorIntervalValidData OBJECT-TYPE
SYNTAX        TruthValue
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "This variable indicates if the data for this interval
    is valid.
    It will be valid if it contains data for 900 seconds
    plus or minus 10 seconds."
::= { rprSpanErrorCountersIntervalEntry 4 }

rprSpanErrorIntervalTimeElapsed OBJECT-TYPE
SYNTAX        Unsigned32 (0..910)
UNITS         "Seconds"
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The duration of a particular interval in seconds.
    If, for some reason, such as an adjustment in the system's
    time-of-day clock, the current interval exceeds the maximum
    value, the agent will return the maximum value."
::= { rprSpanErrorCountersIntervalEntry 5 }

rprSpanErrorIntervalTtlExpFrames OBJECT-TYPE
SYNTAX        HCPerfIntervalCount
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of received (PHY to MAC) frames that were dropped due to
    zero Time To Live (TTL) in a particular interval in the
    past 24 hours."
::= { rprSpanErrorCountersIntervalEntry 6 }

```

```

rprSpanErrorIntervalTooLongFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames that exceed the maximum
         permitted frame size in a particular interval in the
         past 24 hours."
    ::= { rprSpanErrorCountersIntervalEntry 7 }

rprSpanErrorIntervalTooShortFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames shortest than the
         minimum permitted frame size in a particular interval
         in the past 24 hours."
    ::= { rprSpanErrorCountersIntervalEntry 8 }

rprSpanErrorIntervalBadHecFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames with HEC error in a
         particular interval in the past 24 hours."
    ::= { rprSpanErrorCountersIntervalEntry 9 }

rprSpanErrorIntervalBadFcsFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) data and control frames
         where the fcs value did not match the expected fcs value.

        This includes data frames passed to the client as a result of
        rprIfMacOperModes being not set to dropBadFcs."
    ::= { rprSpanErrorCountersIntervalEntry 10 }

rprSpanErrorIntervalSelfSrcUcastFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) unicast frames that were transmitted
         by the station itself. I.e., the source MAC is equal to the
         interface MAC, in a particular interval in the past 24 hours."
    ::= { rprSpanErrorCountersIntervalEntry 11 }

rprSpanErrorIntervalPmdAbortFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) unicast frames that were aborted
         by the PMD layer, in a particular interval in the past 24 hours."

```

```

 ::= { rprSpanErrorCountersIntervalEntry 12 }

rprSpanErrorIntervalBadAddrFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames
         with invalid SA value."
 ::= { rprSpanErrorCountersIntervalEntry 13 }

rprSpanErrorIntervalBadParityFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames
         parity value not matching the expected parity value."
 ::= { rprSpanErrorCountersIntervalEntry 14 }

rprSpanErrorIntervalContainedFrames OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames
         that were removed due to context containment."
 ::= { rprSpanErrorCountersIntervalEntry 15 }

rprSpanErrorIntervalScffErrors OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) errored SCFF,
         with bad parity, bad FCS, or both."
 ::= { rprSpanErrorCountersIntervalEntry 16 }

rprSpanErrorIntervalErroredSeconds OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of errored seconds."
 ::= { rprSpanErrorCountersIntervalEntry 17 }

rprSpanErrorIntervalSeverelyErroredSeconds OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of severely errored seconds."
 ::= { rprSpanErrorCountersIntervalEntry 18 }

rprSpanErrorIntervalUnavailableSeconds OBJECT-TYPE
    SYNTAX      HCPerfIntervalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION

```

```

    "The number of unavailable seconds."
    ::= { rprSpanErrorCountersIntervalEntry 19 }

-- 
-- RPR error day (24 hour summaries) counters table
--

rprSpanErrorCountersDayTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprSpanErrorCountersDayEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The RPR Mac Error Day Table contains the cumulative sum
        of the various statistics for the 24 hour period
        preceding the current interval."
    ::= { rprSpanErrorCounters 3 }

rprSpanErrorCountersDayEntry OBJECT-TYPE
    SYNTAX      RprSpanErrorCountersDayEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry in the RPR Error Day table."
    INDEX { rprSpanErrorDayIfIndex,
            rprSpanErrorDaySpan }
    ::= { rprSpanErrorCountersDayTable 1 }

RprSpanErrorCountersDayEntry ::= SEQUENCE {
    rprSpanErrorDayIfIndex           InterfaceIndex,
    rprSpanErrorDaySpan             RprSpan,
    rprSpanErrorDayTtlExpFrames     HCPerfTotalCount,
    rprSpanErrorDayTooLongFrames    HCPerfTotalCount,
    rprSpanErrorDayTooShortFrames   HCPerfTotalCount,
    rprSpanErrorDayBadHecFrames    HCPerfTotalCount,
    rprSpanErrorDayBadFcsFrames    HCPerfTotalCount,
    rprSpanErrorDaySelfSrcUcastFrames HCPerfTotalCount,
    rprSpanErrorDayPmdAbortFrames   HCPerfTotalCount,
    rprSpanErrorDayBadAddrFrames   HCPerfTotalCount,
    rprSpanErrorDayBadParityFrames HCPerfTotalCount,
    rprSpanErrorDayContainedFrames HCPerfTotalCount,
    rprSpanErrorDayScffErrors       HCPerfTotalCount,
    rprSpanErrorDayErroredSeconds   HCPerfTotalCount,
    rprSpanErrorDaySeverelyErroredSeconds HCPerfTotalCount,
    rprSpanErrorDayUnavailableSeconds HCPerfTotalCount
}

rprSpanErrorDayIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    ::= { rprSpanErrorCountersDayEntry 1 }

rprSpanErrorDaySpan OBJECT-TYPE
    SYNTAX      RprSpan
    MAX-ACCESS  not-accessible

```

```

STATUS      current
DESCRIPTION
    "An indication of the span of the interface for which this
     row contains information."
 ::= { rprSpanErrorCountersDayEntry 2 }

rprSpanErrorDayTtlExpFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames that were dropped
         due to zero Time To Live (TTL)."
 ::= { rprSpanErrorCountersDayEntry 3 }

rprSpanErrorDayTooLongFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames that exceed the
         maximum permitted frame size."
 ::= { rprSpanErrorCountersDayEntry 4 }

rprSpanErrorDayTooShortFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames shortest than the
         minimum permitted frame size."
 ::= { rprSpanErrorCountersDayEntry 5 }

rprSpanErrorDayBadHecFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames with HEC error."
 ::= { rprSpanErrorCountersDayEntry 6 }

rprSpanErrorDayBadFcsFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) data and control frames
         where the fcs value did not match the expected fcs value.

        This includes data frames passed to the client as a result of
         rprIfMacOperModes being not set to dropBadFcs."
 ::= { rprSpanErrorCountersDayEntry 7 }

rprSpanErrorDaySelfSrcUcastFrames OBJECT-TYPE
    SYNTAX      HCPerfTotalCount
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) unicast frames that were

```

```

transmitted by the station itself.
I.e., the source MAC is equal to the interface MAC."
 ::= { rprSpanErrorCountersDayEntry 8 }

rprSpanErrorDayPmdAbortFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of received (PHY to MAC) frames that were
aborted by the PMD."
 ::= { rprSpanErrorCountersDayEntry 9 }

rprSpanErrorDayBadAddrFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of received (PHY to MAC) frames
with invalid SA value."
 ::= { rprSpanErrorCountersDayEntry 10 }

rprSpanErrorDayBadParityFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of received (PHY to MAC) frames
parity value not matching the expected parity value."
 ::= { rprSpanErrorCountersDayEntry 11 }

rprSpanErrorDayContainedFrames OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of received (PHY to MAC) frames
that were removed due to context containment."
 ::= { rprSpanErrorCountersDayEntry 12 }

rprSpanErrorDayScffErrors OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of received (PHY to MAC) errored SCFF,
with bad parity, bad FCS, or both."
 ::= { rprSpanErrorCountersDayEntry 13 }

rprSpanErrorDayErroredSeconds OBJECT-TYPE
SYNTAX      HCPerfTotalCount
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The number of errored seconds."
 ::= { rprSpanErrorCountersDayEntry 14 }

rprSpanErrorDaySeverelyErroredSeconds OBJECT-TYPE
SYNTAX      HCPerfTotalCount

```

```

MAX-ACCESS      read-only
STATUS         current
DESCRIPTION
    "The number of severely errored seconds."
::= { rprSpanErrorCountersDayEntry 15 }

rprSpanErrorDayUnavailableSeconds OBJECT-TYPE
    SYNTAX          HCPerfTotalCount
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of unavailable seconds."
    ::= { rprSpanErrorCountersDayEntry 16 }

--
-- RPR error total continuously running counters table
--

rprSpanErrorCountersStatsTable OBJECT-TYPE
    SYNTAX          SEQUENCE OF RprSpanErrorCountersStatsEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "The RPR Errors total counters table.

        The DiscontinuityTime for this table is indicated by
        ifCounterDiscontinuityTime defined in ifXTable."
    ::= { rprSpanErrorCounters 4 }

rprSpanErrorCountersStatsEntry OBJECT-TYPE
    SYNTAX          RprSpanErrorCountersStatsEntry
    MAX-ACCESS     not-accessible
    STATUS         current
    DESCRIPTION
        "An entry in the span error counter table."
    INDEX { rprSpanErrorStatsIfIndex,
            rprSpanErrorStatsSpan }
    ::= { rprSpanErrorCountersStatsTable 1 }

RprSpanErrorCountersStatsEntry ::= SEQUENCE {
    rprSpanErrorStatsIfIndex           InterfaceIndex,
    rprSpanErrorStatsSpan             RprSpan,
    rprSpanErrorStatsTtlExpFrames    Counter64,
    rprSpanErrorStatsTooLongFrames   Counter64,
    rprSpanErrorStatsTooShortFrames  Counter64,
    rprSpanErrorStatsBadHecFrames    Counter64,
    rprSpanErrorStatsBadFcsFrames   Counter64,
    rprSpanErrorStatsSelfSrcUcastFrames Counter64,
    rprSpanErrorStatsPmdAbortFrames  Counter64,
    rprSpanErrorStatsBadAddrFrames   Counter64,
    rprSpanErrorStatsBadParityFrames Counter64,
    rprSpanErrorStatsContainedFrames Counter64,
    rprSpanErrorStatsScffErrors      Counter64
}

rprSpanErrorStatsIfIndex OBJECT-TYPE
    SYNTAX          InterfaceIndex

```

```

MAX-ACCESS    not-accessible
STATUS        current
DESCRIPTION
    "The ifIndex of this RPR interface."
::= { rprSpanErrorCountersStatsEntry 1 }

rprSpanErrorStatsSpan OBJECT-TYPE
SYNTAX        RprSpan
MAX-ACCESS    not-accessible
STATUS        current
DESCRIPTION
    "An indication of the span of the interface for which this
row contains information."
REFERENCE
    "IEEE 802.17 Subclause 7.2.2, myRi"
::= { rprSpanErrorCountersStatsEntry 2 }

rprSpanErrorStatsTtlExpFrames OBJECT-TYPE
SYNTAX        Counter64
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of received (PHY to MAC) frames that were dropped
due to zero Time To Live (TTL)."
REFERENCE
    "IEEE 802.17 Subclause 7.6.3.6.2, ttlExpiredFrames"
::= { rprSpanErrorCountersStatsEntry 3 }

rprSpanErrorStatsTooLongFrames OBJECT-TYPE
SYNTAX        Counter64
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of received (PHY to MAC) frames that exceed the
maximum permitted frame size."
REFERENCE
    "IEEE 802.17 Subclause 7.6.3.6.2, tooLongFrames"
::= { rprSpanErrorCountersStatsEntry 4 }

rprSpanErrorStatsTooShortFrames OBJECT-TYPE
SYNTAX        Counter64
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of received (PHY to MAC) frames shortest than the
minimum permitted frame size."
REFERENCE
    "IEEE 802.17 Subclause 7.6.3.6.2, tooShortFrames"
::= { rprSpanErrorCountersStatsEntry 5 }

rprSpanErrorStatsBadHecFrames OBJECT-TYPE
SYNTAX        Counter64
MAX-ACCESS    read-only
STATUS        current
DESCRIPTION
    "The number of received (PHY to MAC) frames with HEC error."
REFERENCE
    "IEEE 802.17 Subclause 7.6.3.6.2, badHecFrames"
::= { rprSpanErrorCountersStatsEntry 6 }

```

```
rprSpanErrorStatsBadFcsFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of received (PHY to MAC) data and control frames
         where the fcs value did not match the expected fcs value.

        This includes data frames passed to the client as a result of
         rprIfMacOperModes being not set to dropBadFcs."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.6.2, badFcsFrames"
        ::= { rprSpanErrorCountersStatsEntry 7 }

rprSpanErrorStatsSelfSrcUcastFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of received (PHY to MAC) unicast frames that were
         transmitted by the station itself.
        I.e., the source MAC is equal to the interface MAC."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.8.2, selfSourcedFrames"
        ::= { rprSpanErrorCountersStatsEntry 8 }

rprSpanErrorStatsPmdAbortFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of received (PHY to MAC) frames that
         were aborted by the PMD."
    REFERENCE
        "IEEE 802.17 Subclause 8.2.2, STATUS"
        ::= { rprSpanErrorCountersStatsEntry 9 }

rprSpanErrorStatsBadAddrFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of received (PHY to MAC) frames
         with invalid SA value."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.6.2, badAddressFrames"
        ::= { rprSpanErrorCountersStatsEntry 10 }

rprSpanErrorStatsBadParityFrames OBJECT-TYPE
    SYNTAX          Counter64
    MAX-ACCESS     read-only
    STATUS         current
    DESCRIPTION
        "The number of received (PHY to MAC) frames
         parity value not matching the expected parity value."
    REFERENCE
        "IEEE 802.17 Subclause 7.6.3.6.2, badParityFrames"
        ::= { rprSpanErrorCountersStatsEntry 11 }
```

```

rprSpanErrorStatsContainedFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) frames
         that were removed due to context containment."
    REFERENCE
        "IEEE 802.17 Subclause , containedFrames"
    ::= { rprSpanErrorCountersStatsEntry 12 }

rprSpanErrorStatsScffErrors OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of received (PHY to MAC) errored SCFF,
         with bad parity, bad FCS, or both."
    REFERENCE
        "IEEE 802.17 Subclause 12.5.1.2, scffErrors"
    ::= { rprSpanErrorCountersStatsEntry 13 }

-- 
-- SAS tables
--

-- 
-- SDB table information.
-- 

rprSasSummaryTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprSasSummaryEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A table that reports information about SDB
         tables for each RPR interface running SAS."
    ::= { rprSasObjects 1 }

rprSasSummaryEntry OBJECT-TYPE
    SYNTAX      RprSasSummaryEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "Information about a specific SDB.

        The rprSasSummarySdbPurgeCmd, rprSasSummarySdbPurgeScopeSingle
        and rprSasSummarySdbPurgeScopeMap read-write attributes in
        this table are volatile, i.e., will be cleared on system reset.

        All other read-write attributes in this table are non-volatile,
        i.e., will be retained across system reset."
    INDEX      { rprSasSummaryIfIndex }
    ::= { rprSasSummaryTable 1 }

RprSasSummaryEntry ::= SEQUENCE {
    rprSasSummaryIfIndex                               InterfaceIndex,

```

```

rprSasSummaryDynamicCount          Unsigned32,
rprSasSummaryStaticUcastCount     Unsigned32,
rprSasSummaryStaticMcastCount     Unsigned32,
rprSasSummaryAgingTime           Unsigned32,
rprSasSummaryStdCleave           TruthValue,
rprSasSummarySdbPurgeCmd         INTEGER,
rprSasSummarySdbPurgeScopeSingle Unsigned32,
rprSasSummarySdbPurgeScopeMap    OCTET STRING
}

rprSasSummaryIfIndex OBJECT-TYPE
SYNTAX      InterfaceIndex
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"The ifIndex of the RPR interface running SAS."
::= { rprSasSummaryEntry 1 }

rprSasSummaryDynamicCount OBJECT-TYPE
SYNTAX      Unsigned32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The current number of dynamic entries in this interfaces SDB."
REFERENCE
"IEEE 802.17 Subclause 14.2.5.1, numDynamicEntries"
::= { rprSasSummaryEntry 2 }

rprSasSummaryStaticUcastCount OBJECT-TYPE
SYNTAX      Unsigned32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The current number of static unicast entries in this
interfaces SDB."
REFERENCE
"IEEE 802.17 Subclause 14.2.5.1, NumStaticUcastEntries"
::= { rprSasSummaryEntry 3 }

rprSasSummaryStaticMcastCount OBJECT-TYPE
SYNTAX      Unsigned32
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The current number of static multicast entries in this
interfaces SDB."
REFERENCE
"IEEE 802.17 Subclause 14.2.5.1, NumStaticMcastEntries."
::= { rprSasSummaryEntry 4 }

rprSasSummaryAgingTime OBJECT-TYPE
SYNTAX      Unsigned32 (10..1000000)
UNITS      "seconds"
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION
"The aging timer used for dynamic entries in the SDB.
When an entry is created or updated its age is reset to 0.
Entries are removed when their age reaches or exceeds the value

```

specified by this attribute."

REFERENCE

"IEEE 802.17 Subclause 14.2.5.1, sbdAgingTimer"

DEFVAL { 300 }

::= { rprSasSummaryEntry 5 }

rprSasSummaryStdCleave OBJECT-TYPE

SYNTAX TruthValue

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"This attribute reports if this RPR station uses the standard cleave point algorithm when flooding SAS traffic onto the ring."

REFERENCE

"IEEE 802.17 Subclause 6.2.1, stdCleave"

::= { rprSasSummaryEntry 6 }

rprSasSummarySdbPurgeCmd OBJECT-TYPE

SYNTAX INTEGER {
 idle(1),
 purgeAll(2),
 purgeVid(3),
 purgeMsti(4),
 purgeVidMap(5),
 purgeMstiMap(6)
 }

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"This attribute is used to perform a full or partial purge of the SAS dynamic (not static) database for this RPR interface. This attribute has no effect on the configured static entries.

If the attribute is set to 'purgeAll' then all entries in the SAS database are purged.

If the attribute is set to 'purgeVid' then the value of rprSasSummarySdbPurgeScopeSingle specifies the VID to be purged according to the semantics of VlanIdOrNone.

If the attribute is set to 'purgeMsti' then the value of rprSasSummarySdbPurgeScopeSingle specifies the MSTI to be purged.

If the attribute is set to 'purgeVidMap' then the value of rprSasSummarySdbPurgeScopeMap specifies the set of VIDs to be purged.

If the attribute is set to 'purgeMstiMap' then the value of rprSasSummarySdbPurgeScopeMap specifies the MSTIs to be purged."

REFERENCE

"IEEE 802.17 Subclause 14.2.2, purgeScope"

DEFVAL { idle }

::= { rprSasSummaryEntry 7 }

rprSasSummarySdbPurgeScopeSingle OBJECT-TYPE

SYNTAX Unsigned32(0..4095)

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"This attribute is used to specify a VID or MSTI value used

to select entries to be purged when rprSasSummarySdbPurgeCmd is set to 'purgeVid' or 'purgeMsti'.

This attribute is ignored if rprSasSummarySdbPurgeCmd is set to 'purgeAll', 'purgeVidMap' or 'purgeMstiMap'.

Provides the VID value to be used when rprSasSummaryPurgeCmd is set to 'purgeVid'.

Provides the MSTI value to be used when rprSasSummaryPurgeCmd is set to 'purgeMsti'."

REFERENCE

"IEEE 802.17 Subclause 14.2.2, purgeValue"

DEFVAL { 0 }

::= { rprSasSummaryEntry 8 }

rprSasSummarySdbPurgeScopeMap OBJECT-TYPE
SYNTAX OCTET STRING(SIZE(0..512))
MAX-ACCESS read-write
STATUS current
DESCRIPTION

"This attribute is used to specify a VID or MSTI map used to select entries to be purged when rprSasSummarySdbPurgeCmd is set to 'purgeVidMap' or 'purgeMstiMap'.

This object is a octet string of length from 0 to 512 bytes. Each bit in the octet string represents a single VID or MSTI value from value 0 to value 4095. The first bit of the first octet corresponds to VID/MSTI value 0. The last bit of the first octet corresponds to VID/MSTI value 7. The second octet represents values 8 though 15. The 512th octet represents values 4088 though 4095.

Note that is the length of this octet string is less than the maximum length, all omitted octets are presumed to contain the value zero. This applies to both set and get operations. A string of length zero string is an empty map.

This attribute is ignored if rprSasSummarySdbPurgeCmd is set to 'purgeAll', 'purgeVid' or 'purgeMsti'.

Provides the map of VID values to be used when rprSasSummaryPurgeCmd is set to 'purgeVidMap'.

Provides the map of MSTI values to be used when rprSasSummaryPurgeCmd is set to 'purgeMstiMap'."

REFERENCE

"IEEE 802.17 Subclause 14.2.2, purgeValue"

DEFVAL { "" }

::= { rprSasSummaryEntry 9 }

--
-- The unicast MAC address table
--

rprSasDbTable OBJECT-TYPE
SYNTAX SEQUENCE OF RprSasDbEntry
MAX-ACCESS not-accessible
STATUS current

DESCRIPTION

"A table that contains unicast address association entries, including both dynamic (i.e., learned) and static entries. Except for VID-only static entries, these entries associate a MAC address (provided in the MA_DATA.request destination_address parameter) and SAS Database ID value with an RPR station address. A static VID-only entry ignores the MAC address."

REFERENCE

"IEEE 802.17 Subclause 14.2.5.2"
 $\text{ ::= } \{ \text{rprSasObjects} \text{ 2 } \}$

rprSasDbEntry OBJECT-TYPE

SYNTAX RprSasDbEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A SDB unicast association entry."

INDEX { rprSasDbIfIndex,
 rprSasDbId,
 rprSasDbAddr }

$\text{ ::= } \{ \text{rprSasDbTable} \text{ 1 } \}$

RprSasDbEntry ::=

SEQUENCE {	
rprSasDbIfIndex	InterfaceIndex,
rprSasDbId	RprSasDbIdValue,
rprSasDbAddr	MacAddress,
rprSasDbTargetAddr	MacAddress,
rprSasDbEntryStatus	INTEGER

}

rprSasDbIfIndex OBJECT-TYPE

SYNTAX InterfaceIndex

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The ifIndex of the RPR interface containing this SDB."

REFERENCE

"RFC 2863, ifIndex"
 $\text{ ::= } \{ \text{rprSasDbEntry} \text{ 1 } \}$

rprSasDbId OBJECT-TYPE

SYNTAX RprSasDbIdValue

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The identity of this SDB.

If this MAC interface has a bridging client, this is the same as the dot1qFdbId value used in the corresponding bridge relay FDB entry.

If the MAC interface does not have a bridging client, then the value used is 1.

The value of 1 is used when SAS is VLAN-unaware."

REFERENCE

"IEEE 802.17 Subclause 14.2.5.2, sdbEntryFid"
 $\text{ ::= } \{ \text{rprSasDbEntry} \text{ 2 } \}$

rprSasDbAddr OBJECT-TYPE

SYNTAX MacAddress

```

MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "The unicast destination MAC address for this entry.
     Except for VID-only static entries, this is compared to the
     MA_DATA.request destination_address parameter.

    For a VID-only static entry, this attribute is set to the
    broadcast address FF-FF-FF-FF-FF-FF and ignored in the table
    search."
REFERENCE
    "IEEE 802.17 Subclause 14.2.5.2, sdbEntryDestAddr"
::= { rprSasDbEntry 3 }

rprSasDbTargetAddr OBJECT-TYPE
SYNTAX      MacAddress
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The target address (i.e., local RPR station address) associated
     with this entry."
REFERENCE
    "IEEE 802.17 Subclause 14.2.5.2, sdbEntryTargetAddr"
::= { rprSasDbEntry 4 }

rprSasDbEntryStatus OBJECT-TYPE
SYNTAX      INTEGER {
            other(1),
            invalid(2),
            learned(3),
            self(4),
            mgmt(5)
        }
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The status of this entry. The meanings of the values
     are:
        other(1) - none of the following. This includes
                    the case where some other MIB object (not the
                    corresponding instance of rprSasDbTargetAddr, nor an
                    entry in the rprSasStaticUcastTable) created the entry.
        invalid(2) - this entry is no longer valid (e.g., has
                    aged out), but has not yet been flushed from the table.
        learned(3) - the value of the corresponding instance
                    of rprSasDbTargetAddr was learned and is being used.
        self(4) - represents the local station address.
        mgmt(5) - created using an entry in the rprSasStaticAddress
                    table."
REFERENCE
    "IEEE 802.17 Subclause 14.2.5.2, sdbEntryType"
::= { rprSasDbEntry 5 }

-- 
-- The static unicast MAC address table
--

rprSasStaticUcastTable OBJECT-TYPE

```

```

SYNTAX      SEQUENCE OF RprSasStaticUcastEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "A table containing static association entries for
     unicast MAC addresses configured by management.
     These entries associate a MAC address provided in a
     MA_DATA.request destination_address parameter
     with an RPR station address.
     This table also supports VID only association entries
     which direct all traffic with a given VID (regardless
     of the destination_address) to the specified RPR
     station address.

    All read-write attributes in this table are non-volatile,
    i.e., will be retained across system reset."
REFERENCE
    "IEEE 802.17 Subclause 14.2.5.4"
::= { rprSasObjects 3 }

rprSasStaticUcastEntry OBJECT-TYPE
SYNTAX      RprSasStaticUcastEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "A statically configured SDB unicast association entry."
INDEX      {
    rprSasStaticUcastIfIndex,
    rprSasStaticUcastVid,
    rprSasStaticUcastAddr
}
 ::= { rprSasStaticUcastTable 1 }

RprSasStaticUcastEntry ::=
SEQUENCE {
    rprSasStaticUcastIfIndex          InterfaceIndex,
    rprSasStaticUcastVid             VlanIdOrNone,
    rprSasStaticUcastAddr            MacAddress,
    rprSasStaticUcastVidOnly        TruthValue,
    rprSasStaticUcastTargetAddr     MacAddress,
    rprSasStaticUcastRowStatus      RowStatus
}

rprSasStaticUcastIfIndex OBJECT-TYPE
SYNTAX      InterfaceIndex
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "The ifIndex of the RPR interface containing this SDB."
REFERENCE
    "RFC 2863, ifIndex"
::= { rprSasStaticUcastEntry 1 }

rprSasStaticUcastVid OBJECT-TYPE
SYNTAX      VlanIdOrNone
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "The VLAN ID for this static entry.

```

The value of 0 is used when SAS is VLAN-unaware or for untagged or priority tagged frames."

REFERENCE

"IEEE 802.17 Subclause 14.2.5.4, sdbStaticUcastVid"
 ::= { rprSasStaticUcastEntry 2 }

rprSasStaticUcastAddr OBJECT-TYPE

SYNTAX MacAddress
MAX-ACCESS not-accessible
STATUS current

DESCRIPTION

"The unicast MAC address for this entry.
This is compared to the MA_DATA.request destination address parameter when rprSasStaticUcastVidOnly is set to 'false'.
If rprSasStaticUcastVidOnly is set to true, this attribute is not used in the SAS lookup, and must be set to the broadcast address FF-FF-FF-FF-FF-FF."

REFERENCE

"IEEE 802.17 Subclause 14.2.5.4, sdbStaticUcastDestAddr"
 ::= { rprSasStaticUcastEntry 3 }

rprSasStaticUcastVidOnly OBJECT-TYPE

SYNTAX TruthValue
MAX-ACCESS read-create
STATUS current

DESCRIPTION

"This attribute indicates if this entry matches on VID only, ignoring the value of MA_DATA.request destination address parameter (including not checking for unicast vs. multicast). Setting this attribute to 'true' means all frames being transmitted with the matching VID value (regardless of the destination address) will match this entry and be sent to the station specified in rprSasStaticUcastTargetAddr."

REFERENCE

"IEEE 802.17 Subclause 14.2.5.4, sdbStaticUcastType"
DEFVAL { false }
 ::= { rprSasStaticUcastEntry 4 }

rprSasStaticUcastTargetAddr OBJECT-TYPE

SYNTAX MacAddress
MAX-ACCESS read-create
STATUS current

DESCRIPTION

"The local RPR station address associated with this entry."

REFERENCE

"IEEE 802.17 Subclause 14.2.5.4, sdbStaticUcastTargetAddr"
 ::= { rprSasStaticUcastEntry 5 }

rprSasStaticUcastRowStatus OBJECT-TYPE

SYNTAX RowStatus
MAX-ACCESS read-create
STATUS current

DESCRIPTION

"This attribute is used to control (e.g., create, delete, etc.) entries from this table as described in SNMPv2-TC.

None of the other writable objects in a row can be changed if the status is active(1).

All read-create objects must have valid and consistent values before the row can be activated."

REFERENCE

"RFC 2579, RowStatus and IEEE 802.17 Subclause 14.2.5.4,
sdbStaticUcastState"

::= { rprSasStaticUcastEntry 6 }

--
-- The static multicast MAC address table
--

rprSasStaticMcastTable OBJECT-TYPE

SYNTAX SEQUENCE OF RprSasStaticMcastEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A table containing static association entries for multicast MAC addresses configured by management. These entries associate a destination MAC address (and optionally VLAN ID) with east and west hop counts.

All read-write attributes in this table are non-volatile, i.e., will be retained across system reset."

REFERENCE "IEEE 802.17 Subclause 14.2.5.4."

::= { rprSasObjects 4 }

rprSasStaticMcastEntry OBJECT-TYPE

SYNTAX RprSasStaticMcastEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A statically configured entry for a multicast MAC address."

INDEX {
 rprSasStaticMcastIfIndex,
 rprSasStaticMcastVid,
 rprSasStaticMcastAddr
}

::= { rprSasStaticMcastTable 1 }

RprSasStaticMcastEntry ::=

SEQUENCE {

rprSasStaticMcastIfIndex	InterfaceIndex,
rprSasStaticMcastVid	VlanIdOrNone,
rprSasStaticMcastAddr	MacAddress,
rprSasStaticMcastHopsRinglet0	RprRingHopCount,
rprSasStaticMcastHopsRinglet1	RprRingHopCount,
rprSasStaticMcastExtend	TruthValue,
rprSasStaticMcastRowStatus	RowStatus

}

rprSasStaticMcastIfIndex OBJECT-TYPE

SYNTAX InterfaceIndex

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The ifIndex of the RPR interface containing this SDB."

REFERENCE

"RFC 2863, ifIndex"

```

 ::= { rprSasStaticMcastEntry 1 }

rprSasStaticMcastVid OBJECT-TYPE
    SYNTAX      VlanIdOrNone
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The VLAN ID for this static entry.
        The value of 0 is used when SAS is VLAN-unaware or for
        untagged or priority tagged frames."
    REFERENCE
        "IEEE 802.17 Subclause 14.2.5.5, sdbStaticMcastVid"
    ::= { rprSasStaticMcastEntry 2 }

rprSasStaticMcastAddr OBJECT-TYPE
    SYNTAX      MacAddress
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The multicast MAC address for this entry.
        This is compared to the MA_DATA.request destination_address
        parameter."
    REFERENCE
        "IEEE 802.17 Subclause 14.2.5.5, sdbStaticMcastAddr"
    ::= { rprSasStaticMcastEntry 3 }

rprSasStaticMcastHopsRinglet0 OBJECT-TYPE
    SYNTAX      RprRingHopCount
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The number of hops to send frames matching this entry
        on ringlet0.
        A value of '0' means that the frames are not sent on ringlet0."
    REFERENCE
        "IEEE 802.17 Subclause 14.2.5.5, sdbStaticMcastHopsRinglet0"
    DEFVAL { 0 }
    ::= { rprSasStaticMcastEntry 4 }

rprSasStaticMcastHopsRinglet1 OBJECT-TYPE
    SYNTAX      RprRingHopCount
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The number of hops to send frames matching this entry
        on ringlet1.
        A value of '0' means that the frames are not sent on ringlet1."
    REFERENCE
        "IEEE 802.17 Subclause 14.2.5.5, sdbStaticMcastHopsRinglet1"
    DEFVAL { 0 }
    ::= { rprSasStaticMcastEntry 5 }

rprSasStaticMcastExtend OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This attribute is used to control if the frames that use this
        entry always extended and sent with the SAS group address.

```

Frames extended like this are used by the SAS association learning entity on station that receive the frame.
A value of 'true' means that the frames are always sent extended with the SAS group address as the outer destination address."

REFERENCE

"IEEE 802.17 Subclause 14.2.5.5, sdbStaticMcastExtend"

DEFVAL { true }

::= { rprSasStaticMcastEntry 6 }

rprSasStaticMcastRowStatus OBJECT-TYPE

SYNTAX RowStatus

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"This attribute is used to control (e.g., create, delete, etc.) entries from this table as described in SNMPv2-TC.

None of the other writable objects in a row can be changed if the status is active(1).

All read-create objects must have valid and consistent values before the row can be activated."

REFERENCE

"RFC 2579, RowStatus and IEEE 802.17 Subclause 14.2.5.5, sdbStaticMcastState."

::= { rprSasStaticMcastEntry 7 }

--

-- Sas Counters.

--

rprSasCountersStatsTable OBJECT-TYPE

SYNTAX SEQUENCE OF RprSasCountersStatsEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The SAS interface total counters table.

The DiscontinuityTime for this table is indicated by ifCounterDiscontinuityTime defined in ifXTable."

::= { rprSasCounters 1 }

rprSasCountersStatsEntry OBJECT-TYPE

SYNTAX RprSasCountersStatsEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"Statistics for an RPR interface with SAS enabled."

INDEX { rprSasCountersStatsIfIndex }

::= { rprSasCountersStatsTable 1 }

RprSasCountersStatsEntry ::= SEQUENCE {

rprSasCountersStatsIfIndex	InterfaceIndex,
rprSasCountersStatsLocalPurges	Counter64,
rprSasCountersStatsTopologyPurges	Counter64,
rprSasCountersStatsNotifyPurges	Counter64,
rprSasCountersStatsLearnFulls	Counter64,
rprSasCountersStatsTxDirectedFrames	Counter64,
rprSasCountersStatsTxUndirectedFrames	Counter64,

```

rprSasCountersStatsTxMcastScopedFrames      Counter64
}

rprSasCountersStatsIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The ifIndex of the RPR interface running SAS."
    REFERENCE
        "RFC 2863, ifIndex"
    ::= { rprSasCountersStatsEntry 1 }

rprSasCountersStatsLocalPurges OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of locally initiated SDB purges."
    REFERENCE
        "IEEE 802.17 Subclause 14.2.6, sdbLocalPurges"
    ::= { rprSasCountersStatsEntry 2 }

rprSasCountersStatsTopologyPurges OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of topology initiated SDB purges."
    REFERENCE
        "IEEE 802.17 Subclause 14.2.6, sdbTopologyPurges"
    ::= { rprSasCountersStatsEntry 3 }

rprSasCountersStatsNotifyPurges OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of OAM SAS Notify initiated SDB purges."
    REFERENCE
        "IEEE 802.17 Subclause 14.2.6, sdbNotifyPurges"
    ::= { rprSasCountersStatsEntry 4 }

rprSasCountersStatsLearnFulls OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The number of times SAS learning could not add a new entry
         due to SDB occupancy and either overwrote an existing entry
         or did not learn the new information."
    REFERENCE
        "IEEE 802.17 Subclause 14.2.6, sdbLearnFulls"
    ::= { rprSasCountersStatsEntry 5 }

rprSasCountersStatsTxDirectedFrames OBJECT-TYPE
    SYNTAX      Counter64
    MAX-ACCESS  read-only
    STATUS      current

```

```

DESCRIPTION
    "The number of directed frames sent by SAS on this interface."
REFERENCE
    "IEEE 802.17 Subclause 14.2.6, sasTxDirectedFrames"
::= { rprSasCountersStatsEntry 6 }

rprSasCountersStatsTxUndirectedFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of undirected frames sent by SAS on this interface."
REFERENCE
    "IEEE 802.17 Subclause 14.2.6, sasTxUndirectedFrames"
::= { rprSasCountersStatsEntry 7 }

rprSasCountersStatsTxMcastScopedFrames OBJECT-TYPE
SYNTAX      Counter64
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The number of multicast scoped frames sent by SAS on
     this interface."
REFERENCE
    "IEEE 802.17 Subclause 14.2.6, sasTxMcastScopedFrames"
::= { rprSasCountersStatsEntry 8 }

-- 
-- PIRC tables
--

-- 
-- PIRC Configuration table
--

rprPircConfTable OBJECT-TYPE
SYNTAX      SEQUENCE OF RprPircConfEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "A table for configuration of PIRC related attributes,
     for each RPR interface running PIRC."
::= { rprPircObjects 1 }

rprPircConfEntry OBJECT-TYPE
SYNTAX      RprPircConfEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
    "Attributes of MAC PIRC functionality.
     All the attributes in this table are non-volatile,
     i.e., will be retained across system reset."
INDEX      { rprPircConfIfIndex }
::= { rprPircConfTable 1 }

RprPircConfEntry ::= SEQUENCE {
    rprPircConfIfIndex           InterfaceIndex,
    rprPircConfPrimary          TruthValue,
}

```

```

        rprPircConfRevertive           TruthValue,
        rprPircConfLoadBalancing      TruthValue,
        rprPircConfFastMateMsgRpt    Unsigned32,
        rprPircConfFastMateMsgInterval Unsigned32,
        rprPircConfSlowMateMsgInterval Unsigned32,
        rprPircConfWtrTimeout        Unsigned32,
        rprPircConfPcTimeout         Unsigned32,
        rprPircConfMateKeepAliveInterval Unsigned32,
        rprPircConfProtectionCommand INTEGER
    }

rprPircConfIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  not-accessible
    STATUS     current
    DESCRIPTION
        "The ifIndex of the RPR interface running PIRC."
    ::= { rprPircConfEntry 1 }

rprPircConfPrimary OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-write
    STATUS     current
    DESCRIPTION
        "Indicates that this station is the primary station in
         the protection group."
    REFERENCE
        "IEEE 802.17 Subclause 15.3.1, pircProv.pircPrimary."
    DEFVAL { true }
    ::= { rprPircConfEntry 2 }

rprPircConfRevertive OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-write
    STATUS     current
    DESCRIPTION
        "Indicates that this station will return to PIRC noRequest
         state after WTR interval expires."
    REFERENCE
        "IEEE 802.17 Subclause 15.3.1, pircProv.pircRevertive."
    DEFVAL { false }
    ::= { rprPircConfEntry 3 }

rprPircConfLoadBalancing OBJECT-TYPE
    SYNTAX      TruthValue
    MAX-ACCESS  read-write
    STATUS     current
    DESCRIPTION
        "Indicates that this station operates in PIRC load-balancing
         mode."
    REFERENCE
        "IEEE 802.17 Subclause 15.3.1, pircProv.pircLoadBalancing."
    DEFVAL { false }
    ::= { rprPircConfEntry 4 }

rprPircConfFastMateMsgRpt OBJECT-TYPE
    SYNTAX      Unsigned32 (1..10)
    MAX-ACCESS  read-write
    STATUS     current

```

```

DESCRIPTION
    "Indicates the number of times a PIRC mate message shall
     be retransmitted after PIRC status change."
REFERENCE
    "IEEE 802.17 Subclause 15.3.1, pircProv.fastMateMsgRpt."
DEFVAL { 3 }
 ::= { rprPircConfEntry 5 }

rprPircConfFastMateMsgInterval OBJECT-TYPE
SYNTAX      Unsigned32 (1..1000)
UNITS       "milliseconds"
MAX-ACCESS   read-write
STATUS      current
DESCRIPTION
    "Indicates the time interval between PIRC mate message
     retransmission after PIRC status change."
REFERENCE
    "IEEE 802.17 Subclause 15.3.1, pircProv.fastMateMsgInterval."
DEFVAL { 10 }
 ::= { rprPircConfEntry 6 }

rprPircConfSlowMateMsgInterval OBJECT-TYPE
SYNTAX      Unsigned32 (1..1000)
UNITS       "10 milliseconds"
MAX-ACCESS   read-write
STATUS      current
DESCRIPTION
    "Indicates the time interval between periodic PIRC mate
     message."
REFERENCE
    "IEEE 802.17 Subclause 15.3.1, pircProv.slowMateMsgInterval."
DEFVAL { 10 }
 ::= { rprPircConfEntry 7 }

rprPircConfWtrTimeout OBJECT-TYPE
SYNTAX      Unsigned32 (0..100)
UNITS       "Seconds"
MAX-ACCESS   read-write
STATUS      current
DESCRIPTION
    "Indicates the PIRC WTR timeout interval."
REFERENCE
    "IEEE 802.17 Subclause 15.3.1, pircProv.wtrTimeout."
DEFVAL { 10 }
 ::= { rprPircConfEntry 8 }

rprPircConfPcTimeout OBJECT-TYPE
SYNTAX      Unsigned32 (0..100)
UNITS       "10 milliseconds"
MAX-ACCESS   read-write
STATUS      current
DESCRIPTION
    "Indicates the PIRC containment timeout interval."
REFERENCE
    "IEEE 802.17 Subclause 15.3.1, pircProv.pcTimeout."
DEFVAL { 5 }
 ::= { rprPircConfEntry 9 }

rprPircConfMateKeepAliveInterval OBJECT-TYPE

```

```
SYNTAX      Unsigned32 (3..10000)
UNITS       "10 milliseconds"
MAX-ACCESS   read-write
STATUS      current
DESCRIPTION
    "Indicates the time interval without reception of any PIRC
     mate message for determining loss of connectivity with
     PIRC mate."
REFERENCE
    "IEEE 802.17 Subclause 15.3.1, pircProv.mateKeepAliveInterval."
DEFVAL { 35 }
 ::= { rprPircConfEntry 10 }
```

rprPircConfProtectionCommand OBJECT-TYPE

```
SYNTAX      INTEGER {
    idle          (1),
    manualSwitch (2),
    forcedSwitch (3)
}
```

```
MAX-ACCESS   read-write
STATUS      current
```

DESCRIPTION

"The PIRC protection mode requested by management for the local PIRC station, according to the set of rules describing the PIRC protection.

When read, this object returns the last PIRC command written unless it has been preempted, or idle if no PIRC command has been written to this station since initialization.

There is no pending of commands, that is if a PIRC command has been preempted by a PIRC failure, when the PIRC failure clears the PIRC command is not executed.

If the PIRC command cannot be executed because an equal or higher priority PIRC request is in effect, an error is returned.

Writing idle to a station that has no pending PIRC protection command, has no affect. An idle clears an active PIRC WTR state.

The protection commands (arranged in ascending priority order) are:

idle

This command clears the PIRC protection for the station. This value should be returned by a read request when no PIRC protection command has been written to the object.

manualSwitch

A PIRC protection command for the station. This PIRC command does not have precedence over automatic PIRC protection, and therefore it cannot preempt an existing automatic PIRC protection request.

forcedSwitch

A PIRC command for the station. This PIRC command has precedence over automatic PIRC protection, and therefore it can preempt an existing automatic PIRC protection request.

```

        Default value of PIRC protection command is idle."
REFERENCE
        "IEEE 802.17 Subclause 15.2.2, adminPircReq."
DEFVAL { idle }
 ::= { rprPircConfEntry 11 }

-- 
-- PIRC status summary table
--

rprPircSummaryTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF RprPircSummaryEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A table that reports information about PIRC status and variables
         for each RPR interface running PIRC."
 ::= { rprPircObjects 2 }

rprPircSummaryEntry OBJECT-TYPE
    SYNTAX      RprPircSummaryEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "PIRC functionality information about a specific PIRC station."
INDEX   { rprPircSummaryIfIndex }
 ::= { rprPircSummaryTable 1 }

RprPircSummaryEntry ::= SEQUENCE {
    rprPircSummaryIfIndex           InterfaceIndex,
    rprPircSummaryProtState        RprPircProtectionStatus,
    rprPircSummaryMateLastMsgTime  TimeStamp,
    rprPircSummaryMateMacAddress   MacAddress,
    rprPircSummaryMatePrimary      TruthValue,
    rprPircSummaryMateLoadbalancing TruthValue,
    rprPircSummaryMateOK          TruthValue,
    rprPircSummaryMateManualSwitch TruthValue
}

rprPircSummaryIfIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The ifIndex of the RPR interface running PIRC."
 ::= { rprPircSummaryEntry 1 }

rprPircSummaryProtState OBJECT-TYPE
    SYNTAX      RprPircProtectionStatus
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The current PIRC protection status of the station."
REFERENCE
        "IEEE 802.17 Subclause 15.3.2, pircInfo.ProtState."
 ::= { rprPircSummaryEntry 2 }

rprPircSummaryMateLastMsgTime OBJECT-TYPE

```

```
SYNTAX     TimeStamp
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The value of sysUpTime at the time when the last valid
     PIRC mate message was received."
REFERENCE
    "IEEE 802.17 Subclause 15.3.2, pircMateMsg.lastTimeStamp."
::= { rprPircSummaryEntry 3 }

rprPircSummaryMateMacAddress OBJECT-TYPE
SYNTAX      MacAddress
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The source MAC Address of last valid PIRC mate message
     that was received."
REFERENCE
    "IEEE 802.17 Subclause 15.3.2, pircMateMsg.macAddress."
::= { rprPircSummaryEntry 4 }

rprPircSummaryMatePrimary OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The pircPrimary of last valid PIRC mate message
     that was received."
REFERENCE
    "IEEE 802.17 Subclause 15.3.3, pircMateMsg.pr."
::= { rprPircSummaryEntry 5 }

rprPircSummaryMateLoadbalancing OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The pircLoadBalancing of last valid PIRC mate message
     that was received."
REFERENCE
    "IEEE 802.17 Subclause 15.3.3, pircMateMsg.lb."
::= { rprPircSummaryEntry 6 }

rprPircSummaryMateOK OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
    "The OK indication of last valid PIRC mate message
     that was received."
REFERENCE
    "IEEE 802.17 Subclause 15.3.3, pircMateMsg.ok."
::= { rprPircSummaryEntry 7 }

rprPircSummaryMateManualSwitch OBJECT-TYPE
SYNTAX      TruthValue
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
```

```

    "The manual-switch indication of last valid PIRC
     mate message that was received."
REFERENCE
    "IEEE 802.17 Subclause 15.3.3, pircMateMsg.ms."
::= { rprPircSummaryEntry 8 }

-- 
-- conformance information
--

rprGroups      OBJECT IDENTIFIER ::= { rprConformance 1 }
rprCompliances OBJECT IDENTIFIER ::= { rprConformance 2 }

rprModuleTotalStatsCompliance MODULE-COMPLIANCE
  STATUS current
  DESCRIPTION
    "The compliance statement for agent that support RPR operation
     with total statistics collections."

  MODULE -- this module
    MANDATORY-GROUPS { rprIfGroup,
                        rprSpanGroup,
                        rprSpanProtectionGroup,
                        rprTopoGroup,
                        rprFairnessGroup,
                        rprSpanStatsGroup,
                        rprClientStatsGroup,
                        rprErrorStatsGroup }

    GROUP rprIfGroupOpt
    DESCRIPTION
      "Collection of objects for RPR MAC.
       This group is optional."

    GROUP rprSpanGroupOpt
    DESCRIPTION
      "Collection of objects for RPR span.
       This group is optional."

    GROUP rprIfStatsControlGroup
    DESCRIPTION
      "Current, interval, total intervals and running counters management.
       This group is optional."

    GROUP rprFairnessGroupOpt
    DESCRIPTION
      "Collection of objects for RPR fairness.
       This group is optional."

    GROUP rprOamGroup
    DESCRIPTION
      "OAM Echo and Flush control and status table.
       This group is optional."

    GROUP rprIfChangeSummaryGroup
    DESCRIPTION
      "RPR interfaces changes summary group.

```

This group is optional."

GROUP rprSpanCurrentGroup
DESCRIPTION
"Collection of RPR MAC Span current interval counters.
This group is optional."

GROUP rprSpanIntervalGroup
DESCRIPTION
"Collection of RPR MAC Span counters during specific 15 min
interval. This group is optional."

GROUP rprSpanDayGroup
DESCRIPTION
"Collection of RPR MAC Span Day counters, contains the
cumulative sum of the span statistics for the 24 hour period
preceding the current interval.

This group is optional, the Span Day statistics can be
calculated from the 96 15-min Intervals table."

GROUP rprClientCurrentGroup
DESCRIPTION
"Collection of RPR MAC client interface current interval counters.
This group is optional."

GROUP rprClientIntervalGroup
DESCRIPTION
"Collection of RPR MAC client interface counters during
specific 15 min interval. This group is optional."

GROUP rprClientDayGroup
DESCRIPTION
"Collection of RPR MAC Client Day counters, contains the
cumulative sum of the client interface statistics for the 24
hour period preceding the current interval.

This group is optional, the client Day statistics can be
calculated from the 96 15-min Intervals table."

GROUP rprErrorCurrentGroup
DESCRIPTION
"Collection of RPR MAC span error current interval counters.
This group is optional."

GROUP rprErrorIntervalGroup
DESCRIPTION
"Collection of RPR MAC span error counters during
specific 15 min interval. This group is optional."

GROUP rprErrorDayGroup
DESCRIPTION
"Collection of RPR MAC Error Day counters, contains the
cumulative sum of the span error statistics for the 24
hour period preceding the current interval.

This group is optional, the error Day statistics can be
calculated from the 96 15-min Intervals table."

```

OBJECT rprIfMacOperModes
SYNTAX     BITS { strictOrder(0), dropBadFcs(1) }
DESCRIPTION
    "Only support for the original bit values is required."

OBJECT rprTopoImageStatus
SYNTAX     BITS {
    reachableRinglet0(0),
    reachableRinglet1(1),
    wrapActiveWest(2),
    wrapActiveEast(3),
    receivedBadFcs(4),
    receivedMultichokeFairness(5) }
DESCRIPTION
    "Only support for the original bit values is required.

::= { rprCompliances 1 }

rprModuleSasCompliance MODULE-COMPLIANCE
STATUS current
DESCRIPTION
    "The compliance statement for agents that support RPR
    with the optional Spatially Aware Sublayer as well as the
    requirements from rprModuleTotalStatsCompliance."

MODULE -- this module
MANDATORY-GROUPS {
    rprIfGroup,          -- From rprModuleTotalStatsCompliance
    rprSpanGroup,
    rprSpanProtectionGroup,
    rprTopoGroup,
    rprFairnessGroup,
    rprSpanStatsGroup,
    rprClientStatsGroup,
    rprErrorStatsGroup,
    rprSasSummaryGroup, -- New groups
    rprSasGroup,
    rprSasStatsGroup,
    rprSasSpanStatsGroup
}

GROUP rprSasSummaryPurgeGroup
DESCRIPTION
    "Attributes for SAS database purge control.
    This group is optional."

GROUP rprSasStaticUcastGroup
DESCRIPTION
    "Attributes for SAS Unicast Static Entries.
    This group is optional."

GROUP rprSasStaticMcastGroup
DESCRIPTION
    "Attributes for SAS Multicast Static Entries.
    This group is optional."

OBJECT rprSasStaticUcastRowStatus
SYNTAX RowStatus { active(1), notInService(2) }
WRITE-SYNTAX RowStatus { notInService(2), createAndGo(4), destroy(6) }

```

```

DESCRIPTION
    "Support for createAndWait and notReady values are
     not required."

OBJECT rprSasStaticMcastRowStatus
SYNTAX RowStatus { active(1), notInService(2) }
WRITE-SYNTAX RowStatus { notInService(2), createAndGo(4), destroy(6) }
DESCRIPTION
    "Support for createAndWait and notReady values are
     not required."

OBJECT rprSasSummarySdbPurgeCmd
SYNTAX INTEGER { idle(1), purgeAll(2) }
WRITE-SYNTAX INTEGER { idle(1), purgeAll(2) }
DESCRIPTION
    "Support for purgeVid, purgeMsti, purgeVidMap and
     purgeMstiMap values are not required.

 ::= { rprCompliances 2 }

rprModulePircSasCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
        "The compliance statement for agents that support RPR
         with the optional Protected Ring Interconnect Sublayer
         as well as the requirements from rprModuleSasCompliance."
    MODULE -- this module
        MANDATORY-GROUPS {
            rprIfGroup,           -- From rprModuleTotalStatsCompliance
            rprSpanGroup,
            rprSpanProtectionGroup,
            rprTopoGroup,
            rprFairnessGroup,
            rprSpanStatsGroup,
            rprClientStatsGroup,
            rprErrorStatsGroup,
            rprSasSummaryGroup, -- From rprModuleSasCompliance
            rprSasGroup,
            rprSasStatsGroup,
            rprSasSpanStatsGroup,
            rprPircConfGroup, -- new groups
            rprPircSummaryGroup,
            rprPircSpanStatsGroup
        }
    ::= { rprCompliances 3 }

rprModulePircCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
        "The compliance statement for agents that support RPR
         with the optional Protected Ring Interconnect Sublayer
         as well as the requirements from rprModuleTotalStatsCompliance."
    MODULE -- this module
        MANDATORY-GROUPS {
            rprIfGroup,           -- From rprModuleTotalStatsCompliance
            rprSpanGroup,
            rprSpanProtectionGroup,

```

```

        rprTopoGroup,
        rprFairnessGroup,
        rprSpanStatsGroup,
        rprClientStatsGroup,
        rprErrorStatsGroup,
        rprPircConfGroup, -- new groups
        rprPircSummaryGroup,
        rprPircSpanStatsGroup
    }
::= { rprCompliances 4 }

--  

-- Units of conformance.  

--  

rprIfGroup OBJECT-GROUP
OBJECTS {
    rprIfStationsOnRing,
    rprIfReversionMode,
    rprIfProtectionWTR,
    rprIfProtectionFastTimer,
    rprIfProtectionSlowTimer,
    rprIfAtdTimer,
    rprIfKeepaliveTimeout,
    rprIfFairnessAggressive,
    rprIfPtqSize,
    rprIfStqSize,
    rprIfSTQFullThreshold,
    rprIfIdleThreshold,
    rprIfSesThreshold,
    rprIfWrapConfig,
    rprIfJumboFramePreferred,
    rprIfMacOperModes,
    rprIfRingOperModes,
    rprIfCurrentStatus
}
STATUS current
DESCRIPTION
    "Collection of objects needed for RPR MAC
     configuration."
::= { rprGroups 1 }

rprIfGroupOpt OBJECT-GROUP
OBJECTS {
    rprIfLastChange,
    rprIfChanges
}
STATUS current
DESCRIPTION
    "Collection of objects for RPR MAC."
::= { rprGroups 2 }

rprIfStatsControlGroup OBJECT-GROUP
OBJECTS {
    rprIfStatsControlPeriodClear,
    rprIfStatsControlCountPointClear,
    rprIfStatsControlIntervalClear,
    rprIfStatsControlCommitClear,
}

```

```
rprIfStatsControlTimeElapsed,
rprIfStatsControlValidIntervals
}
STATUS current
DESCRIPTION
"Collection of objects needed for RPR MAC
statistics management."
 ::= { rprGroups 3 }

rprSpanGroup OBJECT-GROUP
OBJECTS {
    rprSpanLowerLayerIfIndex,
    rprSpanTotalRingletReservedRate,
    rprSpanCurrentStatus
}
STATUS current
DESCRIPTION
"Collection of objects needed for RPR Span
configuration."
 ::= { rprGroups 4 }

rprSpanGroupOpt OBJECT-GROUP
OBJECTS {
    rprSpanLastChange,
    rprSpanChanges
}
STATUS current
DESCRIPTION
"Collection of objects for RPR Span."
 ::= { rprGroups 5 }

rprSpanProtectionGroup OBJECT-GROUP
OBJECTS {
    rprSpanProtectionNeighborValid,
    rprSpanProtectionHoldOffTimer,
    rprSpanProtectionCommand,
    rprSpanProtectionCount,
    rprSpanProtectionDuration,
    rprSpanProtectionLastActivationTime
}
STATUS current
DESCRIPTION
"Collection of objects needed for RPR Span
Protection monitoring."
 ::= { rprGroups 6 }

rprTopoGroup OBJECT-GROUP
OBJECTS {
    rprTopoImageSecMacAddress1,
    rprTopoImageSecMacAddress2,
    rprTopoImageStationIfIndex,
    rprTopoImageStationName,
    rprTopoImageInetAddressType,
    rprTopoImageInetAddress,
    rprTopoImageCapability,
    rprTopoImageRinglet0Hops,
    rprTopoImageRinglet0ReservedRate,
    rprTopoImageRinglet1Hops,
    rprTopoImageRinglet1ReservedRate,
```

```

        rprTopoImageWestProtectionStatus,
        rprTopoImageWestWeight,
        rprTopoImageEastProtectionStatus,
        rprTopoImageEastWeight,
        rprTopoImageStatus
    }
STATUS current
DESCRIPTION
    "Collection of objects needed for RPR Topology
     discovery."
::= { rprGroups 7 }

rprFairnessGroup    OBJECT-GROUP
OBJECTS {
    rprFairnessRingletWeight,
    rprFairnessReservedRate,
    rprFairnessMaxAllowed,
    rprFairnessAgeCoef,
    rprFairnessRampCoef,
    rprFairnessLpCoef,
    rprFairnessAdvertisementRatio,
    rprFairnessMcffReportCoef,
    rprFairnessActiveWeightsCoef,
    rprFairnessSTQHighThreshold,
    rprFairnessSTQMedThreshold,
    rprFairnessSTQLowThreshold,
    rprFairnessRateHighThreshold,
    rprFairnessRateLowThreshold,
    rprFairnessResetWaterMarks,
    rprFairnessSTQHighWaterMark,
    rprFairnessSTQLowWaterMark
}
STATUS current
DESCRIPTION
    "Collection of objects needed for RPR Fairness
     configuration."
::= { rprGroups 8 }

rprFairnessGroupOpt   OBJECT-GROUP
OBJECTS {
    rprFairnessLastChange,
    rprFairnessChanges
}
STATUS current
DESCRIPTION
    "Collection of objects for RPR Fairness."
::= { rprGroups 9 }

rprOamGroup    OBJECT-GROUP
OBJECTS {
    rprOamActionType,
    rprOamDestAddress,
    rprOamRequestRinglet,
    rprOamResponseRinglet,
    rprOamClassOfService,
    rprOamUserData,
    rprOamProtected,
    rprOamRequestCount,
    rprOamTimeout,
}

```

```

        rprOamControl,
        rprOamResponseCount,
        rprOamAvResponseTime,
        rprOamResponseStatus
    }
STATUS current
DESCRIPTION
    "Collection of objects needed for RPR OAM
     configuration."
 ::= { rprGroups 10 }

rprIfChangeSummaryGroup   OBJECT-GROUP
OBJECTS {
    rprIfChangeSummaryNumInterfaces,
    rprIfChangeSummaryIfLastChange,
    rprIfChangeSummaryIfChanges,
    rprIfChangeSummarySpanLastChange,
    rprIfChangeSummarySpanChanges,
    rprIfChangeSummaryFairnessLastChange,
    rprIfChangeSummaryFairnessChanges
}
STATUS current
DESCRIPTION
    "Collection of objects summarizes changes on
     the RPR interfaces."
 ::= { rprGroups 11 }

rprSpanCurrentGroup   OBJECT-GROUP
OBJECTS {
    rprSpanCurrentInUcastClassAFrames,
    rprSpanCurrentInUcastClassAOctets,
    rprSpanCurrentInUcastClassBCirFrames,
    rprSpanCurrentInUcastClassBCirOctets,
    rprSpanCurrentInUcastClassBEirFrames,
    rprSpanCurrentInUcastClassBEirOctets,
    rprSpanCurrentInUcastClassCFrames,
    rprSpanCurrentInUcastClassCOctets,
    rprSpanCurrentInMcastClassAFrames,
    rprSpanCurrentInMcastClassAOctets,
    rprSpanCurrentInMcastClassBCirFrames,
    rprSpanCurrentInMcastClassBCirOctets,
    rprSpanCurrentInMcastClassBEirFrames,
    rprSpanCurrentInMcastClassBEirOctets,
    rprSpanCurrentInMcastClassCFrames,
    rprSpanCurrentOutUcastClassAFrames,
    rprSpanCurrentOutUcastClassAOctets,
    rprSpanCurrentOutUcastClassBCirFrames,
    rprSpanCurrentOutUcastClassBCirOctets,
    rprSpanCurrentOutUcastClassBEirFrames,
    rprSpanCurrentOutUcastClassBEirOctets,
    rprSpanCurrentOutUcastClassCFrames,
    rprSpanCurrentOutUcastClassCOctets,
    rprSpanCurrentOutMcastClassAFrames,
    rprSpanCurrentOutMcastClassAOctets,
    rprSpanCurrentOutMcastClassBCirFrames,
    rprSpanCurrentOutMcastClassBCirOctets,
    rprSpanCurrentOutMcastClassBEirFrames,
    rprSpanCurrentOutMcastClassBEirOctets,
}

```

```

        rprSpanCurrentOutMcastClassCFrames,
        rprSpanCurrentOutMcastClassCOctets
    }
STATUS current
DESCRIPTION
    "Collection of objects counting MAC span current
     statistics."
::= { rprGroups 12 }

rprSpanIntervalGroup   OBJECT-GROUP
OBJECTS {
    rprSpanIntervalValidData,
    rprSpanIntervalTimeElapsed,
    rprSpanIntervalStartTime,
    rprSpanIntervalInUcastClassAFrames,
    rprSpanIntervalInUcastClassAOctets,
    rprSpanIntervalInUcastClassBCirFrames,
    rprSpanIntervalInUcastClassBCirOctets,
    rprSpanIntervalInUcastClassBEirFrames,
    rprSpanIntervalInUcastClassBEirOctets,
    rprSpanIntervalInUcastClassCFrames,
    rprSpanIntervalInUcastClassCOctets,
    rprSpanIntervalInMcastClassAFrames,
    rprSpanIntervalInMcastClassAOctets,
    rprSpanIntervalInMcastClassBCirFrames,
    rprSpanIntervalInMcastClassBCirOctets,
    rprSpanIntervalInMcastClassBEirFrames,
    rprSpanIntervalInMcastClassBEirOctets,
    rprSpanIntervalInMcastClassCFrames,
    rprSpanIntervalInMcastClassCOctets,
    rprSpanIntervalOutUcastClassAFrames,
    rprSpanIntervalOutUcastClassAOctets,
    rprSpanIntervalOutUcastClassBCirFrames,
    rprSpanIntervalOutUcastClassBCirOctets,
    rprSpanIntervalOutUcastClassBEirFrames,
    rprSpanIntervalOutUcastClassBEirOctets,
    rprSpanIntervalOutUcastClassCFrames,
    rprSpanIntervalOutUcastClassCOctets,
    rprSpanIntervalOutMcastClassAFrames,
    rprSpanIntervalOutMcastClassAOctets,
    rprSpanIntervalOutMcastClassBCirFrames,
    rprSpanIntervalOutMcastClassBCirOctets,
    rprSpanIntervalOutMcastClassBEirFrames,
    rprSpanIntervalOutMcastClassBEirOctets,
    rprSpanIntervalOutMcastClassCFrames,
    rprSpanIntervalOutMcastClassCOctets
}
STATUS current
DESCRIPTION
    "Collection of objects counting MAC span intervals
     statistics."
::= { rprGroups 13 }

rprSpanDayGroup   OBJECT-GROUP
OBJECTS {
    rprSpanDayInUcastClassAFrames,
    rprSpanDayInUcastClassAOctets,
    rprSpanDayInUcastClassBCirFrames,
    rprSpanDayInUcastClassBCirOctets,

```

```
rprSpanDayInUcastClassBEirFrames,
rprSpanDayInUcastClassBEirOctets,
rprSpanDayInUcastClassCFrames,
rprSpanDayInUcastClassCOctets,
rprSpanDayInMcastClassAFrames,
rprSpanDayInMcastClassAOctets,
rprSpanDayInMcastClassBCirFrames,
rprSpanDayInMcastClassBCirOctets,
rprSpanDayInMcastClassBEirFrames,
rprSpanDayInMcastClassBEirOctets,
rprSpanDayInMcastClassCFrames,
rprSpanDayInMcastClassCOctets,
rprSpanDayOutUcastClassAFrames,
rprSpanDayOutUcastClassAOctets,
rprSpanDayOutUcastClassBCirFrames,
rprSpanDayOutUcastClassBCirOctets,
rprSpanDayOutUcastClassBEirFrames,
rprSpanDayOutUcastClassBEirOctets,
rprSpanDayOutUcastClassCFrames,
rprSpanDayOutUcastClassCOctets,
rprSpanDayOutMcastClassAFrames,
rprSpanDayOutMcastClassAOctets,
rprSpanDayOutMcastClassBCirFrames,
rprSpanDayOutMcastClassBCirOctets,
rprSpanDayOutMcastClassBEirFrames,
rprSpanDayOutMcastClassBEirOctets,
rprSpanDayOutMcastClassCFrames,
rprSpanDayOutMcastClassCOctets
}
STATUS current
DESCRIPTION
"Collection of objects counting MAC span 24 hours
statistics."
 ::= { rprGroups 14 }

rprSpanStatsGroup OBJECT-GROUP
OBJECTS {
    rprSpanStatsInUcastClassAFrames,
    rprSpanStatsInUcastClassAOctets,
    rprSpanStatsInUcastClassBCirFrames,
    rprSpanStatsInUcastClassBCirOctets,
    rprSpanStatsInUcastClassBEirFrames,
    rprSpanStatsInUcastClassBEirOctets,
    rprSpanStatsInUcastClassCFrames,
    rprSpanStatsInUcastClassCOctets,
    rprSpanStatsInMcastClassAFrames,
    rprSpanStatsInMcastClassAOctets,
    rprSpanStatsInMcastClassBCirFrames,
    rprSpanStatsInMcastClassBCirOctets,
    rprSpanStatsInMcastClassBEirFrames,
    rprSpanStatsInMcastClassBEirOctets,
    rprSpanStatsInMcastClassCFrames,
    rprSpanStatsInMcastClassCOctets,
    rprSpanStatsInCtrlFrames,
    rprSpanStatsInOamEchoFrames,
    rprSpanStatsInOamFlushFrames,
    rprSpanStatsInOamOrgFrames,
    rprSpanStatsInTopoAtdFrames,
    rprSpanStatsInTopoChkSumFrames,
```

```

        rprSpanStatsInTopoTpFrames,
        rprSpanStatsOutUcastClassAFrames,
        rprSpanStatsOutUcastClassAOctets,
        rprSpanStatsOutUcastClassBCirFrames,
        rprSpanStatsOutUcastClassBCirOctets,
        rprSpanStatsOutUcastClassBEirFrames,
        rprSpanStatsOutUcastClassBEirOctets,
        rprSpanStatsOutUcastClassCFrames,
        rprSpanStatsOutUcastClassCOctets,
        rprSpanStatsOutMcastClassAFrames,
        rprSpanStatsOutMcastClassAOctets,
        rprSpanStatsOutMcastClassBCirFrames,
        rprSpanStatsOutMcastClassBCirOctets,
        rprSpanStatsOutMcastClassBEirFrames,
        rprSpanStatsOutMcastClassBEirOctets,
        rprSpanStatsOutMcastClassCFrames,
        rprSpanStatsOutMcastClassCOctets,
        rprSpanStatsOutCtrlFrames,
        rprSpanStatsOutOamEchoFrames,
        rprSpanStatsOutOamFlushFrames,
        rprSpanStatsOutOamOrgFrames,
        rprSpanStatsOutTopoAtdFrames,
        rprSpanStatsOutTopoChkSumFrames,
        rprSpanStatsOutTopoTpFrames
    }
STATUS current
DESCRIPTION
    "Collection of objects counting MAC span total
     statistics."
 ::= { rprGroups 15 }

rprClientCurrentGroup    OBJECT-GROUP
OBJECTS {
    rprClientCurrentInUcastClassAFrames,
    rprClientCurrentInUcastClassAOctets,
    rprClientCurrentInUcastClassBCirFrames,
    rprClientCurrentInUcastClassBCirOctets,
    rprClientCurrentInUcastClassBEirFrames,
    rprClientCurrentInUcastClassBEirOctets,
    rprClientCurrentInUcastClassCFrames,
    rprClientCurrentInUcastClassCOctets,
    rprClientCurrentInMcastClassAFrames,
    rprClientCurrentInMcastClassAOctets,
    rprClientCurrentInMcastClassBCirFrames,
    rprClientCurrentInMcastClassBCirOctets,
    rprClientCurrentInMcastClassBEirFrames,
    rprClientCurrentInMcastClassBEirOctets,
    rprClientCurrentInMcastClassCFrames,
    rprClientCurrentInMcastClassCOctets,
    rprClientCurrentOutUcastClassAFrames,
    rprClientCurrentOutUcastClassAOctets,
    rprClientCurrentOutUcastClassBCirFrames,
    rprClientCurrentOutUcastClassBCirOctets,
    rprClientCurrentOutUcastClassBEirFrames,
    rprClientCurrentOutUcastClassBEirOctets,
    rprClientCurrentOutUcastClassCFrames,
    rprClientCurrentOutUcastClassCOctets,
    rprClientCurrentOutMcastClassAFrames,
    rprClientCurrentOutMcastClassAOctets,

```

```
rprClientCurrentOutMcastClassBCirFrames,
rprClientCurrentOutMcastClassBCirOctets,
rprClientCurrentOutMcastClassBEirFrames,
rprClientCurrentOutMcastClassBEirOctets,
rprClientCurrentOutMcastClassCFrames,
rprClientCurrentOutMcastClassCOctets
}
STATUS current
DESCRIPTION
"Collection of objects counting MAC client interface
current statistics."
 ::= { rprGroups 16 }

rprClientIntervalGroup OBJECT-GROUP
OBJECTS {
    rprClientIntervalValidData,
    rprClientIntervalElapsed,
    rprClientIntervalInUcastClassAFrames,
    rprClientIntervalInUcastClassAOctets,
    rprClientIntervalInUcastClassBCirFrames,
    rprClientIntervalInUcastClassBCirOctets,
    rprClientIntervalInUcastClassBEirFrames,
    rprClientIntervalInUcastClassBEirOctets,
    rprClientIntervalInUcastClassCFrames,
    rprClientIntervalInUcastClassCOctets,
    rprClientIntervalInMcastClassAFrames,
    rprClientIntervalInMcastClassAOctets,
    rprClientIntervalInMcastClassBCirFrames,
    rprClientIntervalInMcastClassBCirOctets,
    rprClientIntervalInMcastClassBEirFrames,
    rprClientIntervalInMcastClassBEirOctets,
    rprClientIntervalInMcastClassCFrames,
    rprClientIntervalInMcastClassCOctets,
    rprClientIntervalOutUcastClassAFrames,
    rprClientIntervalOutUcastClassAOctets,
    rprClientIntervalOutUcastClassBCirFrames,
    rprClientIntervalOutUcastClassBCirOctets,
    rprClientIntervalOutUcastClassBEirFrames,
    rprClientIntervalOutUcastClassBEirOctets,
    rprClientIntervalOutUcastClassCFrames,
    rprClientIntervalOutUcastClassCOctets,
    rprClientIntervalOutMcastClassAFrames,
    rprClientIntervalOutMcastClassAOctets,
    rprClientIntervalOutMcastClassBCirFrames,
    rprClientIntervalOutMcastClassBCirOctets,
    rprClientIntervalOutMcastClassBEirFrames,
    rprClientIntervalOutMcastClassBEirOctets,
    rprClientIntervalOutMcastClassCFrames,
    rprClientIntervalOutMcastClassCOctets
}
STATUS current
DESCRIPTION
"Collection of objects counting MAC client interface
intervals statistics."
 ::= { rprGroups 17 }

rprClientDayGroup OBJECT-GROUP
OBJECTS {
    rprClientDayInUcastClassAFrames,
```

```

        rprClientDayInUcastClassAOctets,
        rprClientDayInUcastClassBCirFrames,
        rprClientDayInUcastClassBCirOctets,
        rprClientDayInUcastClassBEirFrames,
        rprClientDayInUcastClassBEirOctets,
        rprClientDayInUcastClassCFCrames,
        rprClientDayInUcastClassCOctets,
        rprClientDayInMcastClassAFrames,
        rprClientDayInMcastClassAOctets,
        rprClientDayInMcastClassBCirFrames,
        rprClientDayInMcastClassBCirOctets,
        rprClientDayInMcastClassBEirFrames,
        rprClientDayInMcastClassBEirOctets,
        rprClientDayInMcastClassCFCrames,
        rprClientDayInMcastClassCOctets,
        rprClientDayOutUcastClassAFrames,
        rprClientDayOutUcastClassAOctets,
        rprClientDayOutUcastClassBCirFrames,
        rprClientDayOutUcastClassBCirOctets,
        rprClientDayOutUcastClassBEirFrames,
        rprClientDayOutUcastClassBEirOctets,
        rprClientDayOutUcastClassCFCrames,
        rprClientDayOutUcastClassCOctets,
        rprClientDayOutMcastClassAFrames,
        rprClientDayOutMcastClassAOctets,
        rprClientDayOutMcastClassBCirFrames,
        rprClientDayOutMcastClassBCirOctets,
        rprClientDayOutMcastClassBEirFrames,
        rprClientDayOutMcastClassBEirOctets,
        rprClientDayOutMcastClassCFCrames,
        rprClientDayOutMcastClassCOctets
    }
STATUS current
DESCRIPTION
"Collection of objects counting MAC client interface
24 hours statistics."
 ::= { rprGroups 18 }

rprClientStatsGroup OBJECT-GROUP
OBJECTS {
        rprClientStatsInUcastClassAFrames,
        rprClientStatsInUcastClassAOctets,
        rprClientStatsInUcastClassBCirFrames,
        rprClientStatsInUcastClassBCirOctets,
        rprClientStatsInUcastClassBEirFrames,
        rprClientStatsInUcastClassBEirOctets,
        rprClientStatsInUcastClassCFCrames,
        rprClientStatsInUcastClassCOctets,
        rprClientStatsInMcastClassAFrames,
        rprClientStatsInMcastClassAOctets,
        rprClientStatsInMcastClassBCirFrames,
        rprClientStatsInMcastClassBCirOctets,
        rprClientStatsInMcastClassBEirFrames,
        rprClientStatsInMcastClassBEirOctets,
        rprClientStatsInMcastClassCFCrames,
        rprClientStatsInMcastClassCOctets,
        rprClientStatsInBcastFrames,
        rprClientStatsOutUcastClassAFrames,
        rprClientStatsOutUcastClassAOctets,

```

```
rprClientStatsOutUcastClassBCirFrames,
rprClientStatsOutUcastClassBCirOctets,
rprClientStatsOutUcastClassBEirFrames,
rprClientStatsOutUcastClassBEirOctets,
rprClientStatsOutUcastClassCFrames,
rprClientStatsOutUcastClassCOctets,
rprClientStatsOutMcastClassAFrames,
rprClientStatsOutMcastClassAOctets,
rprClientStatsOutMcastClassBCirFrames,
rprClientStatsOutMcastClassBCirOctets,
rprClientStatsOutMcastClassBEirFrames,
rprClientStatsOutMcastClassBEirOctets,
rprClientStatsOutMcastClassCFrames,
rprClientStatsOutMcastClassCOctets,
rprClientStatsOutBcastFrames
}
STATUS current
DESCRIPTION
    "Collection of objects counting MAC client interface
     total statistics."
 ::= { rprGroups 19 }

rprErrorCurrentGroup   OBJECT-GROUP
OBJECTS {
    rprSpanErrorCurrentTtlExpFrames,
    rprSpanErrorCurrentTooLongFrames,
    rprSpanErrorCurrentTooShortFrames,
    rprSpanErrorCurrentBadHecFrames,
    rprSpanErrorCurrentBadFcsFrames,
    rprSpanErrorCurrentSelfSrcUcastFrames,
    rprSpanErrorCurrentPmdAbortFrames,
    rprSpanErrorCurrentBadAddrFrames,
    rprSpanErrorCurrentBadParityFrames,
    rprSpanErrorCurrentContainedFrames,
    rprSpanErrorCurrentScffErrors,
    rprSpanErrorCurrentErroredSeconds,
    rprSpanErrorCurrentSeverelyErroredSeconds,
    rprSpanErrorCurrentUnavailableSeconds
}
STATUS current
DESCRIPTION
    "Collection of objects counting MAC span error
     current statistics."
 ::= { rprGroups 20 }

rprErrorIntervalGroup   OBJECT-GROUP
OBJECTS {
    rprSpanErrorIntervalValidData,
    rprSpanErrorIntervalTimeElapsed,
    rprSpanErrorIntervalTtlExpFrames,
    rprSpanErrorIntervalTooLongFrames,
    rprSpanErrorIntervalTooShortFrames,
    rprSpanErrorIntervalBadHecFrames,
    rprSpanErrorIntervalBadFcsFrames,
    rprSpanErrorIntervalSelfSrcUcastFrames,
    rprSpanErrorIntervalPmdAbortFrames,
    rprSpanErrorIntervalBadAddrFrames,
    rprSpanErrorIntervalBadParityFrames,
    rprSpanErrorIntervalContainedFrames,
```

```

        rprSpanErrorIntervalScffErrors,
        rprSpanErrorIntervalErroredSeconds,
        rprSpanErrorIntervalSeverelyErroredSeconds,
        rprSpanErrorIntervalUnavailableSeconds
    }
STATUS current
DESCRIPTION
    "Collection of objects counting MAC span error
     intervals statistics."
 ::= { rprGroups 21 }

rprErrorDayGroup   OBJECT-GROUP
OBJECTS {
    rprSpanErrorDayTtlExpFrames,
    rprSpanErrorDayTooLongFrames,
    rprSpanErrorDayTooShortFrames,
    rprSpanErrorDayBadHecFrames,
    rprSpanErrorDayBadFcsFrames,
    rprSpanErrorDaySelfSrcUcastFrames,
    rprSpanErrorDayPmdAbortFrames,
    rprSpanErrorDayBadAddrFrames,
    rprSpanErrorDayBadParityFrames,
    rprSpanErrorDayContainedFrames,
    rprSpanErrorDayScffErrors,
    rprSpanErrorDayErroredSeconds,
    rprSpanErrorDaySeverelyErroredSeconds,
    rprSpanErrorDayUnavailableSeconds
}
STATUS current
DESCRIPTION
    "Collection of objects counting MAC span error
     24 hours statistics."
 ::= { rprGroups 22 }

rprErrorStatsGroup   OBJECT-GROUP
OBJECTS {
    rprSpanErrorStatsTtlExpFrames,
    rprSpanErrorStatsTooLongFrames,
    rprSpanErrorStatsTooShortFrames,
    rprSpanErrorStatsBadHecFrames,
    rprSpanErrorStatsBadFcsFrames,
    rprSpanErrorStatsSelfSrcUcastFrames,
    rprSpanErrorStatsPmdAbortFrames,
    rprSpanErrorStatsBadAddrFrames,
    rprSpanErrorStatsBadParityFrames,
    rprSpanErrorStatsContainedFrames,
    rprSpanErrorStatsScffErrors
}
STATUS current
DESCRIPTION
    "Collection of objects counting MAC span error
     total statistics."
 ::= { rprGroups 23 }

-- SAS Units of conformance
--

rprSasSummaryGroup OBJECT-GROUP

```

```

OBJECTS {
    rprSasSummaryDynamicCount,
    rprSasSummaryStaticUcastCount,
    rprSasSummaryStaticMcastCount,
    rprSasSummaryAgingTime,
    rprSasSummaryStdCleave
}
STATUS      current
DESCRIPTION
    "A collection of objects providing summary information about
     the SAS database and SAS global settings."
::= { rprGroups 24 }

rprSasSummaryPurgeGroup OBJECT-GROUP
OBJECTS {
    rprSasSummarySdbPurgeCmd,
    rprSasSummarySdbPurgeScopeSingle,
    rprSasSummarySdbPurgeScopeMap
}
STATUS      current
DESCRIPTION
    "A collection of objects providing access to SAS database
     purge functions."
::= { rprGroups 25 }

rprSasGroup OBJECT-GROUP
OBJECTS {
    rprSasDbTargetAddr,
    rprSasDbEntryStatus
}
STATUS      current
DESCRIPTION
    "A collection of objects providing information about dynamic
     entries within a given SDB."
::= { rprGroups 26 }

rprSasStaticUcastGroup OBJECT-GROUP
OBJECTS {
    rprSasStaticUcastVidOnly,
    rprSasStaticUcastTargetAddr,
    rprSasStaticUcastRowStatus
}
STATUS      current
DESCRIPTION
    "A collection of objects providing information about
     unicast addresses statically configured by
     management."
::= { rprGroups 27 }

rprSasStaticMcastGroup OBJECT-GROUP
OBJECTS {
    rprSasStaticMcastHopsRinglet0,
    rprSasStaticMcastHopsRinglet1,
    rprSasStaticMcastExtend,
    rprSasStaticMcastRowStatus
}
STATUS      current
DESCRIPTION

```

```

    "A collection of objects providing information about
    multicast addresses statically configured by
    management."
 ::= { rprGroups 28 }

rprSasStatsGroup OBJECT-GROUP
OBJECTS {
    rprSasCountersStatsLocalPurges,
    rprSasCountersStatsTopologyPurges,
    rprSasCountersStatsNotifyPurges,
    rprSasCountersStatsLearnFulls,
    rprSasCountersStatsTxDirectedFrames,
    rprSasCountersStatsTxUndirectedFrames,
    rprSasCountersStatsTxMcastScopedFrames
}
STATUS current
DESCRIPTION
    "A collection of objects providing statistics about
    SAS operation."
 ::= { rprGroups 29 }

rprSasSpanStatsGroup OBJECT-GROUP
OBJECTS {
    rprSpanStatsInOamSasNotifyFrames,
    rprSpanStatsOutOamSasNotifyFrames
}
STATUS current
DESCRIPTION
    "Collection of objects counting SAS span total
    statistics."
 ::= { rprGroups 30 }

-- PIRC Units of conformance

rprPircConfGroup OBJECT-GROUP
OBJECTS {
    rprPircConfPrimary,
    rprPircConfRevertive,
    rprPircConfLoadBalancing,
    rprPircConfFastMateMsgRpt,
    rprPircConfFastMateMsgInterval,
    rprPircConfSlowMateMsgInterval,
    rprPircConfWtrTimeout,
    rprPircConfPcTimeout,
    rprPircConfMateKeepAliveInterval,
    rprPircConfProtectionCommand}
STATUS current
DESCRIPTION
    "A collection of objects providing configuration of
    PIRC related attributes."
 ::= { rprGroups 31 }

rprPircSummaryGroup OBJECT-GROUP
OBJECTS {
    rprPircSummaryProtState,
    rprPircSummaryMateLastMsgTime,

```

```
rprPircSummaryMateMacAddress,
rprPircSummaryMatePrimary,
rprPircSummaryMateLoadbalancing,
rprPircSummaryMateOK,
rprPircSummaryMateManualSwitch}
STATUS      current
DESCRIPTION
  "A collection of objects providing information about
  PIRC status and variables."
 ::= { rprGroups 32 }

rprPircSpanStatsGroup   OBJECT-GROUP
  OBJECTS {
    rprSpanStatsInOamPircStatusFrames,
    rprSpanStatsOutOamPircStatusFrames
  }
STATUS      current
DESCRIPTION
  "Collection of objects counting PIRC span total
  statistics."
 ::= { rprGroups 33 }

END
```

Annex E

(normative)

CRC and parity calculations

E.1 Cyclic redundancy check 16-bit (CRC16) algorithmic definition

There is a 16-bit check symbol at the end of the data frame and control frame headers. The CRC uses the generator polynomial of Equation (E.1), which is performed on the most significant bits first:

$$x^{16} + x^{12} + x^5 + 1 \quad (\text{E.1})$$

E.1.1 Serial CRC16 calculation

The serial implementation of the CRC-16 polynomial, as applied to the least significant through most significant bits, is specified by the C-code calculations of Equation (E.2) and the hardware implementation illustrated in Figure E.1. The CRC calculation is conceptually computed in a byte-sequential fashion; for each byte, the checksum is updated eight times, once for each bit. The *hec* computations include the first byte of the header through the last byte of the header preceding the *hec*. When computing the checksum on a per-bit basis, the first-through-last of the data bits are the least significant through most significant bits of the byte, respectively:

```
// c00-through-c15 are the most through least significant bits of check. (E.2)
// d00 is the input value, sum is an intermediate value.
Sum= c00^d;
c00= c01;           c01= c02;           c02= c03;           c03= c04^sum;
c04= c05;           c05= c06;           c06= c07;           c07= c08;
c08= c09;           c09= c10;           c10= c11^sum;       c11= c12;
c12= c13;           c13= c12;           c14= c15;           c15= sum;
```

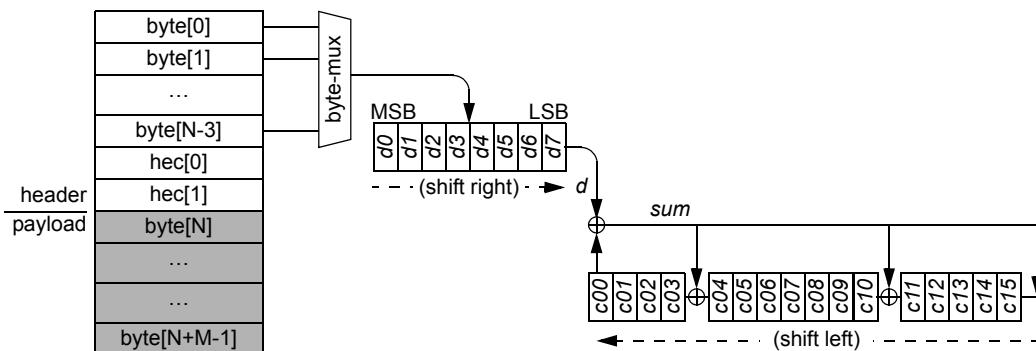


Figure E.1—Serial Crc16 reference model

The CRC16 calculation has several sometimes subtle characteristics, in addition to the basic polynomial-based CRC16 calculations, that could produce non-standard results if implemented differently. To reduce the possibility of creating non-standard implementations, these characteristics are summarized as follows:

- a) Startup. The CRC16 calculations start with an all-ones *crcSum* value.
- b) Completion. The computed CRC16 value is complemented before being appended to the frame.

E.1.2 CRC16 calculations

The generation and checking of 16-bit CRC values is optimized for bit-reversed (least significant to most significant bit within each sequential byte) transmission. The computation of such a CRC16 can be viewed as bit-swap operations on the boundaries of a consistent most significant to least significant bit parallelized CRC computation, as illustrated in Figure E.2. The bit-swap operations involve no circuitry and can be incorporated within the combining-EXOR circuitry.

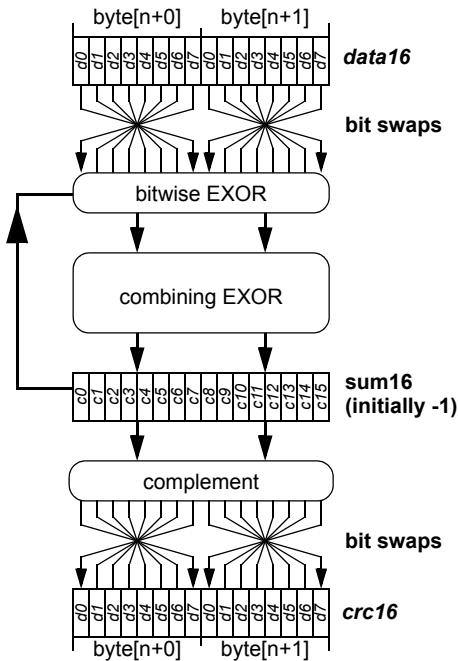


Figure E.2—CRC16 calculations

E.1.3 Protected header-field changes

The time to live field is normally decremented when frames pass through stations, so that corrupted frames can be discarded when the destination station is no longer present or is incorrectly identified. Decrementing the ttl (time to live) field or setting the ps (passed source) bit involves adjusting the following CRC field, as illustrated in Figure E.3.

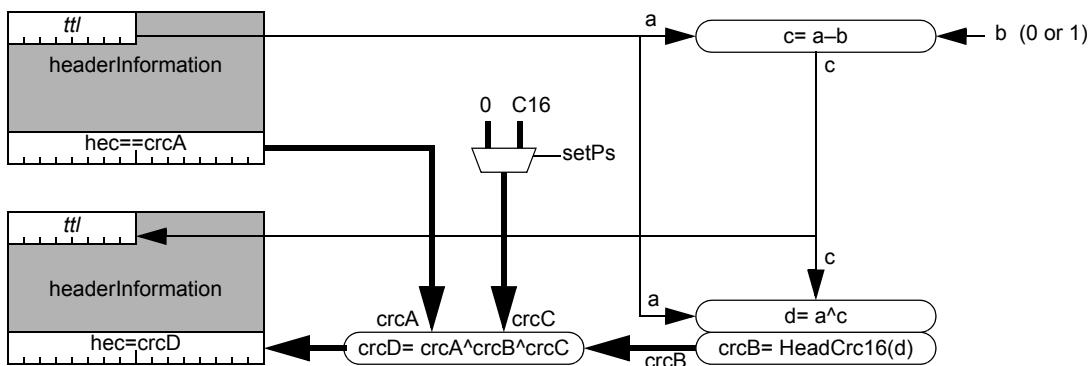


Figure E.3—Protected header-field changes

An incremental update of the CRC may be used to maintain CRC coverage when the *ttl* field is adjusted or when the *ps* bit is set. Implementations that wish to avoid the possibility of introducing errors or masking existing errors in the processing of calculating a new CRC will instead update the existing CRC by adjusting it to compensate for the change in value of the *ttl* or *ps* fields.

Adjusting the CRC for changes in the *ttl* involves computing the new *ttl* field value c and the difference d between new and old *ttl* values. The difference value d generates an incremental CRC value $crcB$.

In a similar fashion, setting the *ps* bit has the effect of generating a *crcC* value. For this computation, the *c16* value represents the bitwise changes in the *hec* value due to a changed *ps* bit.

The initial *crcA* value is EXORed with the *crcB* and *crcC* values to generate the new *crcD* value. The data are never left unprotected: An error in *crcA* or the computation of internal CRC values will (nearly) always be reflected as an error in check value *crcD*.

E.1.4 Illustration of CRC16 checks

To facilitate consistency checks, the $hec = CRC16$ value for a specific header bit pattern is illustrated in Figure E.4.

<i>ttl</i>	<i>baseControl</i>
<i>da{0...15}</i>	
<i>da{16...31}</i>	
<i>da{32...47}</i>	
<i>sa{0...15}</i>	
<i>sa{16...31}</i>	
<i>sa{0...15}</i>	
<i>ttlBase</i>	<i>extendedControl</i>
<i>hec</i>	

09_{16}	74_{16}
0012_{16}	
3456_{16}	
7890_{16}	
0012_{16}	
3498_{16}	
7654_{16}	
09_{16}	00_{16}
6064_{16}	

$0000\ 1001_2$	$0111\ 0100_2$
$0000\ 0000\ 0001\ 0010_2$	
$0011\ 0100\ 0101\ 0110_2$	
$0111\ 1000\ 1001\ 0000_2$	
$0000\ 0000\ 0001\ 0010_2$	
$0011\ 0100\ 1001\ 1000_2$	
$0111\ 0110\ 0101\ 0100_2$	
$0000\ 1001_2$	$0000\ 0000_2$
$0110\ 0000\ 0110\ 0100_2$	

Figure E.4—Illustration of CRC16 checks

E.2 Cyclic redundancy check 32-bit (CRC32) algorithmic definition

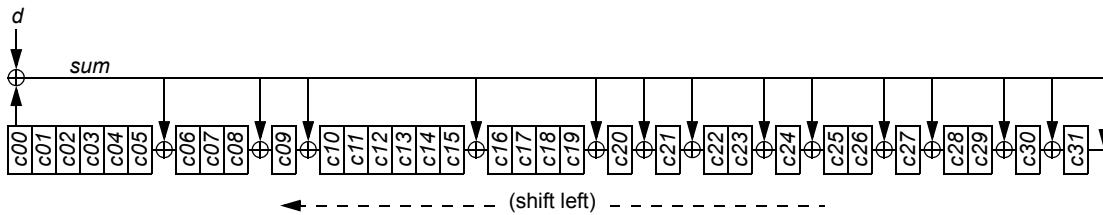
There is a 32-bit check symbol at the end of the payload. The CRC that is used is the same CRC used by IEEE 802 LANs and FDDI. The CRC uses the generator polynomial of Equation (E.3), which is performed on the least significant bits first:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (\text{E.3})$$

E.2.1 Serial CRC32 calculation

The serial implementation of the CRC-32 polynomial, as applied to the least significant through most significant bits, is specified by the C-code calculations of Equation (E.4) and the hardware implementation illustrated in Figure E.5:

```
// c00-through-c31 are the most through least significant bits of check. (E.4)
// d00 is the input value, sum is an intermediate value.
Sum= c00^d00;
c00= c01;           c01= c02;           c02= c03;           c03= c04;
c04= c05;           c05= c06^sum;       c06= c07;           c07= c08;
c08= c09^sum;       c09= c10^sum;       c10= c11;           c11= c12;
c12= c13;           c13= c12;           c14= c15;           c15= c16^sum;
c16= c17;           c17= c18;           c18= c19;           c19= c20^sum;
c20= c21^sum;       c21= c22^sum;       c22= c23;           c23= c24^sum;
c24= c25^sum;       c25= c26;           c26= c27^sum;       c27= c28^sum;
c28= c29;           c29= c30^sum;       c30= c31^sum;       c31= sum;
```

**Figure E.5—Serial CRC32 reference model**

The CRC calculation has several sometimes subtle characteristics, in addition to the basic polynomial-based CRC calculations, that could produce non-standard results if implemented differently. To reduce the possibility of creating non-standard implementations, these characteristics are summarized below.

- Startup. The CRC calculations start with an all-ones crcSum value.
- Completion. The computed CRC value is complemented before being appended to the frame.

E.2.2 Exchanged ExorSum calculations

The generation and checking of 32-bit CRC values is optimized for bit-reversed (least significant to most significant bit within each sequential byte) transmission. The computation of such a CRC32 can be viewed as bit-swap operations on the boundaries of a consistent least significant to most significant bit parallelized CRC computation, as illustrated in Figure E.6. The bit-swap operations involve no circuitry and can be incorporated within the combining-EXOR circuitry.

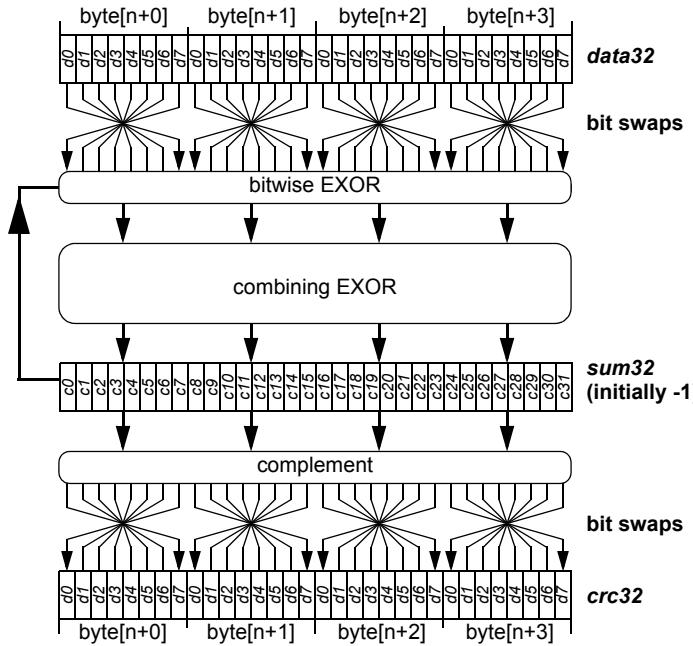


Figure E.6—Exchanged EXORSum calculations

E.2.3 Payload CRC stomping

When a frame with a valid header CRC or header parity and an invalid payload CRC is detected, either as a frame is being transmitted (via cut-through frame processing) or before a frame is transmitted (via store-and-forward processing), the frame continues circulating with the payload CRC marked invalid.

With this invalidation strategy, a payload transmission error causes an error to be logged and the frame's CRC set to a well-defined “stomped” value. That stomped value is also an invalid CRC value, but further logging of the error condition is inhibited. These erroneous-CRC processing steps are illustrated in Figure E.7.

A new CRC value, called *check*, is computed based on the frame's contents. The *checkStomp* value is computed by EXORing the *check* value with a STOMP value (STOMP is a 32-bit constant FFFFFFFFFF_{16} value). If the frame's *crcA* differs from the computed *check* value, the revised *crcB* value is set to the *checkStomp* value. An error condition is flagged if the frame's CRC value is incorrect (*crcA*! = *check*) and the error has apparently not been previously flagged (*crcA*! = *checkStomp*). In all other respects, the frame is treated as an unerrored frame.

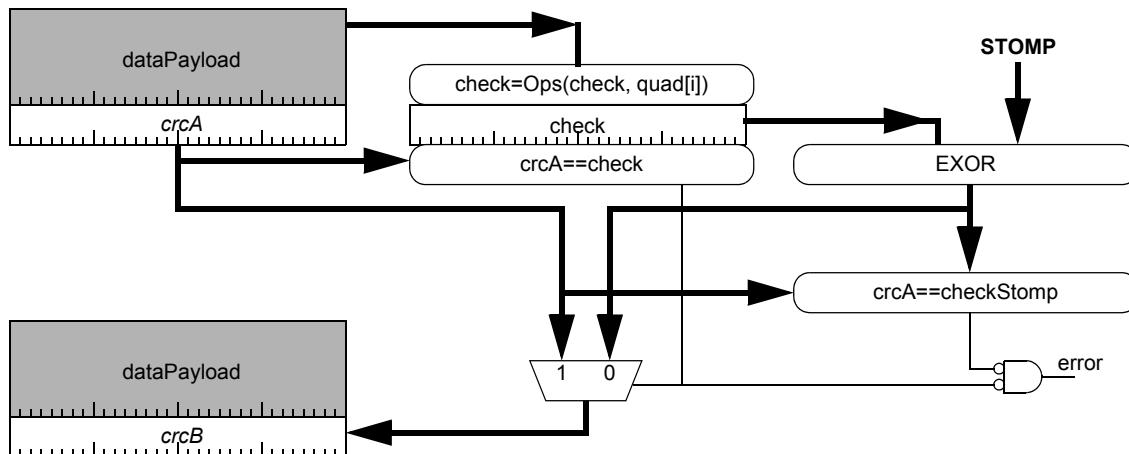


Figure E.7—Data CRC stomping

E.2.4 Illustration of CRC32 checks

To facilitate consistency checks, the $fcs = \text{CRC32}$ value for a specific header bit pattern is illustrated in Figure E.8.

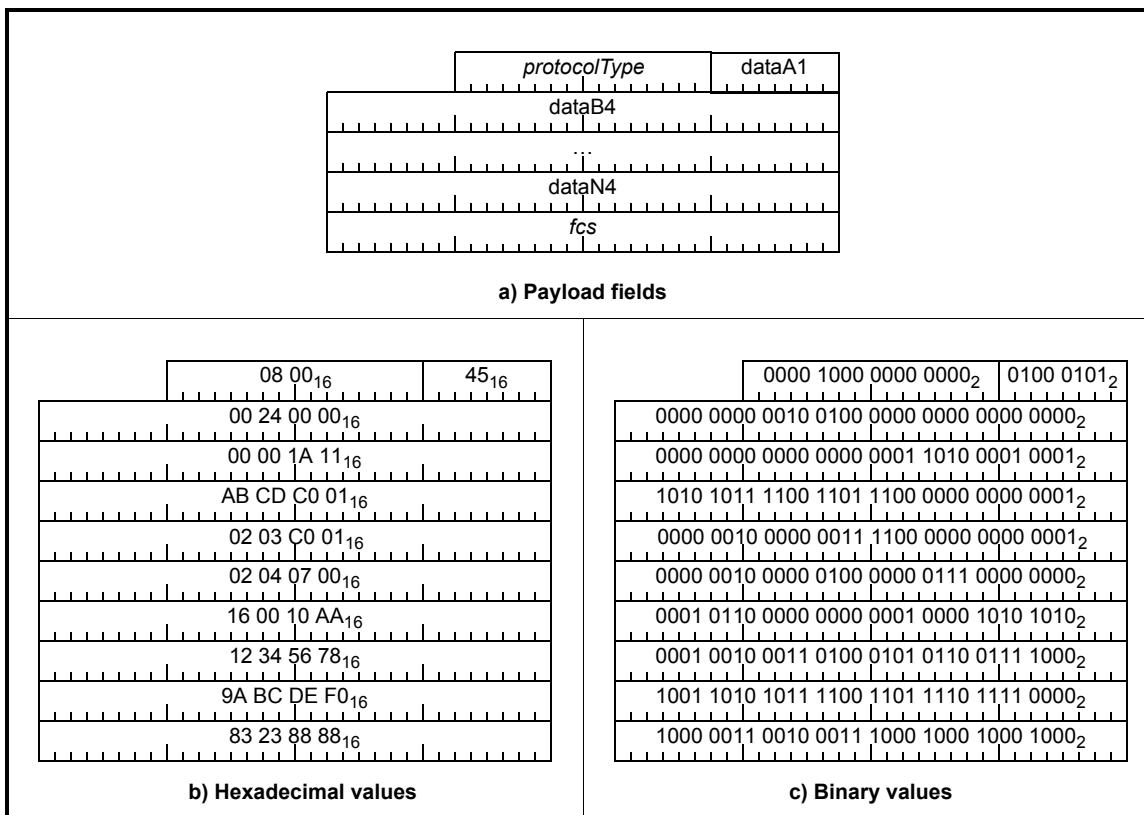


Figure E.8—Illustration of CRC32 checks

E.3 Parity algorithmic definition

There is a parity bit at the end of the idle frame and fairness frame headers. The parity generation is based on an exclusive-OR of the preceding bits within the header, as specified in Equation (E.5):

$$d_{15} = d_0 \oplus d_1 \oplus d_2 \oplus d_3 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_8 \oplus d_9 \oplus d_{10} \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus 1 \quad (\text{E.5})$$

E.3.1 Parity calculation

The implementation of the parity is specified by the C-code calculations of Equation (E.6) and the hardware implementation illustrated in Figure E.9:

```
// Parity generation of the sum-bit value
for (i=0, sum=1; i < 15; i += 1)
    sum ^= (data >> (15-i)) & 1;
```

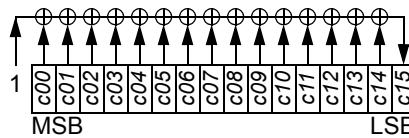
(E.6)


Figure E.9—Parity generation model

E.3.2 Illustration of fairness-frame checks

To facilitate implementation-correctness checks, the *parity* bit and *fcs* = *CRC32* values for a fairness frame are illustrated in Figure E.10.

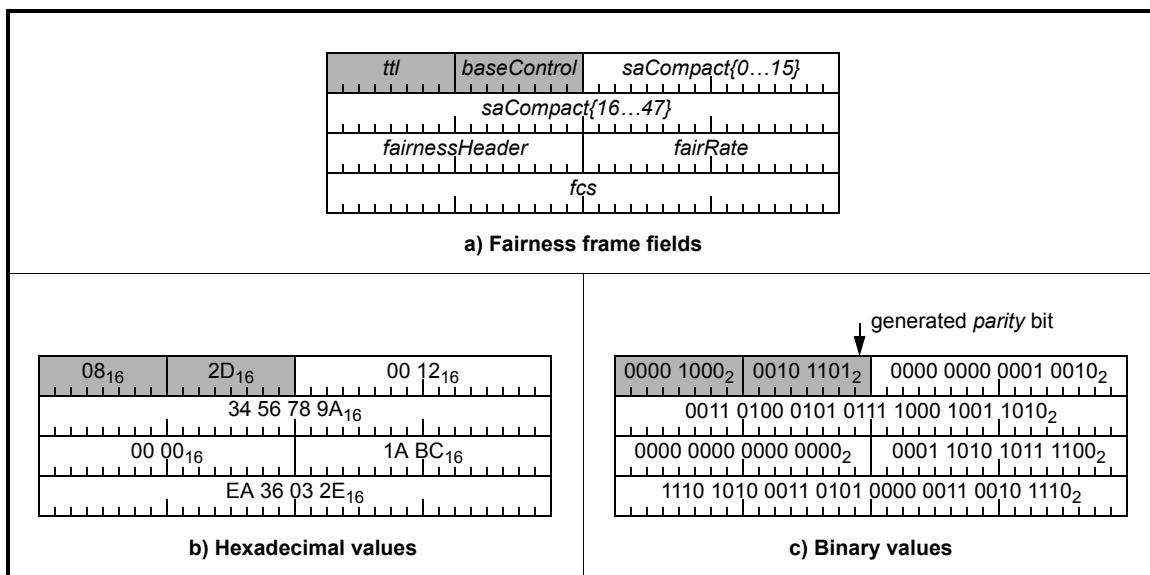


Figure E.10—Illustration of fairness-frame checks

E.4 Protocol Implementation Conformance Statement (PICS) proforma for Annex E³⁴

E.4.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Annex E, CRC and parity calculations, shall complete the following Protocol Implementation Conformance Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-2005.

E.4.2 Identification

E.4.2.1 Implementation identification

Supplier ^a	
Contact point for enquiries about the PICS ^a	
Implementation Name(s) and Version(s) ^{a,c}	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) ^b	

^aRequired for all implementations.

^bMay be completed as appropriate in meeting the requirements for the identification.

^cThe terms *Name* and *Version* should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

E.4.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-2011, Resilient packet ring access method and physical layer specifications, CRC and parity calculations
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [] Yes [] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-2011.)	

Date of Statement	
-------------------	--

³⁴*Copyright release for PICS proformas:* Users of this standard may freely reproduce the PICS proforma in this annex so that it can be used for its intended purpose and may further publish the completed PICS.

E.4.3 PICS tables for Annex E

E.4.3.1 Header error check

Item	Feature	Subclause	Value/Comment	Status	Support
HEC1	<i>hec</i> field	E.1	<i>hec</i> fields are created following Equation E.1	M	Yes []
HEC2	<i>hec</i> field changes	E.1.3	<i>hec</i> fields are updated instead of recalculated	O	Yes [] No []

E.4.3.2 Frame check sequence

Item	Feature	Subclause	Value/Comment	Status	Support
FCS1	<i>fcs</i> field	E.2.2	<i>fcs</i> fields are created following Equation E.3	M	Yes []
FCS2	<i>fcs</i> stomp	E.2.3	invalid fcs fields are stomped	M	Yes []

E.4.3.3 Parity

Item	Feature	Subclause	Value/Comment	Status	Support
PAR1	parity field	E.3	<i>parity</i> fields are created following Equation E.5	M	Yes []

Annex F

(informative)

802.1D and 802.1Q bridging conformance

F.1 Bridging overview

This annex is not intended to define a new bridging specification. IEEE Std 802.1D-2004 specifies an architecture and protocol for the interconnection of IEEE 802 LANs below the MAC service boundary. Within this context, an RPR network defines a ring topology forming a broadcast medium where specific access control mechanisms are employed by the MAC in order to achieve frame delivery and spatial reuse on the ring medium. The purpose of this annex is to ensure and demonstrate that the RPR MAC definition conforms to Transparent Bridging and VLAN Bridging, as defined in IEEE Std 802.1D-2004 and IEEE Std 802.1Q-2005.

F.1.1 802 bridging reference model

The MAC bridging reference model for the RPR MAC is shown in Figure F.1. The bridging reference model supports an RPR network consisting of end stations and IEEE Std 802.1D-2004 (or IEEE Std 802.1Q-2005) compliant transparent bridges, whereas an RPR network acts as the shared broadcast medium. Traffic may originate or terminate at either local or remote end stations and may be forwarded across an RPR network to other 802 networks by transparent bridges. Local end stations are end stations that directly attach to an RPR network. Remote end stations are end stations that originate and terminate LAN traffic, which is forwarded across an RPR network via transparent bridges. RPR stations operating as transparent bridges forward traffic between an RPR network and their other associated LAN networks.

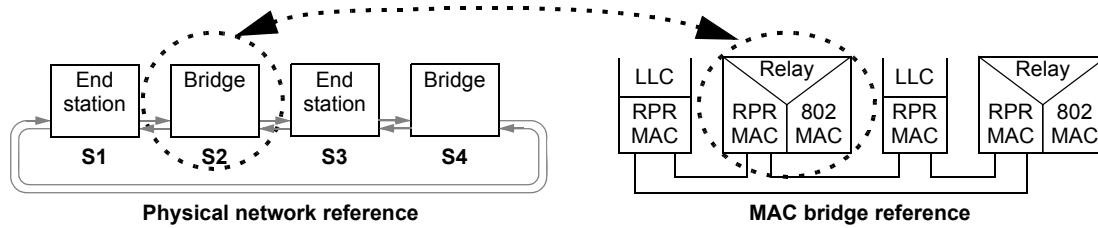


Figure F.1—Bridge architecture reference

The RPR MAC entity appears as a single port to the 802 bridging relay. The RPR MAC protocol is responsible for making the collection of stations attached to the ring appear as a single loop-free broadcast network to the 802 bridge relay.

Spanning tree protocol (STP) supports, preserves, and maintains the quality of the MAC service in all its aspects when operating over RPR networks as discussed in IEEE Std 802.1D-2004, Clause 6. Given that the RPR MAC entity (which encompasses two physical interfaces and MAC data paths, etc.) appears as a single port to the 802 bridging relay, STP will not interfere with the transit path of frames on the RPR media. This is shown in Figure F.2-a, where spanning tree protocol is operating in a network consisting of multiple bridges attached to an RPR network. As the RPR network does not form a loop in the bridged network, spanning tree will not block any of the RPR bridge ports. Other scenarios, spanning tree will block ports to remove loops in the network topology in cases where a loop is detected between the RPR network, any bridges attached to the RPR network, and other 802 networks. This may result in either an RPR port or some

other bridge port being blocked by spanning tree to eliminate the loops. This scenario is highlighted in Figure F.2-b, where a loop is formed between the RPR ring and the secondary path between stations S3 and S5. Spanning tree protocol detects the loop between stations S3 and S5 and blocks one of the bridge ports to eliminate the loop.

The RPR MAC entity also provides LLC services to support the bridge protocol entity and other higher layer protocol users.

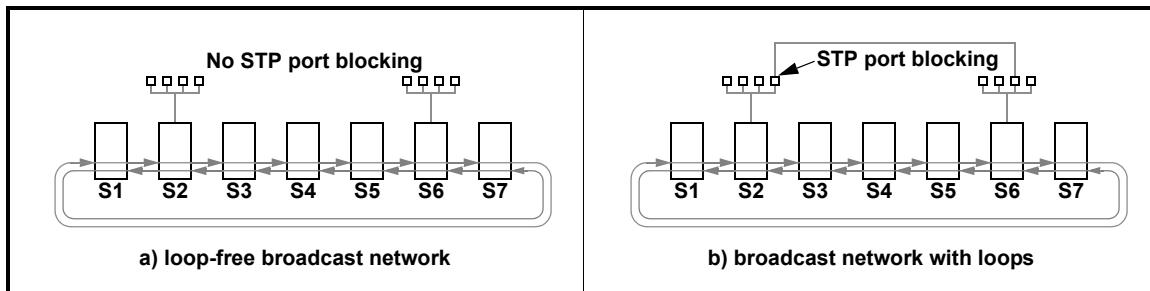


Figure F.2—Loop-free and looped broadcast networks

F.1.2 RPR support of the MAC service

The RPR MAC ensures that a frame is delivered to the intended bridge relay or host client entities. RPR MAC procedures ensure that duplicate or misordered frames that may be received at the MAC interface are not transferred to the Internal Sublayer Service (ISS).³⁵ Misordering and duplicate suppression is assured for frames transmitted onto an RPR network during normal ring operation, as well as under all failure, protection, and topology change scenarios.

In order to support transparent bridging of 802 traffic and maintain the spatial reuse property of the ring, the RPR MAC service interface performs the following functions: simple mapping of 802 traffic to the RPR frame format, transport of 802 traffic across the RPR physical medium, and delivery of 802 traffic to either the bridge relay or intended MAC client at the RPR MAC service interface. The MAC procedures maintain the spatial reuse property of the ring for local traffic. The mapping function performed by the RPR MAC service interface conforms to the interface between a MAC entity and bridge relay and preserves the filtering services and other requirements for bridged LANs as specified in IEEE Std 802.1D-2004 and IEEE Std 802.1Q-2005.

F.1.3 Transmission between local and remote end stations

Traffic transmitted on an RPR network may originate from end stations that are either local or remote to an RPR network. Locally sourced traffic is sourced by end stations that directly attach to the ring (see Figure F.3-a, Figure F.3-c, and Figure F.3-e). Remotely sourced traffic is sourced by remote end stations that do not directly attach to the ring, but whose traffic is bridged onto the ring by a transparent bridge (see Figure F.3-b, Figure F.3-d, and Figure F.3-f). Similarly, traffic on an RPR network may either be terminated by one or more local end stations, one or more remote end stations (via attached bridges), or a combination of local and remote end stations (via attached bridges). RPR traffic that is either sourced or terminated by a remote end station must pass through a bridge that is attached to the ring.

Figure F.3 shows examples of reference networks illustrating the different types of network transmissions that can occur between hosts and bridges on the ring. In this figure, odd-numbered stations (S1, S3, S5, S7) are end stations (hosts), and even-numbered stations (S2, S4, S6) are bridges. For the purpose of these

³⁵ISS is defined in 6.4 and 6.5 of IEEE Std 802.1D-2004.

examples, traffic either originating or terminating from an end station is indicated by a dark-shaded box (see S1 of Figure F.3-a). Traffic traversing the RPR network is indicated by a dark-shaded line that follows the path of the ring. Traffic that is bridged between the RPR network and its associated 802 network is indicated by a dark-shaded LAN segment and dark-shaded arrow connecting the 802 LAN segment with the RPR network (see S2 of Figure F.3-b). Traffic received by the RPR MAC are indicated by a dark upward-turned line from the ringlet. Shorter arrows terminating on a perpendicular line (see S4 of Figure F.3-b) indicate frames that are delivered by the RPR MAC to its client, which are filtered by the client. Frames that are stripped (and not received) are indicated by an arrow terminating on a perpendicular line (see S1 of Figure F.3-b).

The following bridged network scenarios are supported:

- Local end station transmitting unicast traffic to another local end station (see Figure F.3-a).
- Remote end station transmitting unicast traffic to a local end station (via a bridge attached to RPR network) (see Figure F.3-b).
- Local end station transmitting unicast traffic to a remote end station (via a bridge attached to RPR network) (see Figure F.3-c).
- Remote end station transmitting unicast traffic to a remote end station (via bridges attached to RPR network) (see Figure F.3-d).
- Local end station transmitting multicast traffic (see Figure F.3-e).
- Remote end station transmitting multicast traffic (see Figure F.3-f).

Further details on the types of local/source data transfers on the RPR network and their associated frame formats are provided in 5.9.

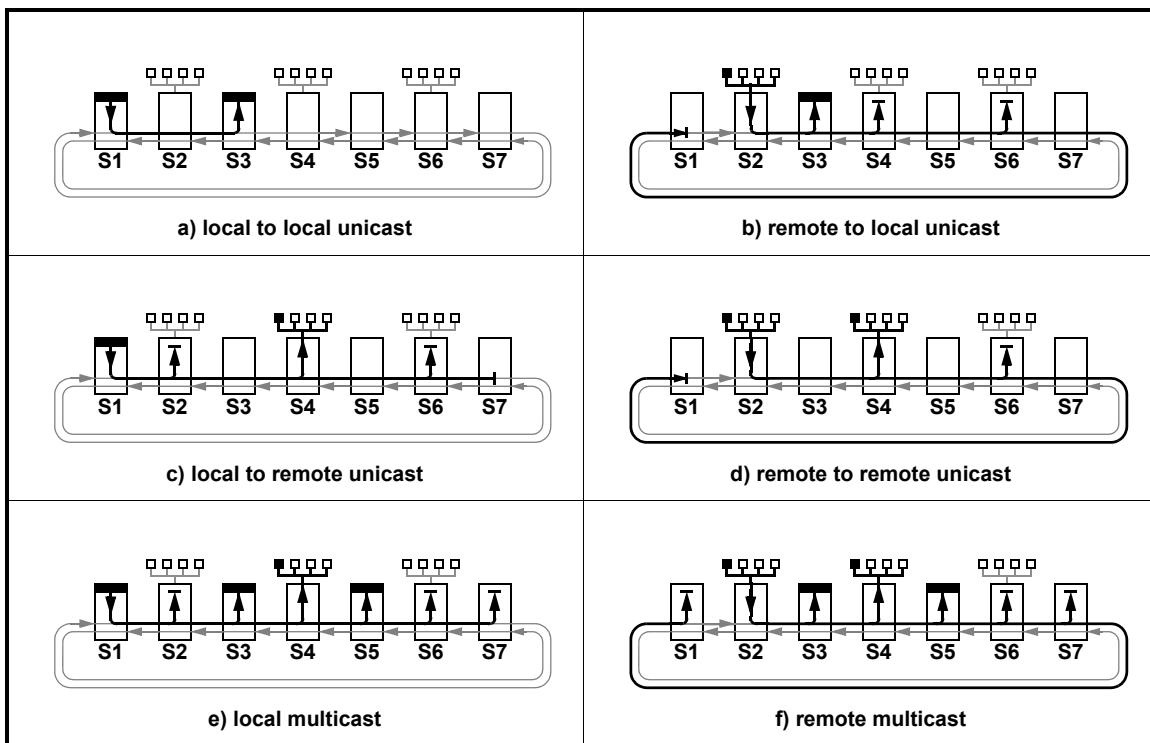


Figure F.3—Local/remote end station transmission scenarios

F.1.4 Maintaining filtering integrity of the 802 bridged network

The bridging conformance model ensures the integrity of RPR attached bridges and their associated bridged networks is maintained for the various network scenarios involving bridges and end stations attached to the ring. Bridge learning and forwarding procedures can be compromised in scenarios where a bridge sees unicast traffic on an RPR network that is destined to a specific MAC address but never sees traffic sourced from this MAC address. This is illustrated in Figure F.4 and Figure F.6 and the examples that follow. If these scenarios were to occur, the particular MAC address is never learned by the bridge, and anytime the bridge sees traffic destined to this MAC address, the bridge floods this traffic to all its other network ports. This is a significant problem resulting in a persistent flooding behavior affecting the rest of the 802 bridged network. These scenarios are prevented by the RPR MAC.

There are two problem traffic scenarios that the RPR MAC procedures specifically address in order to maintain integrity of the bridged network:

- a) Asymmetric traffic flows between a pair of local end stations and intermediate bridges on an RPR network.
- b) Traffic originating from a remote end station (via a bridge) that is destined to a local end station on the ring, where the local end station strips the frame from the ring prior to other bridges on the ring learning the remote end station's MAC address from the transmitted frame.

The following subclauses (F.1.4.1–F.1.4.2) describe the nature of these issues, and how the RPR MAC procedures ensure these scenarios do not occur.

F.1.4.1 Asymmetric traffic flows between a pair of local end stations

The asymmetric traffic scenario involves a pair of local end stations and one or more bridges on the ring. Frames transmitted by the first end station traverse a subset of the stations and are stripped by the second end station before being received by other stations on the ring. Similarly, frames transmitted by the second end station traverse a different subset of intermediate stations and are stripped by the first end station (see Figure F.4). Each of the intermediate stations on the ring only receive frames sourced by one of the end stations, but not others.

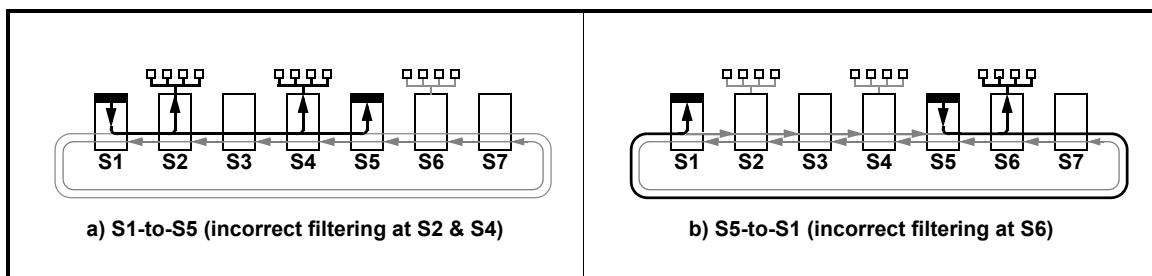


Figure F.4—Asymmetric traffic flow issue

The following two examples discuss scenarios involving asymmetric traffic flows. The first example shows how operation of the bridged network can be compromised if the RPR MAC were not to correctly handle asymmetric traffic flows. The second example shows how the RPR MAC properly supports asymmetric traffic flows.

F.1.4.1.1 Asymmetric traffic flow issue

If bridges on the RPR network receive asymmetric traffic flows between a pair of stations on the RPR ring, they would only see traffic sourced by one of the local end stations and not the other. They would never learn the other local station's source address. As a result, all the bridges on the ring continually receive frames intended for unknown destination addresses. These frames would be incorrectly flooded to all adjoining 802 networks because the bridges are unable to learn the local station's MAC address.

The following example (see Figure F.4) illustrates how an incorrect RPR MAC implementation can adversely affect 802 bridged network if it does not properly handle asymmetric traffic flows. In these examples, even-numbered stations (S2, S4, and S6) are bridges and odd-numbered stations (S1, S3, S5, and S7) are end stations (hosts). End stations S1 and S5 have MAC addresses MAC A and MAC B, respectively.

- a) End station S1 (MAC A) transmits unicast frames to end station S5 (MAC B).
These frames are received and incorrectly flooded by {S2, S4}, and stripped by S5.
 - 1) The S1-to-S5 frames are incorrectly flooded by bridge stations {S2, S4} to all their adjoining 802 networks, because destination address (MAC B) is unknown in bridge stations' {S2, S4} filtering data bases (FDBs).
 - 2) S1's MAC address (MAC A) is not learned by S6 because these frames are stripped by S5.
- b) End station S5 (MAC B) transmits unicast frames to end station S1 (MAC A).
These frames are received and incorrectly flooded by S6 and stripped by S1.
 - 1) The S5-to-S1 frames are incorrectly flooded by bridge station S6 to all its adjoining 802 networks, because destination address (MAC A) is unknown in bridge station's S6 filtering data base (FDB).
 - 2) S5's MAC address (MAC B) is not learned by {S2, S4} because the frame is stripped by S1.

The persistent flooding sequence described above in step a) and step b) repeats indefinitely, because bridges {S2, S4, S6} receive frames from only one of the end stations. Proper MAC address learning is unable to occur because they do not receive frames transmitted by the other end station.

F.1.4.1.2 RPR MAC handling of asymmetric traffic flows

The RPR MAC correctly handles asymmetric traffic flows between pairs of local end stations by ignoring frames that are addressed to other stations on the ring. The RPR MAC uses the flooding indication and MAC destination address in the frame to determine whether the frame should be passed to its bridge relay client. Intermediate bridges receiving frames that are destined to another station having a flooding indicator of FI_NONE are ignored by the MAC. This illustrated by Figure F.5 and the following example:

- a) End station S1 (MAC A) transmits unicast frames to end station S5 (MAC B).
These frames are received and stripped by S5.
- b) End station S5 (MAC B) transmits unicast frames to end station S1 (MAC A).
These frames are received and stripped by S1.

Local frames are correctly delivered to their intended destination. Hosts can employ destination stripping of local unicast traffic without affecting bridges on the ring. Intermediate bridges do not have the issue of asymmetric learning, because the MAC does not deliver local frames to the bridge relay.

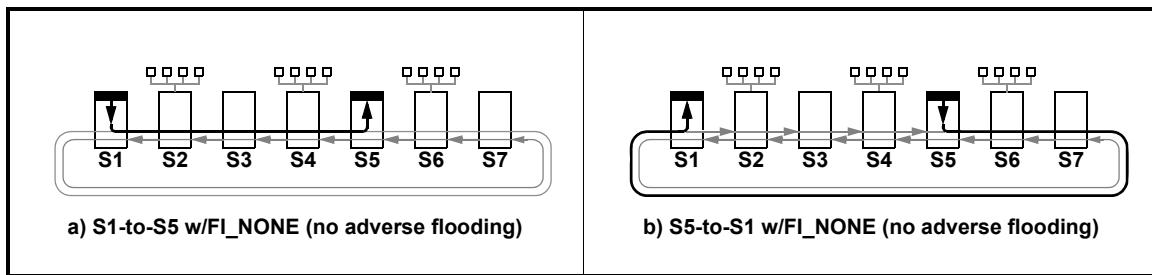


Figure F.5—RPR MAC handling of asymmetric traffic flows

F.1.4.2 Remote bridged traffic

The RPR MAC properly handles remote traffic that is bridged onto the RPR ring. Remote bridge traffic is traffic that originates from a remote end station that is bridged onto the ring by an 802 compliant bridge (see Figure F.6). Intermediate bridges can be affected by remote bridge traffic if the remote bridge traffic is prematurely stripped prior to being received by all bridges on the ring. The following two examples discuss handling of remote bridged traffic. The first example discusses how operation of the bridged network can be compromised if the RPR MAC does not correctly handle remote bridge traffic flows. The second example shows how the RPR MAC properly handles remote bridge traffic.

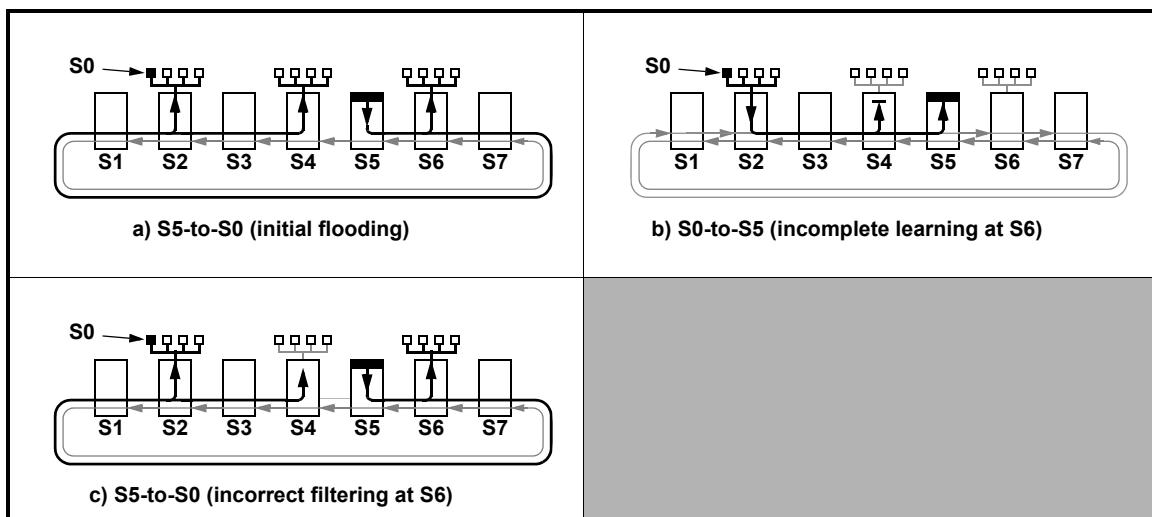


Figure F.6—Incorrect bridge filtering behavior

F.1.4.2.1 Remote bridge traffic issue

Traffic from a remote end station that is bridged onto the ring can compromise the filtering operation of other bridges on the ring, if the remote traffic is prematurely stripped by a local station on the ring. A local station might consider stripping these frames if they are addressed to the local station's MAC address. This creates problems for any bridges that are downstream from the stripping point, because they are unable to learn the remote end station's MAC address. Any subsequent frames transmitted on the ring that are sent to this remote station's MAC address are seen as unknown by the bridges. The bridges in turn flood these frames to all adjoining 802 networks creating a persistent flooding problem.

The destination stripping issue is illustrated in Figure F.6. The RPR network consists of bridges (S2, S4, and S6), and end stations (S1, S3, S5, and S7) that are all attached to the RPR network. End station S0 is attached to the network via bridge S2. End stations (S0, S5) have MAC addresses A and B, respectively. The following series of events can lead to this issue:

- a) End station S5 (MAC B) transmits a unicast frame to end station S0 (MAC A).
The frame is flooded on the RPR ring and correctly flooded by bridges S2, S4, and S6.
 - 1) Because destination address MAC A is unknown by S2, S4, and S6 bridge FDBs, this frame is flooded on all other 802 networks attached to these bridges and correctly received by S0.
 - 2) Learning takes place by all bridges on the network. MAC B is learned and associated with each bridge's receive interface.
- b) End station S0 (MAC A) transmits a unicast frame to end station S5 (MAC B).
The frame is transmitted on the RPR ring, received by stations S4 and S5, and stripped by S5.
 - 1) Bridge S4 correctly receives, learns, and filters the frame. Bridge S4 learns source address MAC A in its FDB and associates it with its RPR interface.
 - 2) Because the frame is incorrectly stripped by S5, bridge S6 never sees frames originating from end station S0. Consequently, bridge S6 is unable to learn and associate source address MAC A in its FDB.
- c) End station S5 (MAC B) transmits unicast frames to end station S0 (MAC A).
These frames are flooded on the RPR ring and erroneously flooded by bridge S6.
 - 1) Because destination address MAC A is now known by bridges S2 and S4, these frames are correctly forwarded/filtered. Bridge S2 forwards to S0. Bridge S4 filters the frame.
 - 2) Because MAC A is never learned by bridge S6, any frames destined to MAC A that are received by S6 are incorrectly filtered. Consequently, these frames are persistently flooded on all their adjacent LAN networks.

The filtering mechanism in bridge S6 is compromised as a result of destination stripping of remote bridge traffic by S5. The RPR MAC addresses this issue by ensuring that traffic from remote end stations is always flooded and never destination stripped by a local station on the ring.

F.1.4.2.2 Correct handling of remote bridge traffic

The correct MAC behavior is illustrated in Figure F.7. In this case, all bridged traffic is flooded on the ring. The correct sequence of events between remote end station A and local end station B is as follows:

- a) End station S5 (MAC B) transmits a unicast frame to end station S0 (MAC A).
The frame is flooded on the RPR ring and correctly flooded by bridges (S2, S4, and S6).
 - 1) Bridges (S2, S4, S6) learn S5 source MAC address MAC B in their FDB and associate MAC B with the RPR network.
- b) End station S0 (MAC A) transmits a unicast frame to end station S5 (MAC B).
The frame is flooded on the RPR ring and received by stations (S4, S5, and S6).
 - 1) The frame is received by end station S5, but not stripped. This allows other stations to learn the remote station's source address.
 - 2) Bridges (S4 and S6) learn S0 source address MAC A in their FDB and associate S0 with the RPR network.
- c) End station S5 (MAC B) transmits unicast frames to end station S0 (MAC A).
These frames are flooded on the RPR ring and correctly filtered by all bridges on the RPR.
 - 1) Because destination address MAC A is now known by bridges (S2, S4, and S6), these frames are correctly forwarded/filtered. Bridge S2 forwards to S0. Bridges (S4, and S6) filter the frame.

The RPR MAC ensures that remote bridged traffic is always flooded and never destination stripped, avoiding the issues previously described in F.1.4.1.1 and F.1.4.2.1.

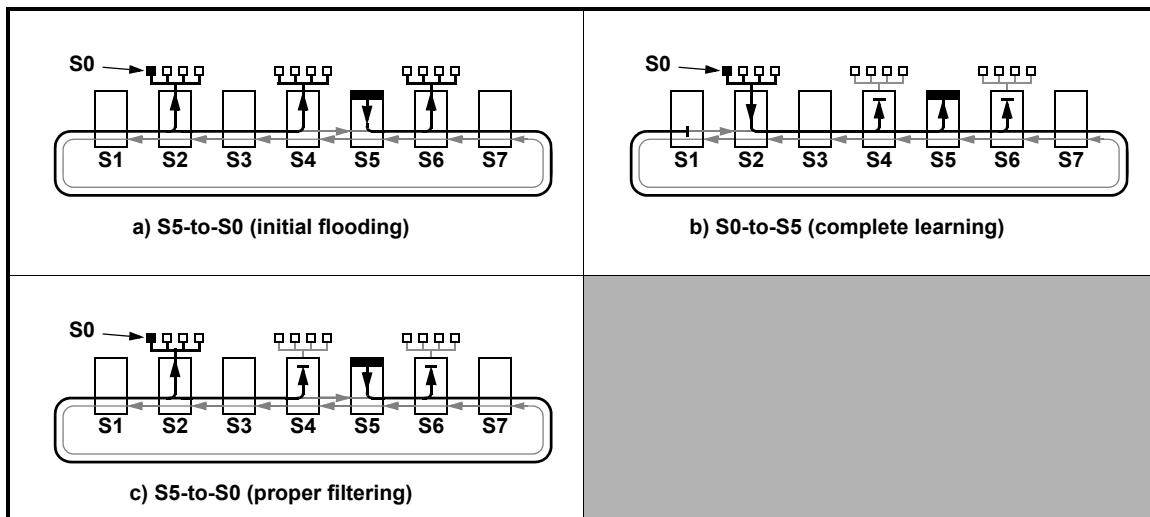


Figure F.7—Correct bridge filtering behavior

F.1.5 RPR support for basic bridging model with spatial reuse for local hosts

In the basic bridging model, all bridged traffic is flooded on the ring. It is also desirable to not have to flood unicast traffic throughout the entire ring in cases where a pair of local end stations are directly transmitting to each other. This allows local end stations to achieve spatial reuse while still interworking with and conforming to the 802 basic bridging model. Traffic originating from a bridge is always flooded on the ring. Traffic originating from a local end station destined to a local end station is not flooded. Traffic destined to a remote end station is always flooded.

As local end stations typically strip frames having a MAC destination address that matches their station address, the flooding indication in the frame is used by local end stations to distinguish between frames that are to be destination stripped and frames that are flooded. Frames having a flooding indication continue on the ring until the flooding completes.

F.1.6 Duplication/misordering prevention

An RPR network may have ring topology changes and protection events that may affect frame duplication and misordering aspects of bridging compliance. Frame duplication issues might occur in scenarios where frames are not properly stripped from the ring. Although the ring employs various mechanisms to prevent frames from circulating on the ring forever, very specific procedures need to be employed to ensure that the same frame is never seen more than once by a station. Frame misordering can occur with changes due to protection. Protection events may cause frames to take an alternative path around the ring. Cases where changing to a protect path or changing from a protect path back to the primary path can lead to frame misordering. Procedures are defined in 7.6.1 to preserve ordering and prevent duplication with changing of network paths during protection and restoration events.

F.1.7 MAC client invocation of optional RPR service parameters

MAC clients invoking optional RPR MA_DATA.request service parameters should maintain 802.1D/Q bridging conformance and 802 bridged network filtering integrity (see F.1.4) by adhering to the following rules:

- a) MAC clients requesting transmission of frames that should be flooded, having mac_protection equal to FALSE, should guarantee delivery to all reachable stations on the ring.
- b) MAC clients requesting transmission of frames using the following optional primitive parameters: flooding_form, destination_address_extended, or source_address_extended, should follow these rules:
 - 1) If the client provided destination_address or destination_address_extended is nonlocal, or the client provided source_address or source_address_extended is not *myMacAddress*, then the flooding_form parameter should be set to ensure the frame will be flooded by the MAC.
 - 2) Otherwise, flooding_form should be set to FI_NONE.

F.1.8 RPR requirements for 802.1D and 802.1Q bridging conformance

The RPR MAC conformance to the IEEE Std 802.1D-2004 and IEEE Std 802.1Q-2005 bridging standards is achieved with the following MAC capabilities:

- a) The MAC provides an Internal Sublayer Service (ISS), when used to interface with the bridging relay entity. The ISS conforms to 6.4 of IEEE Std 802.1D-2004 and 7.1 of IEEE Std 802.1Q-2005, when appropriate.
- b) The MAC can communicate with any bridge protocol entity via the LLC sublayer, in conformance with the bridging specifications.
- c) The MAC service supports a mode that guarantees no reordering with a given user priority for a given combination of MAC destination address and source address to support 802 networks requiring no reordering.³⁶
- d) The MAC service supports a mode that guarantees no duplication with a given user priority for a given combination of MAC destination address and source address to support 802 networks requiring no duplication.³⁷
- e) In order to support the transmission of bridged frames appropriately over an RPR shared medium, the MAC transmission rules conform to Table E.3. This includes handling of local unicast, remote unicast, broadcast, and multicast frames.
- f) In order to allow end stations and bridges to coexist on the ring, the MAC reception rules conform to Table E.4.
- g) The MAC copies all frames having a flooding indication of flood to its bridge relay client, when used to support a bridge relay client.

³⁶The MAC Service (9.2 of ISO/IEC 15802-1) permits a negligible rate of reordering of frames (6.3.3 of ISO/IEC 15802-3 [IEEE Std 802.1D-2004]).

³⁷The MAC Service (9.2 of [ISO/IEC 15802-1]) permits a negligible rate of duplication of frames (6.3.4 of ISO/IEC 15802-3 [IEEE Std 802.1D-2004]).

F.2 Architectural model of an 802.1D compliant RPR bridge

The RPR MAC conforms to the architectural model of a bridge as defined by IEEE Std 802.1D-2004. The component LANs are interconnected by means of MAC bridges; each port of a MAC bridge connects to a single LAN. Figure F.8 illustrates the architecture of such a bridge. The RPR MAC entity and its associated dual-ring interface fits within the context of the port definition of the bridge architecture model.

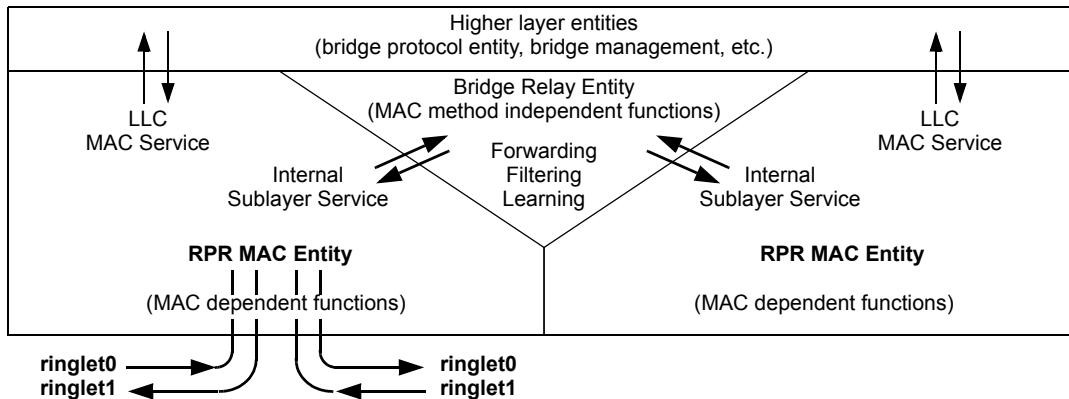


Figure F.8—Bridge architecture model

A bridge consists of the following:

- A bridge relay entity that interconnects the bridge's ports.
- At least two ports.
- Higher layer entities, including at least a bridge protocol entity.

F.2.1 Bridge relay entity

The bridge relay entity handles the MAC method independent functions of relaying frames among bridge ports, filtering frames, and learning filtering information. It uses the Internal Sublayer Service provided by the separate MAC entities for each port. Frames are relayed between ports attached to different LANs.

F.2.2 Ports

Each bridge port transmits and receives frames to and from the LAN to which it is attached. An individual MAC entity permanently associated with the port provides the Internal Sublayer Service used for frame transmission and reception. The RPR MAC handles all MAC method dependent functions (MAC protocol and procedures).

F.2.3 Higher layer entities

The bridge protocol entity handles calculation and configuration of bridged LAN topology.

The bridge protocol entity and other higher layer protocol users, such as bridge management (see 7.1.3 of IEEE Std 802.1D-2004 and GARP application entities including GARP participants (see Clause 12 of IEEE Std 802.1D-2004)), make use of logical link control procedures. These procedures are provided separately for each port, and they use the MAC service provided by the individual MAC entities.

F.3 RPR MAC Internal Sublayer Service

The ISS and EISS service interfaces are provided by the RPR MAC entity when supporting a bridge relay entity. The RPR MAC support of the ISS is defined in IEEE Std 802.1D-2004. The RPR MAC support of the EISS is defined in IEEE Std 802.1Q-2005. A summary of the RPR MAC support of ISS and EISS and associated frame format mappings is included here for informative purposes.

F.3.1 RPR MAC support of Internal Sublayer Service

Support of the ISS by the RPR MAC is specified by IEEE Std 802.17a-2004. Figure F.9 and Figure F.10 illustrate the mapping of ISS service primitives onto the RPR frame structure.

Figure F.9 shows the mapping of the M_UNITDATA.request and M_UNITDATA.indication primitive parameters to the RPR MAC frame fields for frames that are transmitted by the bridge LLC or a host. The mapping in Figure F.9 arises whenever the bridge LLC or host transmits a frame where the source address parameter equals *myMacAddress*. The frame is transmitted using basic data frame format. These frames may be received by either a host or a bridge giving rise to the M_UNITDATA.indication primitive.

Figure F.10 shows the mapping of the M_UNITDATA.request and M_UNITDATA.indication primitive parameters to the RPR MAC frame fields for frames that are transmitted by a bridge that do not originate from the bridge LLC. The mapping in Figure F.10 arises whenever a host or a bridge transmits a frame where the source address parameter does not equal *myMacAddress*. The frame is transmitted using the extended data frame format. These frames may be received by either a host or a bridge giving rise to the M_UNITDATA.indication primitive.

F.3.2 RPR MAC support of Enhanced Internal Sublayer Service

An Enhanced Internal Sublayer Service (EISS) is derived from the Internal Sublayer Service (ISS, see 6.4 of IEEE Std 802.1D-2004) by augmenting that specification with elements necessary for the tagging and untagging functions in a MAC bridge. The EISS provided by the MAC conforms to 7.1 of IEEE Std 802.1Q-2005.

On receipt of a MAC frame by receive medium access management, the frame is passed to the reconciliation sublayer, which disassembles the MAC frame into parameters, as specified below, that are supplied with the MA_UNITDATA.indication primitive. In addition to the parameters specified by the Internal Sublayer Service, the EISS supports the routing information parameter.

The rif_info parameter is the routing information field parameter specified by the EISS in 7.1.1 of IEEE Std 802.1Q-2005. It is passed transparently by the RPR MAC.

Figure F.11 shows the mapping of the EM_UNITDATA.request and EM_UNITDATA.indication primitive parameters to the MAC frame fields for frames that are transmitted by the bridge LLC or a host. The mapping in Figure F.11 arises whenever the bridge LLC or host transmits a frame where the source address parameter equals *myMacAddress*. The frame is transmitted using basic data frame format. These frames may be received by either a bridge or a host giving rise to the EM_UNITDATA.indication primitive.

Figure F.12 shows the mapping of the EM_UNITDATA.request and EM_UNITDATA.indication primitive parameters to the MAC frame fields for frames that are transmitted by a bridge that do not originate from the bridge LLC. The mapping in Figure F.12 arises whenever a host or a bridge transmits a frame where the source address parameter does not equal *myMacAddress*. The frame is transmitted using extended data frame format. These frames may be received by either a bridge or a host giving rise to the EM_UNITDATA.indication primitive.

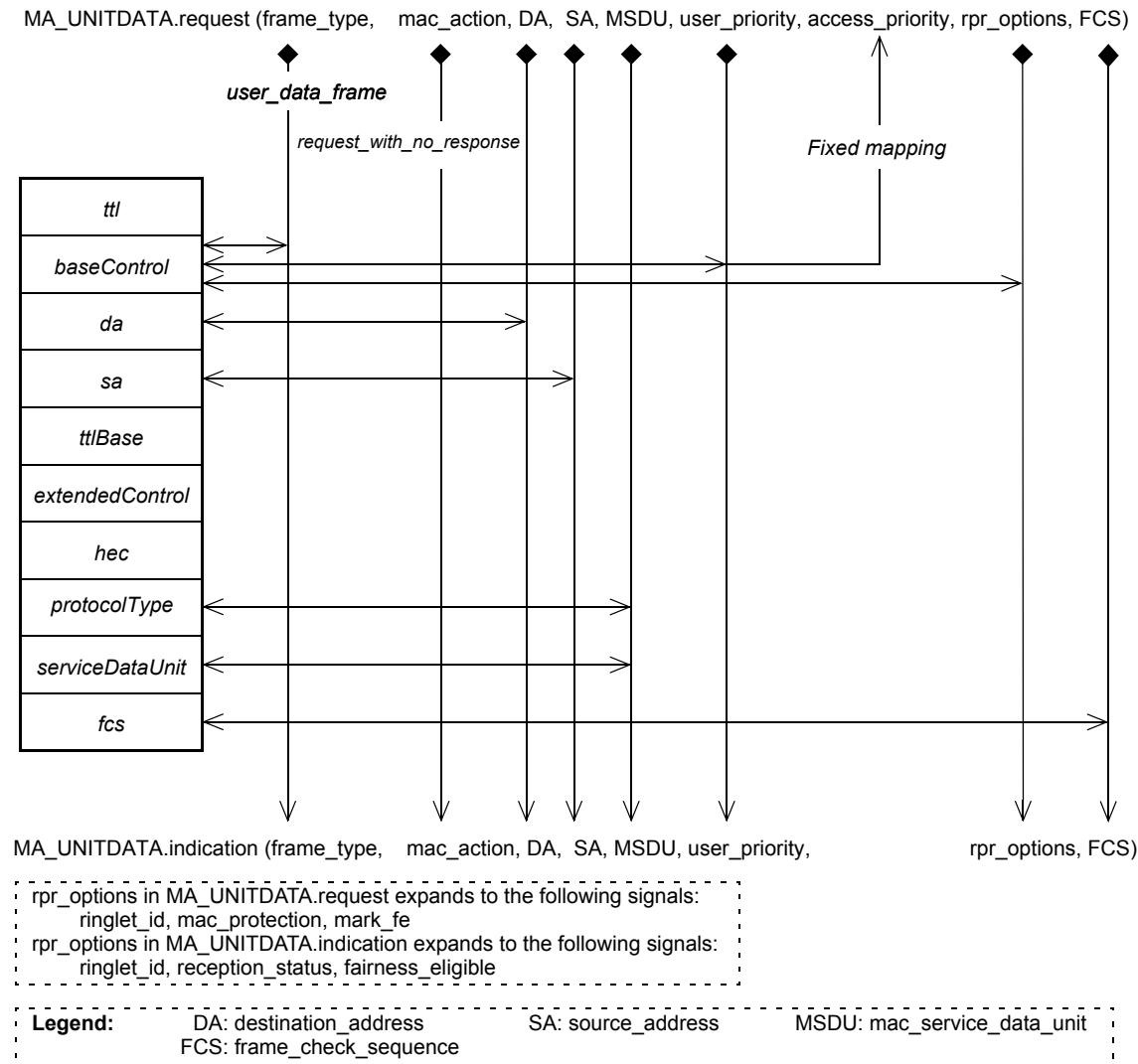


Figure F.9—Service primitive mapping for frames from a bridge LLC or host

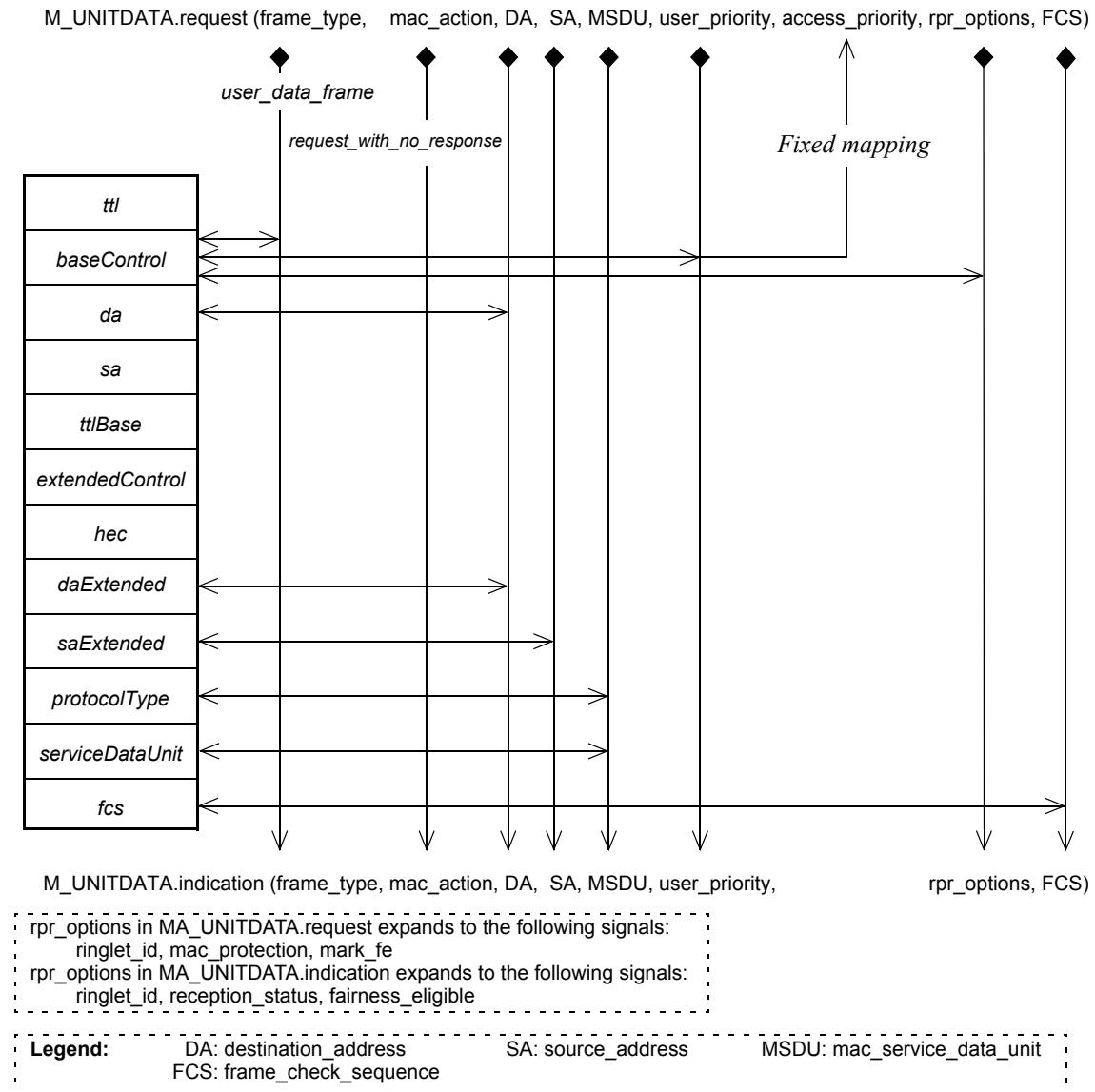


Figure F.10—Service primitive mapping for non-LLC frames from a bridge

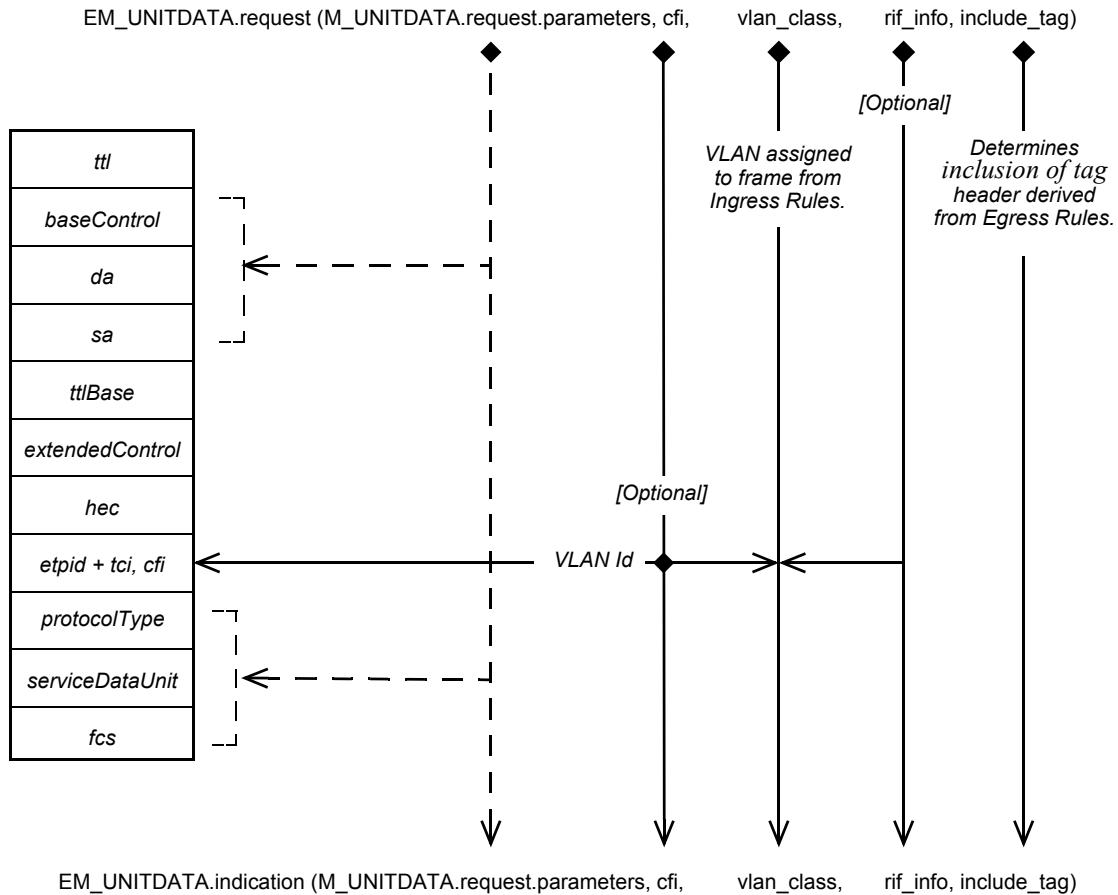


Figure F.11—Enhanced service primitive mapping for frames from a bridge LLC or host

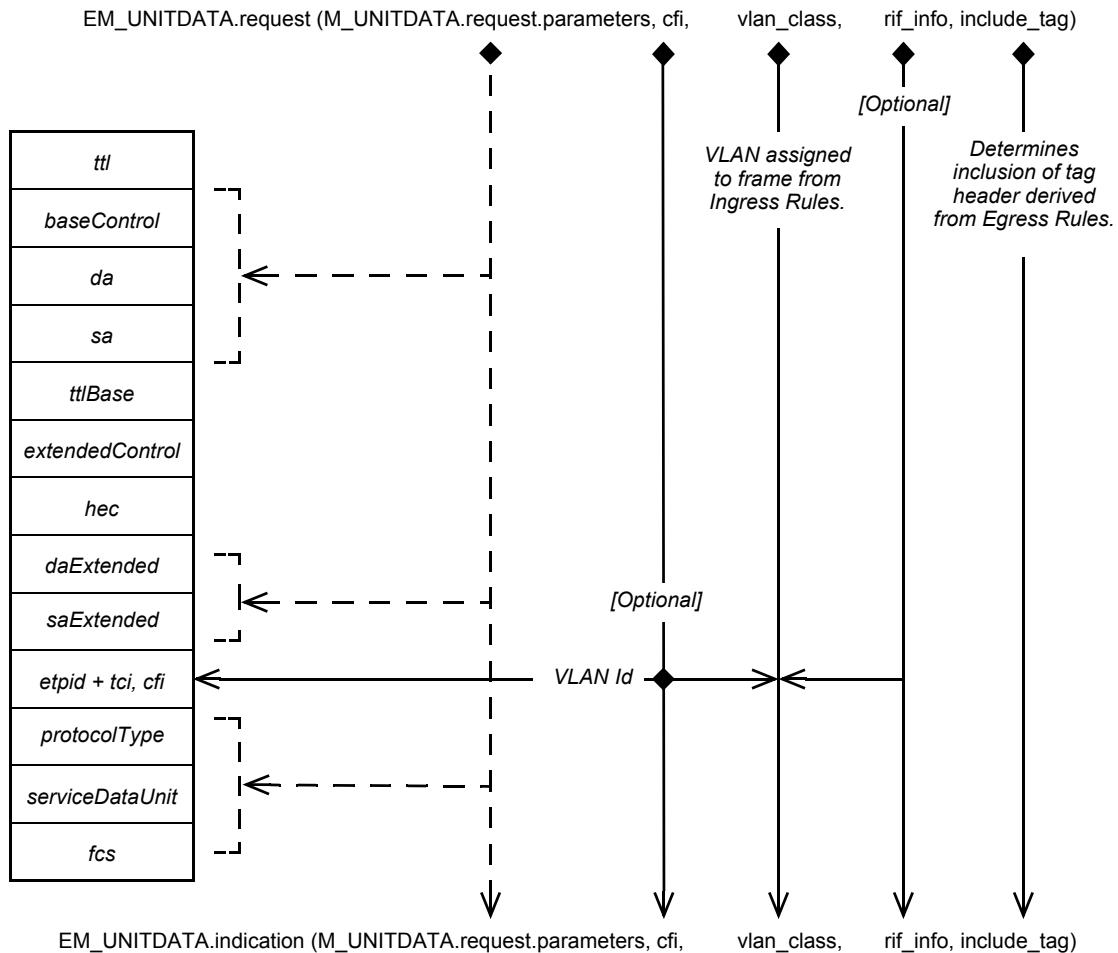


Figure F.12—Enhanced service primitive mapping for non-LLC frames from a bridge

F.4 Bridge protocol entity interactions

The RPR MAC provides a MAC sublayer that conforms to 2.2 of IEEE Std 802.2-1998.

F.5 MAC client transmission requirements

Bridge relayed frames are submitted for transmission by the bridging forwarding process. The service request primitive associated with such a frame conveys the values of the source and destination address fields received in the Service indication primitive. Refer to Figure F.9, Figure F.10, Figure F.11, and Figure F.12 for the mappings.

The RPR MAC client transmission requirements depend on whether the frame is local only, originates from a remote end station, or is destined to a remote end station. There are a common set of MAC transmission requirements consistent with the frame transmission requirements for frames originating from either bridge or host type clients. The MAC sets the flooding indication in the RPR frame based on whether the frame is to be flooded or transmitted locally on the ring. The MAC also determines whether the frame is to be transmitted using basic data frame or extended data frame format. The flooding indication is not set for control, fairness, and idle frames.

More detailed description of general transmission processing including transmission of control, fairness, and idle frames is defined in 7.7. Table F.1 summarizes data frame transmission requirements for bridging compliance, by defining the mapping of the MA.unidata.request parameters into the associated frame header and payload fields. Within this context, the variable *myMacAddress* represents the receiving station's RPR MAC address as reported through topology discovery.

Table F.1—Transmit rule requirements

Conditions		Row	Frame fields				
MA_UNITDATA.request			frame header		frame payload		
destination_address	source_address		<i>fi</i>	<i>ef</i>	<i>daExtended</i>	<i>saExtended</i>	
—	! <i>myMacAddress</i>	1	!FI_NONE	1	destination_address	source_address	
Local(destination_address)	<i>myMacAddress</i>	2	FI_NONE	0	-na-		
!Local(destination_address)	<i>myMacAddress</i>	3	!FI_NONE				

```
#define Local(macAddress) (TopologyDataBaseLocation(macAddress) != NULL).
```

NOTE 1—*frame.da* is always set to the destination_address parameter.
 NOTE 2—*frame.sa* is always set to *myMacAddress*.

NOTE—Insertion of *myMacAddress* in the *frame.sa* header field ensures stripping of frames that return to their source (source stripping). In addition, the MA_UNITDATA.request destination_address parameter is copied into the *frame.da* field.

Row F.1-1: Remote-source frames are flooded using the extended frame format.

Row F.1-2: Local unicast frames are transmitted using the basic frame format and are not flooded.

Row F.1-3: Remote unicast, broadcast, and multicast frames are flooded using the basic frame format.

F.6 MAC client reception requirements

Received frames are processed by the MAC, which performs the copy/strip rules and determines whether received frames should be passed to the LLC or bridge relay entities. Table F.2 defines the reception requirements in order to maintain 802 bridging compliance on an RPR network. Included in the table are the reception requirements for host and bridge client types, which differ only in how they handle flooded unicast traffic. Host clients receive all frames directed to the host's MAC address, broadcast, and multicast frames. Bridge relay clients receive all flooded unicast frames, in addition to broadcast, multicast, and frames directed to the LLC entity's MAC address (see 7.12.7 and Figure 7-9, of IEEE Std 802.1D-2004).

The MAC accommodates the host and bridge reception requirements via the MAC receive rules and the *myRxFilter* configuration parameter. The *myRxFilter* configuration parameter handles the differences in bridge and host reception requirements. Within this context, the variable *myMacAddress* in Table F.2 represents the receiving station's RPR MAC address as reported through topology discovery. More detailed description of MAC reception processing including reception of control and fairness frames is defined in 7.6.

Table F.2—Combined host/bridge receive rule requirements

Conditions				Row	Results		
clientType	Frame fields				Receive actions		
	<i>fi</i>	<i>da</i>	<i>sa</i>		copy to Client	strip	
—	—	—	MyMac(sa)	1	!COPY	STRIP	
—	FI_NONE	MyMac(da)	!MyMac(sa)	2	COPY	STRIP	
—	FI_NONE	!MyMac(da) && Unicast(da)	!MyMac(sa)	3	!COPY	!STRIP	
—	FI_NONE	!Unicast(da)	!MyMac(sa)	4	COPY	!STRIP	
—	!FI_NONE	MyMac(da)	!MyMac(sa)	5	COPY	!STRIP	
—	!FI_NONE	!Unicast(da)	!MyMac(sa)	6	COPY	!STRIP	
HOST	!FI_NONE	!MyMac(da) && Unicast(da)	!MyMac(sa)	7	!COPY	!STRIP	
BRIDGE	!FI_NONE	!MyMac(da) && Unicast(da)	!MyMac(sa)	8	COPY	!STRIP	

```
#define MyMac(macAddress) (macAddress == myMacAddress).
#define Unicast (address) ((address >> 40 & 1) == 0).
```

Row F.2-1: Self-sourced frames are never copied to the client and always source stripped.

Row F.2-2: Non-flooded frames destined to this destination address are copied and destination stripped.

Row F.2-3: Non-flooded unicast frames destined to another destination address are not copied or stripped.

Row F.2-4: A non-flooded multicast frame is copied, but not stripped. Such frames are normally source stripped or stripped when their *ttl* (time to live) reaches zero. Although non-flooded multicast frames on the RPR are never generated (per transmit rules), client reception of such frames is well defined.

Row F.2-5: Flooded frames destined to this destination address are copied, but not stripped. Such frames are normally source stripped or stripped when their *ttl* (time to live) reaches zero.

Row F.2-6: Flooded multicast frames are copied, but not stripped. Such frames are normally source stripped or stripped when their *ttl* (time to live) reaches zero.

Row F.2-7: Flooded unicast frames destined to another destination address are not copied to a host client or stripped. Such frames are normally source stripped or stripped when their *ttl* (time to live) reaches zero.

Row F.2-8: Flooded unicast frames destined to another destination address are copied to a bridge client, but not stripped. Such frames are normally source stripped or stripped when their *ttl* (time to live) reaches zero.

Annex G

(informative)

Implementation guidelines

G.1 Sizing of secondary transit queue and *addRateA1*

The size of the secondary transit queue in a dual queue implementation is associated with how much subclassA1 traffic can be supported by that implementation. This relationship is useful to consider when creating the implementation, and it is necessary to consider when operating the implementation.

G.1.1 Calculation of minimum size for secondary transit queue

During the design of an implementation, the designer should consider the maximum sized ring into which the implementation will be deployed and the amount of subclassA1 traffic that is desired to be supported. Given these, and the worst-case assumptions that all incoming transit traffic is either classB or classC, and that the local add traffic is subclassA1, classB, and classC, a recommended sizing of the secondary transit queue (STQ) can be estimated by Equation (G.1), where all units (directly or indirectly used) are converted to bytes, bytes/second, or seconds, as appropriate for each value:

$$\text{sizeStq} \geq \text{stqHighThreshold} + ((\text{addRateA1} + \text{addRateB}) * \text{responseTime}) \quad (\text{G.1})$$

responseTime represents the maximum amount of time from becoming congested until congestion causing traffic stops being received. The value of *responseTime* is given by Equation (G.2):

$$\text{responseTime} = \text{fairnessAgingTime} + \text{maxFRTT} + \text{queueDrainTime} \quad (\text{G.2})$$

fairnessAgingTime represents the maximum amount of time that could be incurred waiting for the fairness algorithm to detect the level of congestion. This occurs due to the low-pass filtering of the fairness variables, and therefore, it depends upon the values used for *agingInterval* and *lpCoef*.

maxFRTT (maximum fairness round-trip time) represents the maximum fairness round-trip time of a congestion domain for which this station could be the congestion domain head. It is assumed to be the round-trip time for propagation of a fairness value around an entire ring and for the first effected traffic to return. The value of *maxFRTT* can be estimated by Equation (G.3), where *ringKm* is the length (or circumference) of the ring in kilometers (and used in the equation without any units):

$$\text{maxFRTT} = \text{numStations} * \text{advertisingInterval} + 2 * (5 \mu\text{s} * \text{ringKm}) \quad (\text{G.3})$$

NOTE—This calculation for *maxFRTT* both overestimates and possibly underestimates the true FRTT. It overestimates the true value because it assumes that the congestion domain extends for the entire ring and that each per hop delay will last for the maximum amount of time (*advertisingInterval*). It either underestimates or overestimates the true value depending upon whether the implementation's value of *advertisingInterval* is higher or lower than the average value used by the other stations in the congestion domain. The implementer should take these into account when using the above equation.

queueDrainTime represents the maximum amount of time that could be incurred waiting for the upstream STQs to be drained. It is assumed that all upstream STQs are filled up completely (to their *stqFullThreshold*) at the moment they receive the new fairness message, and therefore, they have to drain *stqFullThreshold* worth of bytes before they fully back off to the amount requested by the fairness message. The value of *queueDrainTime* can be estimated by Equation (G.4):

$$\text{queueDrainTime} = (\text{numStations} * \text{stqFullThreshold}) / \text{unreservedRate} \quad (\text{G.4})$$

The STQ sizing is being estimated such that it is large enough to hold as much traffic as could be received while adding local traffic and while waiting for upstream stations to back off in response to a congestion message from the local station.

In Equation (G.1), the *responseTime* is measured from when the STQ fill level reaches *stqLowThreshold*, because this is the definition of congestion. However, the MAC may continue to add local fairness eligible traffic until the STQ fill level reaches *stqHighThreshold*; hence, the size difference between *stqHighThreshold* and *stqLowThreshold* must also be added, resulting in *stqHighThreshold* being the first term in Equation (G.1).

The second term in Equation (G.1), $(\text{addRateA1} + \text{addRateB}) * \text{responseTime}$, is the maximum amount of subclassA1 and classB traffic that the local station is expected to add during the time it takes to signal congestion to the farthest upstream station and start receiving the reduced amount of fairness eligible traffic from said station. The time, *fairnessAgingTime* + *maxFRTT* + *queueDrainTime*, is the maximum amount of time for detecting congestion, sending a fairness message, receiving the first fairness affected byte back from the farthest away station, and the maximum amount of time that all upstream stations could be draining already queued traffic.

G.1.2 Calculation of maximum *addRateA1*

Given an existing implementation, with a known STQ size, the implementation can estimate the amount of subclassA1 traffic that can be supported by solving Equation (G.1) to give Equation (G.5):

$$\text{addRateA1} \leq ((\text{sizeStq} - \text{stqHighThreshold}) / \text{responseTime}) - \text{addRateB} \quad (\text{G.5})$$

G.2 ClassA shaping effects on jitter

G.2.1 ClassA shaper characteristics

This subclause describes the behavior of classA traffic shapers, which are intended to provide guaranteed transmissions rate and bounded jitter for classA frames. For illustration purposes, a 10-station topology with four major classA traffic consumers is considered, as illustrated in Figure G.1.

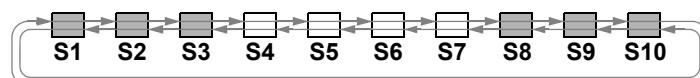


Figure G.1—Provisioned classA bandwidth scenario

In this scenario, stations {S4, S5, S6, S7} are configured to each transmit at 20% of the link bandwidth. All other stations are allocated an insignificant amount of classA bandwidth.

G.2.2 ClassA shaper behaviors

Each shaper restricts transmissions based on accumulated time varying credits, such as those illustrated in Figure G.2.

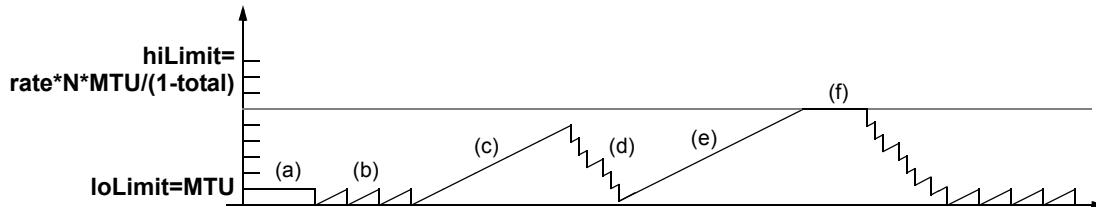


Figure G.2—ClassA credits fluctuations

The types of credit adjustments depend on the transmission properties of the station, as listed below:

- a) When transmissions are not ready, the credits are clamped to a $loLimit=MTU$ value. (The intent is to limit jitter due to bursts of accumulated credits.)
- b) On an unblocked station, credits decrease by the frame size during transmissions and increase thereafter. The upward slope is proportional to the allocated classA transmission rate.
- c) Transmissions are blocked when $credits < loLimit$.
- d) Transmissions are unblocked when $credits \geq loLimit$.
- e) When transmissions are ready but blocked, credits continue to increase at the upward slope described in step b). These positive credits account for missed opportunities.
- f) An unblocked station sends high-rate transmissions until excess credits are consumed. The intent is to maintain a constant transmission rate, over the blocked and unblocked intervals.
- g) Overprovisioned classA bandwidths can cause stations to be blocked indefinitely, allowing credits to accumulate beyond normally expected levels.
- h) Excess credits are clamped to a $hiLimit$ value, to recover from abnormal conditions (credits would not normally reach the $hiLimit$ value, if classA traffic is properly provisioned).

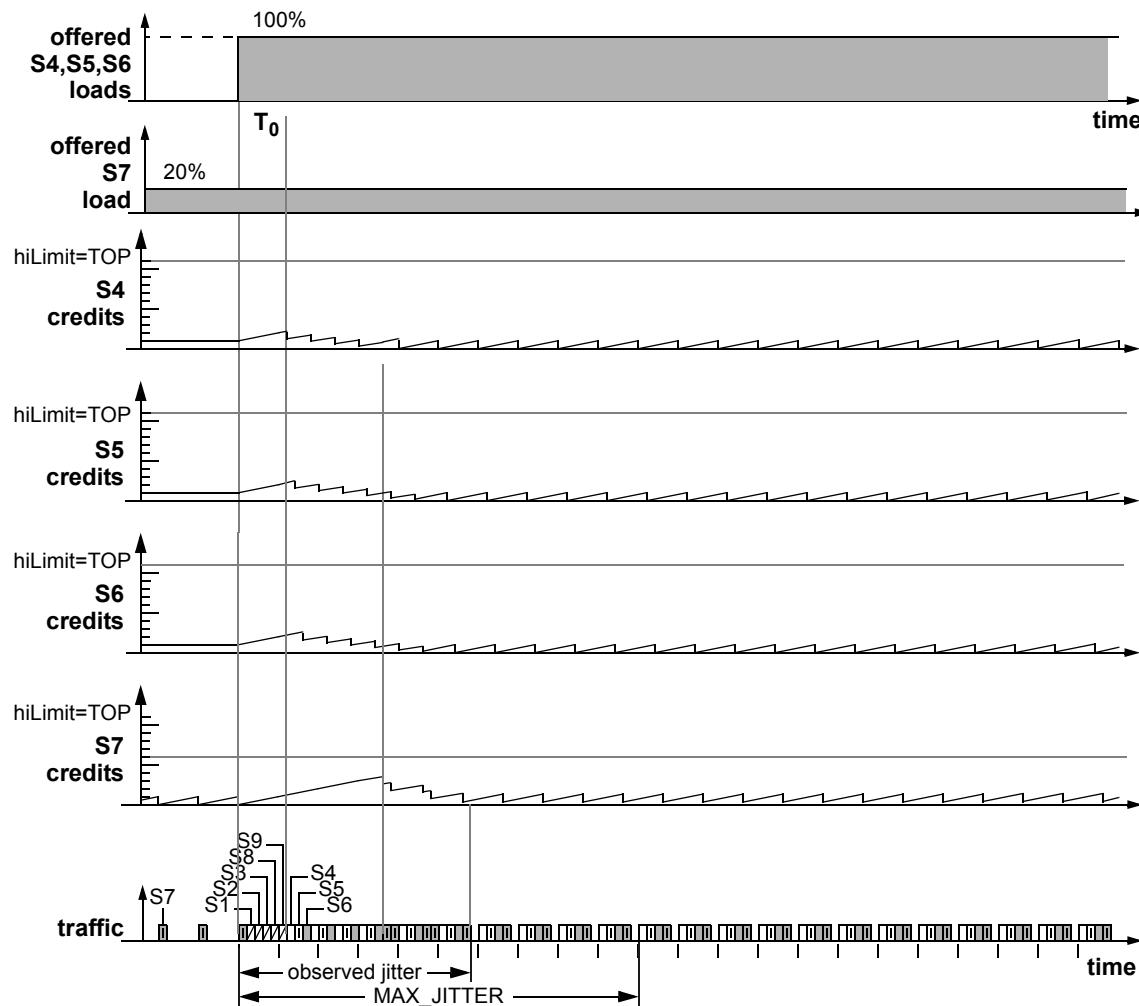
G.2.3 ClassA clamped-credit shaper behaviors

The clamped-credit shaper, illustrated in this subclause, clamps its transmission credits to the $loLimit$ value when nothing is ready to be sent. The intent is to enforce fine-grained jitter and bandwidth guarantees, rather than optimizing the shapers for bursting traffic patterns.

The behaviors of classA shapers, in the presence of large transient offered load, are illustrated in Figure G.3. In this example, stations {S4, S5, S6} initially provide no offered load, although each are allocated 20% of the transmission bandwidth. At time T_0 , clients in stations {S4, S5, S6} each supply the allocated bandwidth.

For this scenario and the lost-credits model, the following properties are observed:

- a) The maximum S10 transmission delays remain within the guaranteed MAX_JITTER bound.
- b) The $hiLimit$ value is never reached, preserving the {S4, S5, S6, S7} guaranteed transmissions rates.



```
#define MAX_JITTER ((numberOfStations*MTU)/(1-totalRate))// 50 MTU in this example
#define TOP (MTU+myRate*MAX_JITTER)// 11 MTU in this example
```

Figure G.3—ClassA clamped-credit shaper behaviors

Annex H

(informative)

C-code illustrations of CRC computations

This annex provides code examples that illustrate the computation of the CRCs. The code in this annex is purely for informational purposes, and it should not be construed as mandating any particular implementation. In the event of a conflict between the contents of this annex and a normative portion of this standard, the normative portion shall take precedence.

The syntax used for the following code examples conforms to ISO/IEC 9899:1999.

```
// **** * Commenting ****
// **** * The "://" commenting convention will be used.
// **** * An in-line comment will be properly indented.
// **** * A comment may follow the statement line:
//
// Sample code is preceded by "| ", as shown below
// | // Checking for exception conditions
// | if (x > 0)          // Warning processing code
// |     goto Label;
```

```
// **** Integer Type Definitions ****
// ***** Assuming standard C definitions *****

// typedef unsigned char uint8_t;           // assuming 'char' is an 8-bit value
// typedef unsigned short uint16_t;          // assuming 'short' is a 16-bit value
// typedef unsigned long uint32_t;           // assuming 'long' is a 32-bit value
// typedef unsigned long long uint64_t;       // assuming 'long long' is a 64-bit value
// typedef signed char int8_t;              // assuming 'char' is an 8-bit value
// typedef signed long int16_t;             // assuming 'short' is a 16-bit value
// typedef signed long int32_t;             // assuming 'long' is a 32-bit value
// typedef signed long long int64_t;         // assuming 'long long' is a 64-bit value
```

```

// ****
// ***** Indenting ****
// ***** The code is formatted using indent (GNU version 1.2) with the following flags set:
// indent -bap -bl -bli0 -bls -c50 -cbi0 -cd35 -cil -cli0 -di4 -hnl -i4 -ip0 -l132 -lc132 -nbad -ncdb -ncs -nfc1 -nlp -npcs \
// -nut -sc -ss
//
// -bap    force a blank line after procedure bodies
// -bl     put braces on line after if, etc.
// -bli0   insert braces 0 spaces
// -bls    put braces after structure declaration lines
// -c50    code comments start at line position 50
// -cbi0   insert braces after a case label, 0 spaces
// -cd35   declaration comments start at line position 35
// -cil    continuation indent, 1 space
// -cli0   case label indent of 0 spaces
// -di4    put identifiers in second available position after type
// -hnl    prefer to break long lines at position of input new lines
// -i4     indentation level is 4 spaces
// -ip0    indent parameters in old-style function definitions, 0 spaces
// -l132   maximum length of a line is 132 characters
// -lc132  the maximum line length for comments
// -nbad   no blank line is forced after a declaration
// -ncdb   no comment delimiters on blank lines
// -ncs    no space after a cast operator
// -nfc1   do not format comments in the first column as normal
// -nlp    do not line up continued lines at parentheses
// -npcs   no space after a procedure name and '('
// -nut    do not use tabs
// -sc     put '*' at left of comments
// -ss     for 1-line for and while statement, force a blank before the semicolon

// ****
// ***** Compiler dependencies ****
// ***** The GNU C compiler defines assert() properly, most C compilers do not.
// ***** These assert() definitions are provided for non GNU C compilers.

#include <stdio.h>
#ifndef __GNUC__
#include <assert.h>
#else
#ifndef NDEBUG
#define assert(ex) ((void)0)
#else
#define assert(ex) ((void)((ex) ? 0 : ((void)fprintf(stderr, \
"Assertion failed: file \"%s\", line %d\n", __FILE__, __LINE__), \
((void)abort(), 0))))
#endif
#endif

```

676

```
// ****CRC-32 And CRC-16 Generation Code ****
// ****
uint8_t dataBytes[16384];

void PrintTable(int, int);
int ValidateCrc32(int, uint8_t *, int);
uint32_t GenerateCrc32(int, uint8_t *, int);
uint32_t CalculateCrc32(int, uint8_t *, int);
uint32_t CrcByte32(uint32_t, uint8_t);

uint16_t GenerateCrc16(int, uint8_t *, int);
uint16_t CalculateCrc16(int, uint8_t *, int);
uint16_t CrcByte16(uint16_t, uint8_t);

uint32_t CrcBits(uint32_t, uint32_t, int);
uint32_t BitReverse(uint32_t);
void Error(char *);
void CrcProduce(char *, int);

#define MSB8    ((unsigned)1<<7)
#define MSB16   ((unsigned)1<<15)
#define MSB32   ((unsigned)1<<31)
#define ONES16  0xFFFF
#define ONES32  0xFFFFFFFF
#define CRC16_COMPUTE ((uint16_t)0X1021)
#define CRC32_COMPUTE ((uint32_t)0X04C11DB7)
#define CRC_RESULTS   ((uint32_t)0XC704DD7B)
#define DEBUG 1
```

```

int
main(int argc, char **argv)
{
    int i, check;
    int size = 32, reverse = 1, table = 1;
    int setRev = 0, setHow = 0;
    char *argPtr, *fileName;

    // Command line specifies number of bits computed in parallel
    for (i = 1; i < argc; i += 1)
    {
        argPtr = argv[i];
        if (*argPtr != '-')
            Error("Illegal argument, use: -n -r -tdd -c");
        argPtr += 1;
        switch (*argPtr)
        {
        case 'n':
        case 'r':
            if (setRev)
                Error("Mutually exclusive options: -n -r");
            reverse = (*argPtr == 'r');
            setRev = 1;
            break;
        case 't':
            if (setHow)
                Error("Mutually exclusive options: -tdd -c");
            size = atoi(argPtr + 1);
            if (size != 1 && size != 2 && size != 4 && size != 8 && size != 16 && size != 32)
                Error("Incorrect width; -t1 -t2 -t4 -t8 -t16 or -t32\n");
            table = 1;
            break;
        case 'c':
            if (setHow)
                Error("Mutually exclusive options: -wdd -c");
            table = 0;
            setHow = 1;
            break;
        case 'f':
            size = atoi(argPtr + 1);
            if (size != 16 && size != 32)
                Error("Incorrect CRC width; -t16 or -t32\n");
            check = size;
            fileName = argv[i + 1];
            if (i >= argc || *fileName == '-')
                Error("Incorrect file specification\n");
            break;
        }
    }
}

```

```
678     default:
        Error("Arguments: -n -r -t[1,2,4,8,16,32] -c\n");
        break;
    }
    if (check == 16 || check == 32)
    {
        printf("check %d, file= %s\n", check, fileName);
        CrcProduce(fileName, check);
        exit(0);
    }
    assert(table);
    PrintTable(reverse, size);
    return (0);
}
```

```

void
CrcProduce(char *fileName, int size)
{
    int i;
    char *dataPtr, dataBuf[256];
    uint32_t crcSum;
    FILE *filePtr;

    filePtr = fopen(fileName, "r");
    if (filePtr == NULL)
        printf("Could not open %s\n", fileName);
    if (DEBUG)
        printf("Opened %s\n", fileName);
    for (i = 0; i < sizeof(dataBytes); i += 1)
    {
        if ((dataPtr = fgets(dataBuf, sizeof(dataBuf), filePtr)) == NULL)
            break;
        dataBytes[i] = strtol(dataPtr, NULL, 16);
        if (DEBUG)
            printf("Line %3d is %02x\n", i, dataBytes[i]);
    }
    if (i == 0)
        Error("Input file is empty\n");
    if (i == sizeof(dataBytes))
        Error("Input file too long\n");
    if (DEBUG)
        printf("Calling GenerateCrc32(size=%d)\n", i + 4);
    switch (size)
    {
    case 16:
        crcSum = GenerateCrc16(1, dataBytes, i + 2);
        printf("crcSum = %04x\n", crcSum);
        break;
    case 32:
        crcSum = GenerateCrc32(1, dataBytes, i + 4);
        printf("crcSum = %08x\n", crcSum);
        break;
    }
}

void
Error(char *string)
{
    printf(string);
    exit(1);
}

```

```
680 char keys[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'j', 'k', 'm', 'n', 'p', 'r', 's', 't' };
void PrintTable(int reverse, int size)
{
    uint32_t last, next, select, mask, sum;
    int i, j, numb;
    for (i = 0; i < 32; i += 1)
    {
        // Calculate contributing-input values
        select = 1 << (31 - i);
        mask = reverse ? BitReverse(select) : select;
        printf("c%02d= ", i);
        for (j = sum = 0; j < 32; j += 1)
        {
            last = 1 << (31 - j);
            next = CrcBits(last, (uint32_t) 0, size);
            if (next & mask)
                sum |= last;
        }
        for (j = 0; j < 32; j += 1)
        {
            select = 1 << (31 - j);
            mask = reverse ? BitReverse(select) : select;
            numb = j < 16 ? keys[j] : keys[j - 16] + 'A' - 'a';
            if ((sum & mask) != 0)
            {
                sum &= ~mask;
                printf("%c", numb);
                if (j != 31)
                    printf(sum != 0 ? "^" : " ");
            }
            else
            {
                printf(j != 31 ? " " : " ");
            }
            printf(";\n");
        }
    }
}

// Generate CRC32 for payload with "sizeInBytes" bytes (including the CRC)
uint32_t
GenerateCrc32(int reverse, uint8_t * inputs, int sizeInBytes)
{
    uint32_t crcSum;

    assert(sizeInBytes >= 5);
    crcSum = CalculateCrc32(reverse, inputs, sizeInBytes - 4);
    return (crcSum);
}
```

```
// Validate CRC32 for payload with "sizeInBytes" bytes (including the CRC)
int
ValidateCrc32(int reverse, uint8_t * inputs, int sizeInBytes)
{
    uint32_t crcSum, check;
    int i;

    assert(sizeInBytes >= 1);
    crcSum = CalculateCrc32(reverse, inputs, sizeInBytes - 4);
    for (check = i = 0; i < 4; i += 1)
        check = (check << 8) | inputs[sizeInBytes + i - 4];
    return (crcSum != check);
}

// The CalculateCrc32() function points to protected values,
// it checks these values and return a final 32-bit result
uint32_t
CalculateCrc32(int reverse, uint8_t * inputs, int sizeInBytes)
{
    uint32_t crcSum, sum, final;
    uint8_t data;
    int i;

    // The crcSum value is initialized to all ones
    crcSum = (uint32_t) 0xFFFFFFFF;

    // Process each uint8_t covered by the CRC value
    for (i = 0; i < sizeInBytes; i += 1)
    {
        data = reverse ? BitReverse(inputs[i]) : inputs[i];
        crcSum = CrcByte32(crcSum, data);
    }
    sum = reverse ? BitReverse(crcSum) : crcSum;
    final = ~sum & 0xFFFFFFFF;
    return (final);
}
```

```
682     uint32_t
CrcByte32(uint32_t last, uint8_t input)
{
    uint32_t crcSum, newMask;
    int i, oldBit, newBit, sumBit;

    // Process each of the bits within the input uint32_t value
    crcSum = last;
    for (newMask = MSB8; newMask != 0; newMask >>= 1)
    {
        newBit = ((input & newMask) != 0);           // The next input bit
        oldBit = ((crcSum & MSB32) != 0);           // and MSB of crcSum
        sumBit = oldBit ^ newBit;                     // are EXOR'd together
        // Left-shift crcSum and exclusive-OR the newBit-selected values
        crcSum = ((crcSum << 1) & ONES32) ^ (sumBit ? CRC32_COMPUTE : 0);
    }
    return (crcSum);
}

// Generate CRC16 for header with "sizeInBytes" bytes (including the CRC)
uint16_t
GenerateCrc16(int reverse, uint8_t * inputs, int sizeInBytes)
{
    uint32_t crcSum;

    assert(sizeInBytes >= 5);
    crcSum = CalculateCrc16(reverse, inputs, sizeInBytes - 2);
    return (crcSum);
}

// Validate CRC16 for payload with "sizeInBytes" bytes (including the CRC)
int
ValidateCrc16(int reverse, uint8_t * inputs, int sizeInBytes)
{
    uint16_t crcSum, check;
    int i;

    assert(sizeInBytes >= 1);
    crcSum = CalculateCrc32(reverse, inputs, sizeInBytes - 2);
    for (check = i = 0; i < 2; i += 1)
        check = (check << 8) | inputs[sizeInBytes + i - 2];
    return (crcSum != check);
}
```

```

// The CalculateCrc16() function points to protected values,
// it checks these values and return a final 16-bit result
uint16_t
CalculateCrc16(int reverse, uint8_t * inputs, int sizeInBytes)
{
    uint16_t crcSum, sum, final;
    uint8_t data;
    int i;

    // The crcSum value is initialized to all ones
    crcSum = (uint16_t) 0xFFFF;

    // Process each uint8_t covered by the CRC value
    for (i = 0; i < sizeInBytes; i += 1)
    {
        data = reverse ? BitReverse(inputs[i]) : inputs[i];
        crcSum = CrcByte16(crcSum, data);
    }
    sum = reverse ? BitReverse(crcSum) : crcSum;
    final = ~sum & 0xFFFF;
    return (final);
}

uint16_t
CrcByte16(uint16_t last, uint8_t input)
{
    uint16_t crcSum, newMask;
    int i, oldBit, newBit, sumBit;

    // Process each of the bits within the input uint32_t value
    crcSum = last;
    for (newMask = MSB8; newMask != 0; newMask >>= 1)
    {
        newBit = ((input & newMask) != 0);           // The next input bit
        oldBit = ((crcSum & MSB16) != 0);           // and MSB of crcSum
        sumBit = oldBit ^ newBit;                     // are EXOR'd together
        // Left-shift crcSum and exclusive-OR the newBit-selected values
        crcSum = ((crcSum << 1) & ONES16) ^ (sumBit ? CRC16_COMPUTE : 0);
    }
    return (crcSum);
}

uint32_t
CrcBits(uint32_t last, uint32_t input, int size)
{
    uint32_t crcSum;
    int i;
}

```

684

```
// Process each of the bits within the input uint32_t value
for (crcSum = last, i = 0; i < 4; i += 1)
    crcSum = CrcByte32(crcSum, input >> (24 - 8 * i));
return (crcSum);
}

// Reverse the order of bits within bytes
uint32_t
BitReverse(uint32_t old)
{
    uint32_t new, oldMask, newMask;
    int i, j;

    for (i = new = 0; i < 4; i += 1)
    {
        for (j = 0; j < 8; j += 1)
        {
            oldMask = 1 << (8 * i + 7 - j);
            newMask = 1 << (8 * i + j);
            new |= (old & oldMask) ? newMask : 0;
        }
    }
    return (new);
}
```

```

// ****C00-to-c31 are the most to least significant bits of check; d00-to-d31 are the most to least significant bits of input.
// "a".."t"A.."T" are intermediate bit values.
a = c00 ^ d00, b = c01 ^ d01, c = c02 ^ d02, d = c03 ^ d03, e = c04 ^ d04, f = c05 ^ d05, g = c06 ^ d06, h = c07 ^ d07;
j = c08 ^ d08, k = c09 ^ d09, m = c10 ^ d10, n = c11 ^ d11, p = c12 ^ d12, r = c13 ^ d13, s = c14 ^ d14, t = c15 ^ d15;
A = c16 ^ d16, B = c17 ^ d17, C = c18 ^ d18, D = c19 ^ d19, E = c20 ^ d20, F = c21 ^ d21, G = c22 ^ d22, H = c23 ^ d23;
J = c24 ^ d24, K = c25 ^ d25, M = c26 ^ d26, N = c27 ^ d27, P = c28 ^ d28, R = c29 ^ d29, S = c30 ^ d30, T = c31 ^ d31;
//      00          10          20          30
//      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
//      a b c d e f g h j k m n p r s t A B C D E F G H J K M N P R S T
c00=    d^e^g^j^k^m^n^p^r^s^A^C^G^K^M^T;
c01=    e^f^h^k^m^n^r^s^A^D^H^M^N^T;
c02=    a^b^c^e^h^m^n^p^s^t^C^J^N^P^S^;
c03=    a^b^c^d^f^n^p^r^t^D^K^P^R^T;
c04=    a^b^c^d^e^g^p^r^s^A^E^M^R^S^;
c05=    b^c^d^e^f^h^r^s^t^B^F^N^S^T;
c06=    a^c^d^e^f^g^s^t^A^C^G^P^T;
c07=    a^b^d^e^f^g^h^t^A^B^D^H^R^;
c08=    a^c^f^g^k^n^r^s^A^E^F^J^P^R^;
c09=    b^d^g^h^m^p^s^t^B^F^G^K^R^S^;
c10=    a^c^e^h^n^r^t^C^G^H^M^S^T;
c11=    a^b^d^f^j^p^s^A^D^H^N^T;
c12=    b^c^e^g^j^k^r^t^A^B^E^P^;
c13=    a^c^d^f^h^k^m^n^s^B^C^F^R^;
c14=    a^c^d^f^h^j^k^m^n^t^B^D^E^G^J^K^P^;
c15=    c^d^f^h^j^k^m^n^p^B^F^H^J^K^S^;
c16=    e^h^k^s^t^A^C^D^E^J^K^N^P^;
c17=    a^f^m^t^B^D^E^F^K^M^P^R^;
c18=    c^e^f^h^j^n^B^F^G^J^K^M^R^;
c19=    a^b^c^d^e^h^j^k^p^B^E^G^H^J^K^N^P^;
c20=    a^d^g^h^k^m^n^r^B^E^F^H^J^K^M^P^R^S^;
c21=    b^e^h^j^m^n^s^C^F^G^K^M^N^R^S^T;
c22=    c^f^k^n^p^t^A^D^G^H^M^N^P^S^T;
c23=    a^d^g^j^m^p^r^A^B^E^H^N^P^R^T;
c24=    a^b^c^e^f^g^h^j^B^C^E^J^K^S^;
c25=    a^d^e^j^k^B^D^E^F^J^K^S^T;
c26=    a^c^g^h^j^k^m^A^B^F^G^K^M^N^S^T;
c27=    b^d^h^k^m^n^A^B^C^G^H^K^M^N^T;
c28=    a^b^f^g^h^m^n^p^A^D^E^H^J^K^M^N^P^S^;
c29=    a^e^f^n^p^r^C^F^J^K^N^P^R^S^T;
c30=    b^f^g^p^r^s^A^D^G^K^M^P^R^S^T;
c31=    a^b^e^f^j^r^s^t^A^C^H^J^K^M^N^R^T;
*/

```

```

// ****C00-to-c31 are the most to least significant bits of check; d00-to-d15 are the most to least significant bits of input.
// "a".."t" "A".."T" are intermediate bit values.
a = c00 ^ d00, b = c01 ^ d01, c = c02 ^ d02, d = c03 ^ d03, e = c04 ^ d04, f = c05 ^ d05, g = c06 ^ d06, h = c07 ^ d07;
j = c08 ^ d08, k = c09 ^ d09, m = c10 ^ d10, n = c11 ^ d11, p = c12 ^ d12, r = c13 ^ d13, s = c14 ^ d14, t = c15 ^ d15;
A = c16, B = c17, C = c18, D = c19, E = c20, F = c21, G = c22, H = c23;
J = c24, K = c25, M = c26, N = c27, P = c28, R = c29, S = c30, T = c31;
// 00          10          20          30
// 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
// a b c d e f g h j k m n p r s t A B C D E F G H J K M N P R S T
c00=   c^g^k^m^t^A; ; ;
c01= a^d^h^m^n^B; ; ;
c02= c^j^n^p^s^C; ; ;
c03= d^k^p^r^t^D; ; ;
c04= a^e^m^r^s^E; ; ;
c05= b^f^n^s^t^F; ; ;
c06= a^c^g^p^t^G; ; ;
c07= a^b^d^h^r^H; ; ;
c08= a^e^f^j^p^r^J; ; ;
c09= b^f^g^k^r^s^K; ; ;
c10= c^g^h^m^s^t^M; ; ;
c11= a^d^h^n^t^N; ; ;
c12= a^b^e^p^P; ; ;
c13= b^c^f^r^R; ; ;
c14= b^d^e^g^j^S; ; ;
c15= b^f^h^j^k^s^T; ; ;
c16= a^c^d^e^j^k^n^p; ; ;
c17= b^d^e^f^k^m^p^r; ; ;
c18= b^f^g^j^m^n^r; ; ;
c19= b^e^g^h^j^k^n^p; ; ;
c20= b^e^f^h^j^k^m^p^r^s; ; ;
c21= c^f^g^k^m^n^r^s^t; ; ;
c22= a^d^g^h^m^n^p^s^t; ; ;
c23= a^b^e^h^n^p^r^t; ; ;
c24= b^c^e^j^s; ; ;
c25= b^d^e^f^j^k^s^t; ; ;
c26= a^b^f^j^k^m^s^t; ; ;
c27= a^b^c^g^h^k^m^n^t; ; ;
c28= a^d^e^h^j^m^n^p^s; ; ;
c29= c^f^j^k^n^p^r^s^t; ; ;
c30= a^d^g^k^m^p^r^s^t; ; ;
c31= a^c^h^j^m^n^r^t; ; ;
*/

```

```

// **** CRC_32_EXCHANGED_BY_8 ****
/*
 * c00-to-c31 are the most to least significant bits of check; d00-to-d07 are the most to least significant bits of input.
 * "a".."t" are intermediate bit values.
a = c00 ^ d00, b = c01 ^ d01, c = c02 ^ d02, d = c03 ^ d03, e = c04 ^ d04, f = c05 ^ d05, g = c06 ^ d06, h = c07 ^ d07;
j = c08, k = c09, m = c10, n = c11, p = c12, r = c13, s = c14, t = c15;
A = c16, B = c17, C = c18, D = c19, E = c20, F = c21, G = c22, H = c23;
J = c24, K = c25, M = c26, N = c27, P = c28, R = c29, S = c30, T = c31;
// 00          10          20          30
// 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
// a b c d e f g h j k m n p r s t A B C D E F G H J K M N P R S T
// 00          10          20          30
c00= b^c^      h^j           ;
c01= c^d^      k             ;
c02= a^        d^e^g^       m             ;
c03= b^        e^f^h^       n             ;
c04= c^        f^g^         p             ;
c05= d^        g^h^         r             ;
c06= e^        h^           s             ;
c07= f^        t             ;
c08= a^        e^f^         A             ;
c09= b^        f^g^         B             ;
c10= c^        g^h^         C             ;
c11= d^        h^           D             ;
c12= e^        f^           E             ;
c13= f^        F             ;
c14= a^        G             ;
c15= a^b^      g^           ;
c16= a^b^      d^e^         H             ;
c17= b^c^      e^f^         J             ;
c18= a^        c^d^f^       K             ;
c19= a^b^      d^e^         M             ;
c20= a^b^c^    e^f^g^       N             ;
c21= b^c^d^    f^g^h^       P             ;
c22= c^d^e^    g^h^         R             ;
c23= d^e^f^    h^           S             ;
c24= a^        g             T             ;
c25= a^b^      g^h           ;
c26= a^b^c^    g^h           ;
c27= b^c^d^    h             ;
c28= a^        c^d^e^g^     ;
c29= a^b^      d^e^f^g^h^   ;
c30= b^c^      e^f^g^h^     ;
c31= a^        c^d^f^h^     ;
*/

```

Annex I

(informative)

Datapath scenarios

This annex provides information about possible adverse scenarios on the datapath. To better understand the implications of possible approaches to different datapath problems, multiple strategies are illustrated, each as a separate scenario. For each scenario, the possible problematic and desired behaviors are described.

I.1 Duplicate frame scenarios

I.1.1 Unidirectional source bypass

Unidirectional flooding is susceptible to a source-station-pair loss during flooding, as illustrated in Figure I.1. In this example, source-station S2 and its upstream neighbor S3 are both bypassed while the S2-sourced frame is circulating. Correct source-bypass processing involves discarding the frame when it recirculates beyond its virtual source, as illustrated by the *x* mark within Figure I.1.

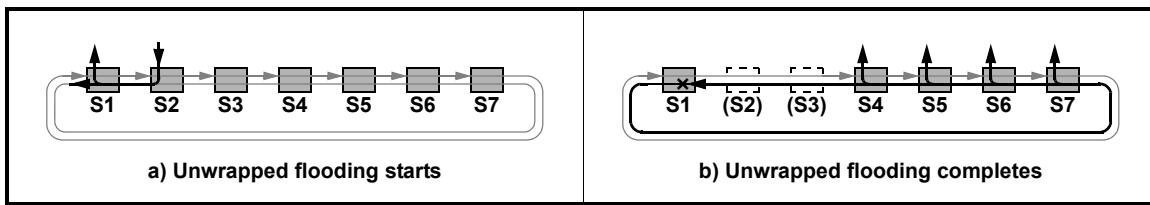


Figure I.1—Duplicate scenario 1: Unidirectional source bypass

Cause: The source (that was responsible for frame deletion) disappears before its frame returns.

Problem: The frame passing through stations S3 and S2 may be falsely accepted by station S1 (and others).

Solution: The frame is discarded at S4 and S7 because $\text{sourceCheck}[\text{ttlBase} - \text{ttl} + 1] \neq \text{frame.sa}$ for the received ringlet.

I.1.2 Unidirectional wrapped source bypass

Unidirectional wrapped flooding is also susceptible to a source-station loss during flooding, as illustrated in Figure I.2. In this example, source-station S2 and its upstream neighbor S3 are both bypassed while the S2-sourced frame is circulating on the right side of station S3. Correct source-bypass processing involves discarding others' transfers when recirculate beyond the source, as illustrated by the *x* mark within Figure I.2.

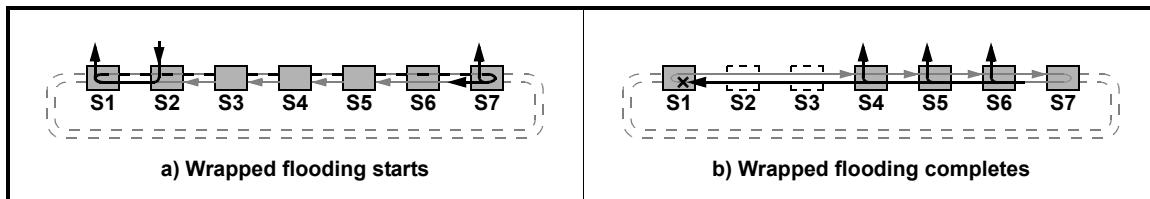


Figure I.2—Duplicate scenario 2: Unidirectional wrapped source bypass

Cause: The source (that was responsible for frame deletion) disappears before its frame returns.

Problem: The frame passing through station S2 may be falsely accepted by station S1 (and others).

Solution: *frame.ps* is not set, which prevents wrap exit.

I.1.3 Bidirectional destination bypass

Bidirectional flooding is susceptible to a destination-station-pair loss during flooding, as illustrated in Figure I.3. In this example, destination stations S5 and S6 are bypassed while the S2-sourced frame is circulating. Correct destination-bypass processing involves discarding the frame when its circulates beyond its virtual destination, as illustrated by the *x* marks within Figure I.3.

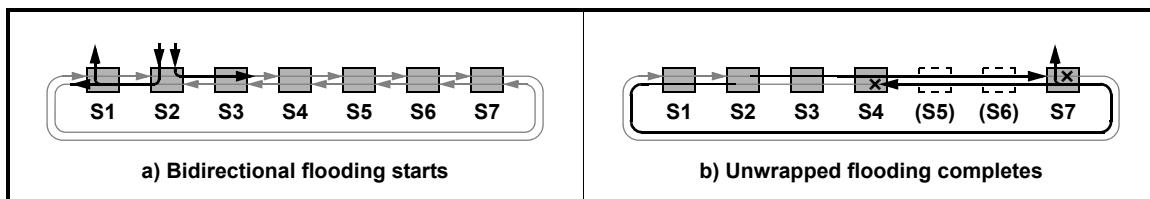


Figure I.3—Duplicate scenario 3: Bidirectional destination bypass

Cause: The destination (that was responsible for frame deletion) disappears before its frame arrives.

Problem: The frame passing through stations S5 and S6 may be falsely duplicated at stations S4 and S7.

Solution: The frame is discarded at S4 and S7 because $\text{sourceCheck}[\text{ttlBase} - \text{ttl} + 1] \neq \text{frame.sa}$ for the received ringlet.

I.1.4 Bidirectional destination removals

Bidirectional wrapped flooding is susceptible to a destination-station-pair loss during flooding, as illustrated in Figure I.4. In this example, destination stations S5 and S6 are removed while the S2-sourced frame is circulating. Correct destination-bypass processing involves discarding the frame when its circulates beyond its virtual destination, as illustrated by the *x* marks within Figure I.4.

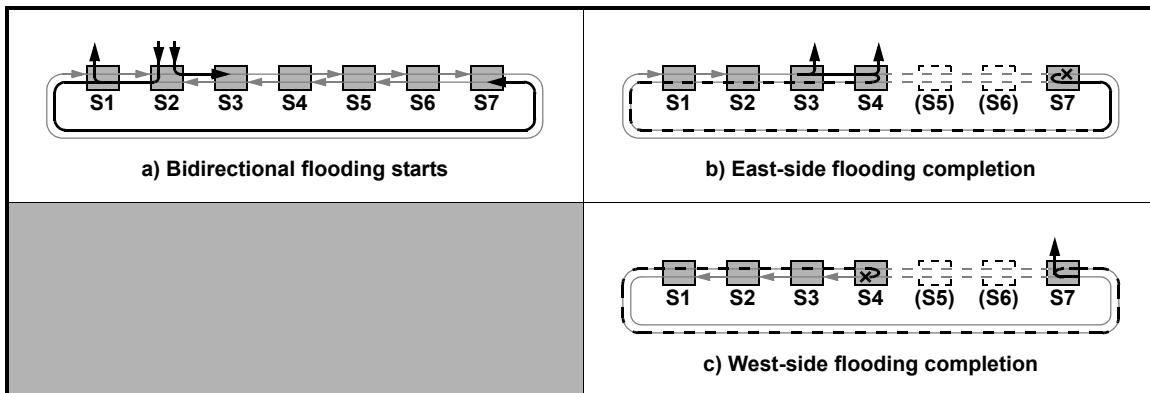


Figure I.4—Duplicate scenario 4: Bidirectional destination removals

Cause: The destination (that was responsible for frame deletion) disappears before its frame arrives.

Problem: The frame wrapped before stations S5 and S6 may be falsely duplicated at stations S4 and S7.

Solution: S7 and S4 will discard frame because $\text{sourceCheck}[\text{ttlBase} - \text{ttl} + 1] \neq \text{frame.sa}$.

I.1.5 Source and destination removals

Unidirectional flooding could be disrupted when half of the stations (including the source and destination stations) are removed, as illustrated in Figure I.5. In this example, source station S2 along with stations S1, S7, and S8 are removed while the S2-sourced frame is circulating. Correct processing involves discarding returning frames when their source is missed.

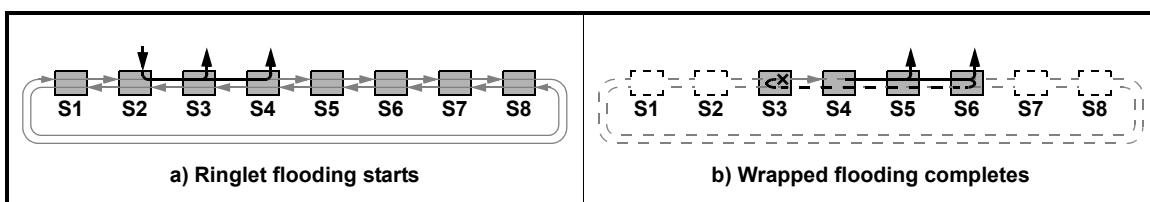


Figure I.5—Duplicate scenario 5: Source and destination removals

Cause: The source (that was responsible for frame deletion) disappears before its frame recirculates.

Problem: The frame may be falsely duplicated when recirculated to station S3 and others.

Solution: The frame is discarded at S3 the second time around because of detecting that $\text{sourceCheck}[\text{ttlBase} - \text{ttl} + 1] \neq \text{frame.sa}$.

I.2 Reordered frame scenarios

I.2.1 Protection switch during bidirectional flood

Bidirectional flooding is susceptible to protection switching during flooding, as illustrated in Figure I.6. In this example, while station S2 is launching bidirectionally flooded frames, a link failure is detected between S1 and S2. When station S2 updates its topology database (i.e., new context), it will start to launch the bidirectional flooded frames using new flooding scopes derived by the new context. Frame reorder is a concern at station S6 and S7 if in-flight frames launched by station S2 (using an old context) arrive after frames launched by station S2, using the new context.

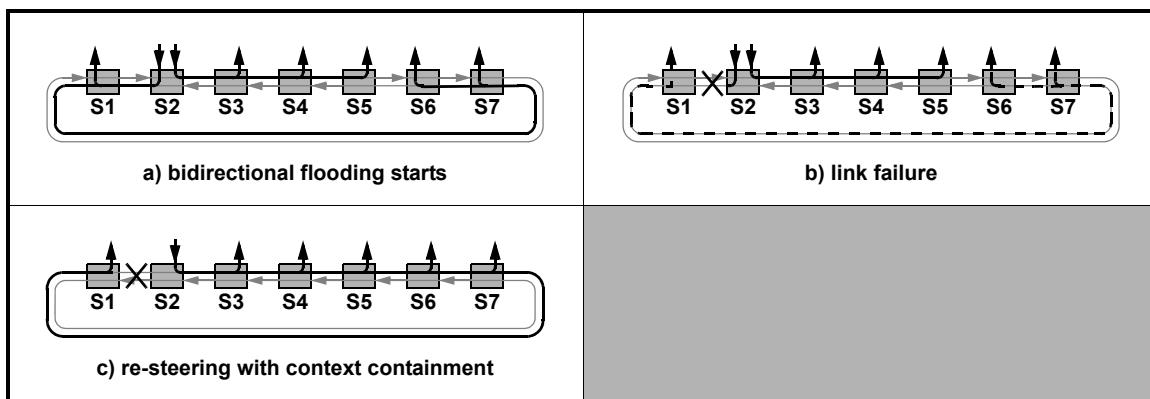


Figure I.6—Reorder scenario 1: Protection switch during bidirectional flood

Cause: In-flight frames dispatched by source (S2) using an old context arrive at a destination after frames dispatched by source using new context.

Problem: The frames received by station S6 or S7 may be received in the wrong order.

Solution: Steering systems use the context containment mechanism to ensure in-flight strict data frames are removed from the ring before strict data frames using a new context are launched.

I.2.2 Cascading failures during bidirectional flood

Bidirectional flooding is susceptible to rapid cascading failures occurring during bidirectional transmission, as illustrated in Figure I.7. In this example, while station S2 is launching bidirectionally flooding frames, the link between station S3 and S4 is restored and fails in rapid succession. Frame reorder is a concern at station S4 and S5 (in this example) if in-flight frames transmit using the context at step 2 are received before in-flight frames transmitted using the context from step 1.

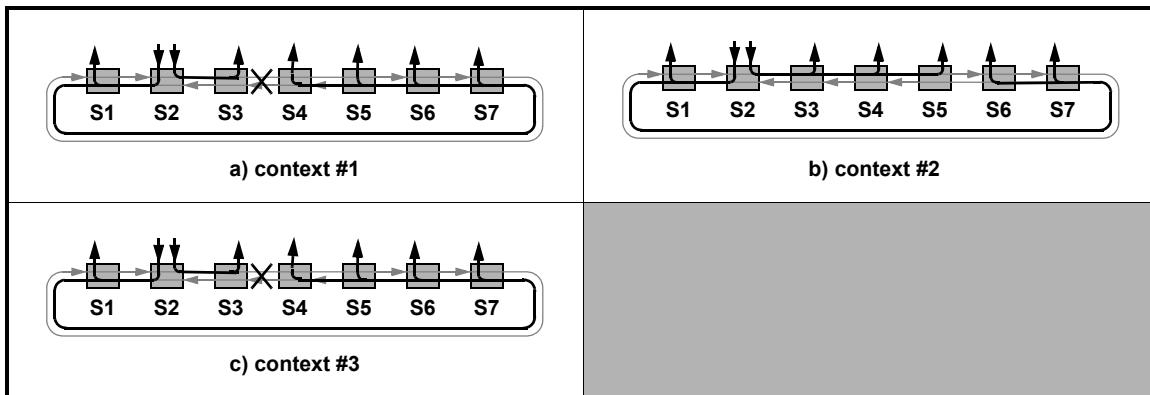


Figure I.7—Reorder scenario 2: Cascading failures during bidirectional flood

Cause: At Figure I.7-a, consider frames in-flight on ringlet1 (using context #1). At Figure I.7-b, frames are launched on ringlet0 and ringlet1 (using context #2). Assume station S5 is now using context #2. That is, station S5 accepts frames launched from S2 using context #2. At Figure I.7-c, station S5 is using context #3. Assume step 2 and step 3 occur while ringlet1 in-flight frames using context #1 are still in-flight. Station S5 can accept in-flight frames on ringlet1 sourced by S2 using context #1.

Problem: Frame reorder can occur if station S5 receives in-flight frames from step 1 after in-flight frames from step 2.

Solution: Steering systems use the context containment mechanism to ensure in-flight strict data frame are removed from the ring before strict data frames using a new context are launched.

I.2.3 Protection switch during unicast transmission on steering system

Unicast frame transmission is susceptible to a protection switch event occurring, as illustrated in Figure I.8. In this example, station S2 is transmitting unicast traffic destined for station S6 over ringlet1. A link failure occurs between station S1 and S2, causing station S2 to dispatch the unicast traffic destined to S6 over ringlet0.

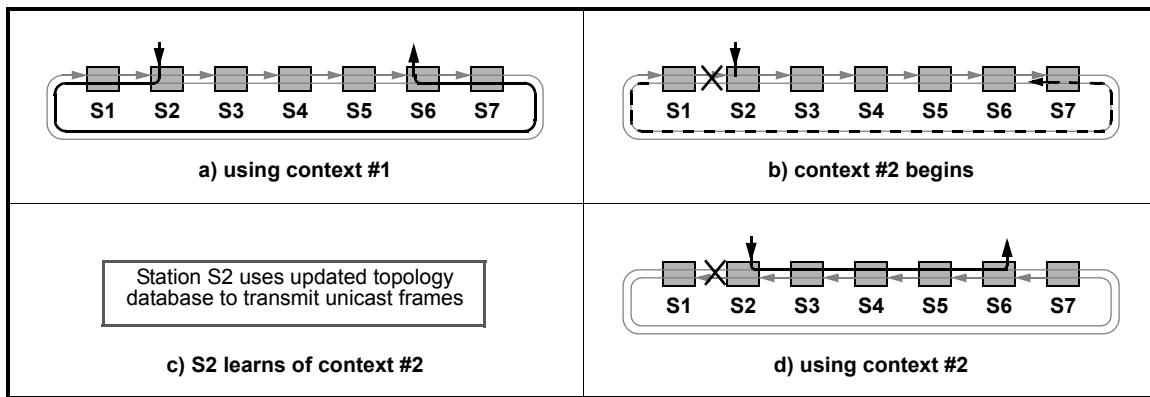


Figure I.8—Reorder scenario 3: Protection switching during unicast transmission

Cause: At Figure I.8-a, consider frames in-flight on ringlet1 (using context #1). At Figure I.8-b, station S2 detects a link failure between station S1 and S2. The context used by station S2 is updated to reflect configuration shown in step 2. At Figure I.8-d, unicast frames destined to S6 are sent over ringlet0.

Problem: Frame reorder can occur if station S6 receive context 2 frames before context 1 frames.

Solution: Steering systems use the context containment mechanism to ensure in-flight strict data frame are removed from the ring before strict data frames using a new context are launched.

I.2.4 Cascading protection switch during unidirectional flood, wrapping

Unidirectional flooding is susceptible to rapid cascading failures occurring during unidirectional transmission, as illustrated in Figure I.9. In this example, while station S2 is launching unidirectional flooded frames, the link between station S3 and S4 is restored and fails. Frame reorder is a concern at station S3 (in this example) if frames transmitted at step 3 are received before wrapped frames on secondary ring transmitted at step 1 potentially get unwrapped at station S3.

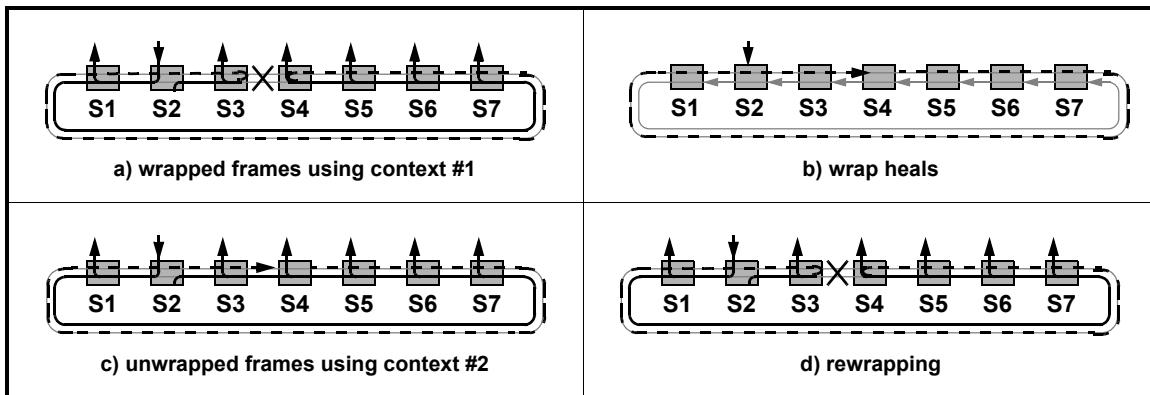


Figure I.9—Reorder scenario 4: Cascading failures during unidirectional flood

Cause: At Figure I.9-a, consider wrapped frames on secondary ringlet. At Figure I.9-b, the ring heals, however the frames on secondary ringlet continue to circulate until its time to live expires. During frame circulation on secondary ringlet, frames are transmitted by station S2 (as shown in Figure I.9-c). If another failure occurs (Figure I.9-d), prior to circulation of frames on secondary ringlet having their time to live expire, frames launched at step 3 can be received by station S3 before unexpired frames on the secondary ringlet get exit the wrap condition at station S3.

Problem: Frame reorder can occur if station S3 receive context 3 frames before context 1 frames.

Solution: Wrapping systems will purge all strict data frames with $frame.ri \neq myRi$ for a specified duration. All wrapped strict data frames using an outdated context will be removed from the ring.

Annex J

(informative)

Spatial indications and shaping

J.1 Overview

This text is included only for illustrative purposes for those implementers who might want to understand the current standard or extend it in the future.

There are two forms of traffic handling with respect to allocation, queuing, and shaping: uniform and spatial. Uniform traffic handling uses a single rate per class for the entire ring. Spatial traffic handling uses independent rates per class for each link. Uniform traffic handling makes no distinction among traffic flows based on the number of spans traversed by their paths. Spatial traffic handling differentiates traffic amounts on a per hop-count basis. This standard describes only how to accomplish uniform traffic handling.

J.2 Spatial bandwidth allocation

This subclause describes how bandwidth allocation profiles are created in a spatially aware system.

J.2.1 Single queue spatial allocation

Each station has allocated levels of classA and classB traffic, as illustrated in Figure J.1 for station S3. Spatial allocation makes a distinction between classA traffic sent from S3-to-S2 (Figure J.1-a) and classA traffic sent from S3-to-S1 (Figure J.1-b). Similarly, spatial allocation makes a distinction between classB traffic sent from S3-to-S1 (Figure J.1-c) and classB traffic sent from S3-to-S4 (Figure J.1-d).

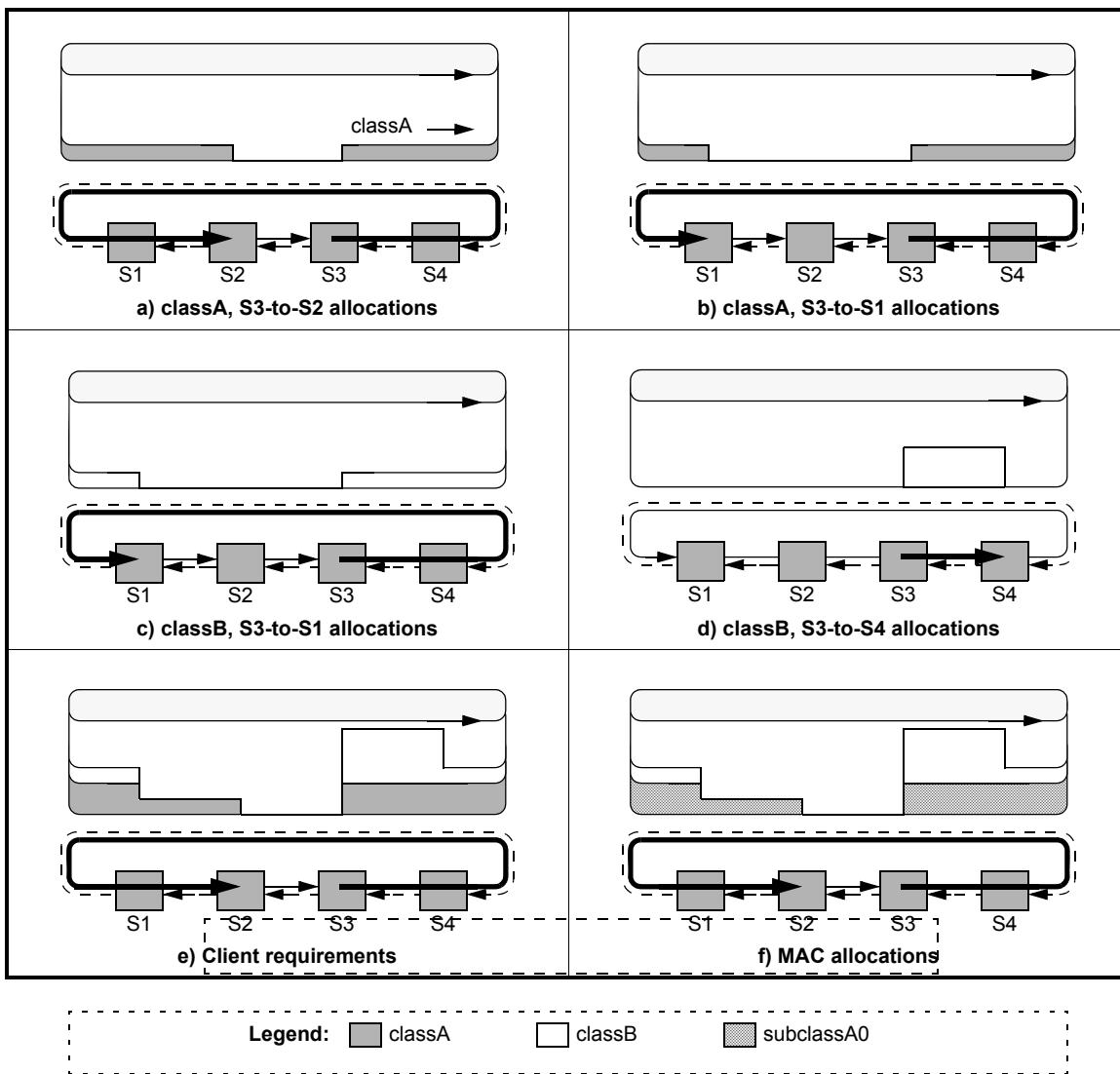


Figure J.1—Single queue spatial allocation

For each station, the sum of class profiles forms a cumulative bandwidth profile (Figure J.1-e) of client-visible classA and classB bandwidth allocations. The classA service (Figure J.1-f) consists of only subclassA0 bandwidth, because a second transit queue is necessary to support subclassA1.

J.2.2 Dual queue spatial allocation

Each station has allocated levels of classA and classB traffic, as illustrated in Figure J.2 for station S4. Spatial allocation of classA traffic makes a distinction between traffic sent from S4-to-S3 (Figure J.2-a) and traffic sent from S4-to-S2 (Figure J.2-b). Similarly, spatial allocation makes a distinction between classB traffic sent from S4-to-S2 (Figure J.2-c) and classB traffic sent from S4-to-S1 (Figure J.2-d).

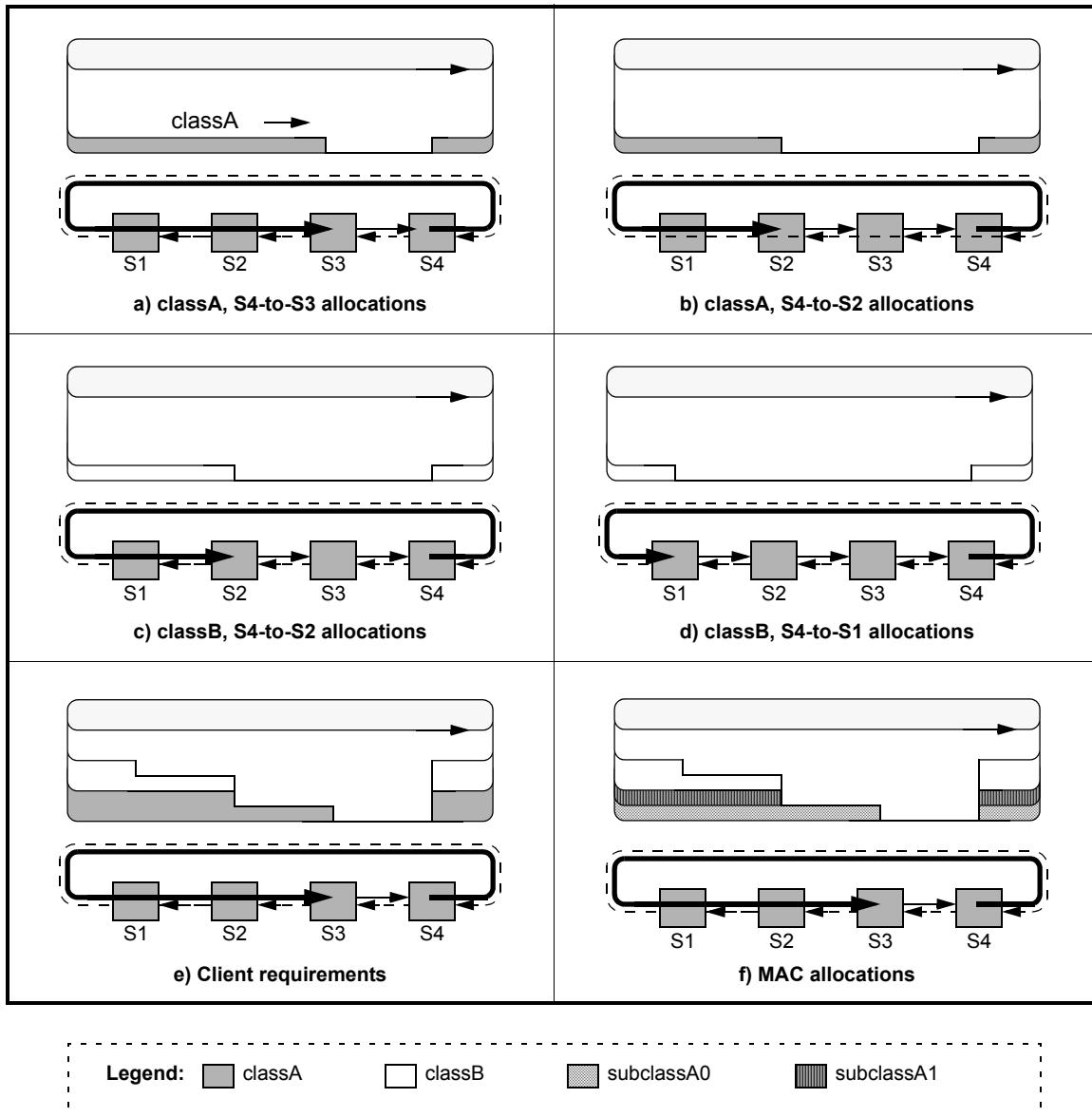


Figure J.2—Dual queue spatial allocation

For each station, the sum of class profiles forms a cumulative bandwidth profile (Figure J.2-e) of client-visible classA and classB bandwidth allocations. The classA service (Figure J.2-f) consists of subclassA0 and subclassA1 bandwidths, where the level of supportable subclassA1 bandwidth is proportional to the size of the secondary transit queue.

J.2.3 Cumulative ringlet allocation

Each station has its own cumulative class profile, as illustrated in Figure J.3a through Figure J.3-d. These can be combined (Figure J.3-e) into a ringlet class profile. Within consistent ringlet profiles, the sum of subclassA0, subclassA1, and classB profiles (*allocated* in Figure J.3-e) will be less than any individual link capacity.

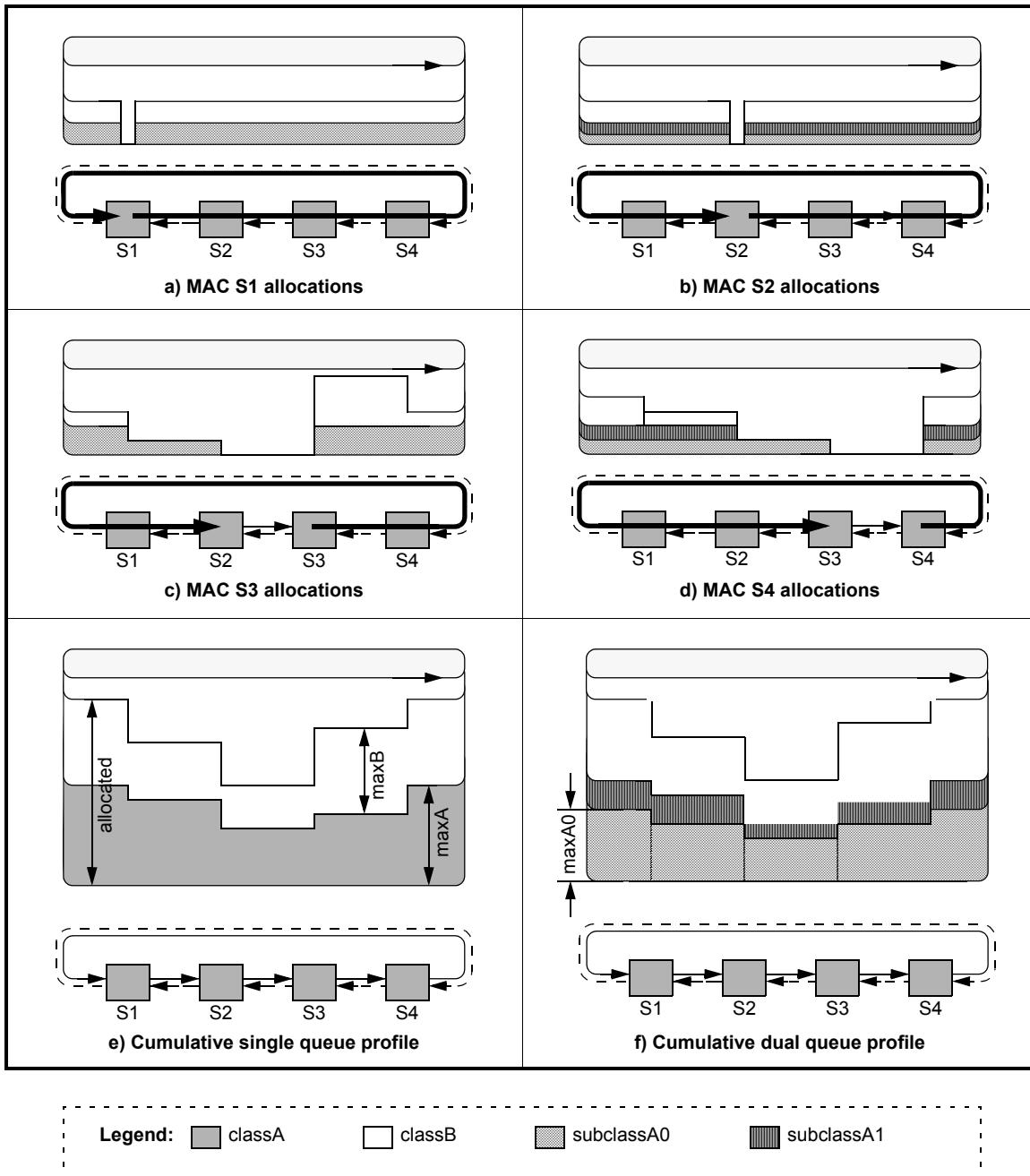


Figure J.3—Cumulative station allocations

For uniform rate control, the cumulative class profile yields $maxA$ and $maxA0$, the worst-case allocated classA and subclassA0 segments, respectively. For spatial rate control, the cumulative class profile provides $rateA[n]$ and $rateA0[n]$, the allocated classA and subclassA0 levels on segment n , respectively. This information is used to rate-limit each station's classB and classC transmissions, with the intent of sustaining downstream classA transmissions.

During allocation, the cumulative class profile also yields $maxB$, the worst-case allocated classB segment. To ensure interoperability between spatial-aware and non-spatial-aware stations, the sum of $maxA$ and $maxB$ will be less than the link capacity.

NOTE—The value of $maxA+maxB$ equals the maximum value of $rateA[n]+rateB[m]$, measured on all links n and m . The requirement for this sum to be less than the link capacity is more restrictive than the physical mandated restriction that $rateA[n]+rateB[n]$ be less than the link capacity on any link n .

J.3 Spatial client queuing

The behavior of the MAC client is presented for descriptive purposes. This subclause does not impose any behavior for the MAC client. The RPR standard defines a set of primitives at the MSAP interface. The number of queues and the queue managers at the MAC client are a matter of choice.

The simplest MAC client can have one queue for each traffic class. For these client queues, the $sendA$, $sendB$, and $sendC$ indications indicate which traffic class can be sent. If the MAC client sends a frame for a currently blocked traffic class, the MAC rate control functionality disallows additional frames until that traffic class becomes unblocked.

A possible extension to this standard would be to enhance the send indication from binary values to vector values. A simple (not spatially aware) client using such an enhanced MAC would transmit frames only when the value of $sendA$, $sendB$, or $sendC$ were set to MAX_STATIONS. All other values would be interpreted as a status disallowing transmission. This is illustrated in Table J.1.

Table J.1—MAC client interface

sendA	sendB	sendC	Queue selection (top-to-bottom precedence within each cell)
SEND	SEND	SEND	Select frame from classA queue. Select frame from classB queue. Select frame from classC queue.
		!SEND	Select frame from classA queue. Select frame from classB queue.
	!SEND	SEND	Select frame from classA queue. Select frame from classB queue, set $fe=1$ and treat as classC. Select frame from classC queue.
		!SEND	Select frame from classA queue.
!SEND	SEND	SEND	Select frame from classB queue. Select frame from classC queue.
		!SEND	Select frame from classB queue.
	!SEND	SEND	Select frame from classB queue, set $fe=1$ and treat as classC. Select frame from classC queue.
		!SEND	No frame selected

Optionally, the MAC client may implement a more sophisticated queuing scheme to avoid head-of-line blocking and to utilize more bandwidth. This can be accomplished through the use of hop-count information transmitted to the client within the *sendA*, *sendB*, and *sendC* indications. For this purpose, the MAC client can implement virtual output queues for each destination on the ring.

The MAC client is allowed to send a frame from a virtual output queue for classA traffic if the hop count to the selected destination does not exceed the *sendA*-specified value. The MAC client is allowed to send a frame from a virtual output queue for classB traffic if the hop count to the selected destination does not exceed the *sendB*-specified value. The MAC client is allowed to send a frame from a virtual output queue for classC or excess classB traffic if the hop count to the selected destination does not exceed the *sendC*-specified value.

At any time there can be more than one virtual output queue that will satisfy the condition. In this case, a round-robin approach can be chosen to simplify the solution. However, a better approach will be using deficit-round-robin, which will avoid possible unfairness among virtual output queues.

Depending on the client's behavior, the interpretation of *sendA*, *sendB*, and *sendC* indications will vary. For virtual destination queuing, intermediate values (between 0 and MAX_STATIONS) affect the selection of virtual output queues. In the absence of virtual queues, only the distinction between MAX_STATIONS and non-MAX_STATIONS values is relevant.

Annex K

(informative)

Client-based OAM operations using echo and flush

K.1 Connectivity monitoring using echo request/response

K.1.1 Background

RPR stations are connected by point-to-point links that form a ring. The ring acts as a shared media, in a way that any station can directly communicate with any other station that is connected to the ring. It is desirable to have comprehensive tools to detect failures that impair the normal communication between any pair of stations at the RPR layer.

The normal operation of the lower layer cannot be interpreted as the sole indication for the performance of the communication at the RPR layer between the stations. Even if the lower layer is operating normally for all the spans present between the communicating stations, the communication between them may be impaired by faults in the RPR MAC. Examples of such faults are stations “stealing” frames destined to other stations and transit queue failures.

K.1.2 Scope of operations

The RPR MAC does not directly support peer-to-peer connectivity monitoring at the RPR layer, but similar functionality may be created in the LME layer using the RPR OAM echo request and echo response functions. This annex describes how such a function may be implemented.

All stations are required to implement processing of echo request and generation of the echo response. As a result, a station can implement this connectivity monitor without any special configuration of the peer station.

K.1.3 Connectivity monitor

Verifying normal connectivity between stations implies monitoring the possible paths for the relevant service classes. Between any pairs of stations, there are two possible paths (east and west) that can carry up to three service classes (classA, classB, and classC). To validate connectivity, some or all these paths and service classes may be checked.

RPR OAM functions provide the ability to send an echo request to any peer station on the ring, via a given ringlet and service class with a block of user data. The peer station must generate an echo response in the same service class, on the ringlet as specified in the echo request, and including the user data from the echo request.

This allows each station to monitor connectivity with:

- a) One station
- b) A set of stations
- c) All stations on the ring

K.1.3.1 Monitored paths

As stated before, there are up to two possible paths between stations. This allows each station to monitor connectivity with:

- a) A specific path: east or west
- b) Both paths: east and west
- c) The default path

The echo request and response may be protected or unprotected, as specified in 6.2.

For each path, this allows each station to monitor connectivity with:

- a) A single service class (classA, classB, or classC)
- b) Two out of the three service classes
- c) All the service classes

K.1.3.2 Monitoring characteristics

To limit the bandwidth consumed by the echo frames, the following limit on the number of echo request frames transmitted per time interval is recommended:

One echo request transmission every T1 milliseconds, where $T1 > 0.5$ milliseconds with a default value of 10.

The expected time between an echo request and an echo response is a function of the distance to the target station and the echo request processing time. The echo request processing time is defined as the time interval between when a station receives an echo request and the time it transmits back the echo response.

K.1.4 Failure declaration and clearing

Loss of continuity defect (dLOC) is declared if an echo response is not received T2 milliseconds after the echo request was transmitted. T2 is defined as follows:

T2 is a configurable value between 1 and 100 milliseconds with a default value of 10.

It is the responsibility of the echo request generating station to correlate between the echo requests and echo responses. This annex does not specify how this correlation is implemented; it is assumed that implementers will use the user data field of the echo request/response frame for that purpose.

Loss of continuity failure (LOC) is declared if dLOC persists for 2.5 ± 0.5 seconds. LOC is cleared if no dLOC is detected during 10 ± 0.5 seconds.

LOC and dLOC are monitored per target station, ringlet, and service class.

K.2 Resteering using flush

K.2.1 Background

RPR stations are not supposed to change the direction that frames are sent on a ring for frames that share the same values for the tuple of $\{sa, da, \text{service_class}\}$ (see 7.6.1). The direction is allowed to change as a result of changes in protection and topology. However, a station may want to change the direction that frames are sent during other times.

During normal operation, a station's higher level management may wish to change frame transmission direction based on criteria beyond the scope of the MAC. These criteria could include bandwidth or connection balancing, preparing in advance for a planned outage along the current direction, and reacting to a previous protection or topology change.

K.2.2 Approach

In order to preserve the order of frames within the same $\{sa, da, \text{service_class}\}$ tuple, frames that are to be steered in a new direction must be withheld from transmission until all previous frames within the same tuple have been delivered. Once they are known to have been delivered (or discarded), the new direction may be safely taken via client steering or via changing of the default steering algorithm used by the MAC for this set of frames.

A simple and robust method to assure that all frames within a set of frames has been delivered is to send an OAM flush frame to oneself along the current ringlet and with the same (or lower priority) service_class as used by the frames to be resteered. Upon receipt of the flush frame, because of the FIFO nature of the transit path for any individual service class, the MAC knows that all other frames of the same (and higher priority) service_class have had the opportunity to be delivered.

K.2.3 Using echo frames to flush steer protected rings

Flushing is not possible on a steer-protected ring, because traffic is discarded (rather than wrapped) at the endpoints. However, an endpoint-addressed echo can be used in a similar fashion. Figure K.1 illustrates an example of the use of echo frames for flushing the transit queue. S3 is transmitting frames to S1 through ringlet0 (Figure K.1-a). When a failure is present between S4 and S5 (Figure K.1-b), some frames may be in transit between S5 and S1 in ringlet0. To flush the ringlet0 transit path, S3 may issue a echo frame to S5 (Figure K.1-c). Once the echo response is received, it is safe for S3 to steer its transmission to S1 to ringlet1 (Figure K.1-d).

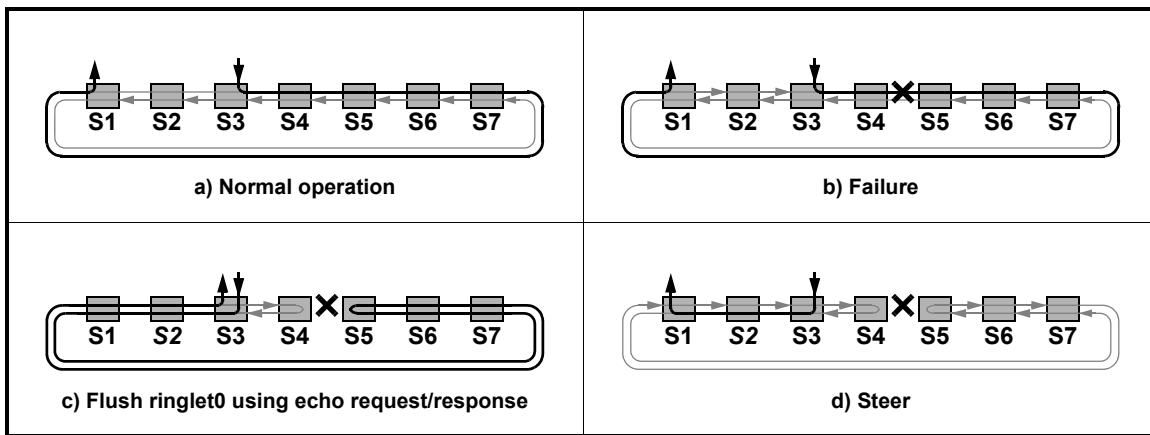


Figure K.1—Flushing a steered ring

This method effectively flushes the PTQ in all stations. For dual transit buffer implementations, the STQ of the echo target station is not flushed because the echo request is stripped before reaching the STQ and the echo response is transmitted by the target station.

Index

A

addrType
See ATT_MANAGE_ADDRESS

ATD frame

<i>controlDataUnit</i>	
<i>res1</i>	297
<i>type</i>	297
ATT_WEIGHT	297
ATT_STATION_BW	297
ATT_STATION_SET	298
ATT_STATION_NAME	298
ATT_MANAGE_ADDRESS	298
ATT_INTERFACE_INDEX	298
ATT_SECONDARY_MACS	298
ATT_ORG_SPECIFIC	298
<i>res2</i>	298
<i>length</i>	298
<i>attDataUnit</i>	298

ATT_INTERFACE_INDEX

<i>attDataUnit</i>	
<i>rprIfIndex</i>	301

See also ATD frame; type

ATT_MANAGE_ADDRESS

<i>attDataUnit</i>	
<i>addrType</i>	301
<i>stationIpAddress</i>	301

See also ATD frame; type

ATT_ORG_SPECIFIC

<i>attDataUnit</i>	
<i>oui</i>	302
<i>dependentID</i>	302
<i>data</i>	302

See also ATD frame; type

ATT_SECONDARY_MACS

<i>attDataUnit</i>	
<i>secondaryMacAddress0</i>	302
<i>secondaryMacAddress1</i>	302

See also ATD frame; type

ATT_STATION_BW

<i>attDataUnit</i>	
<i>ringlet0ReservedBw</i>	299
<i>ringlet1ReservedBw</i>	299

See also ATD frame; type

ATT_STATION_NAME

<i>attDataUnit</i>	
<i>stationName</i>	300

See also ATD frame; type

ATT_STATION_SET

<i>attDataUnit</i>	
<i>res</i>	299
<i>bf</i>	299
<i>co</i>	300
<i>mu</i>	300

See also ATD frame; type

ATT_WEIGHT

<i>attDataUnit</i>	
<i>ringlet0Weight</i>	298
<i>ringlet1Weight</i>	298

See also ATD frame; type

attDataUnit

See ATD frame

B

baseControl

<i>ri</i>	
RINGLET_0	210
RINGLET_1	210

<i>fe</i>	210
-----------------	-----

<i>ft</i>	
FT_IDLE	211
FT_CONTROL	211
FT_FAIRNESS	211
FT_DATA	211

<i>sc</i>	
CLASS_A0	211
CLASS_A1	211
CLASS_B	211
CLASS_C	211

<i>we</i>	212
-----------------	-----

<i>parity</i>	212
---------------------	-----

See also basic data frame

See also control frame

See also extended data frame

See also fairness frame

See also idle frame

basic data frame

<i>ttl</i>	204
------------------	-----

<i>baseControl</i>	204
--------------------------	-----

See also baseControl

<i>da</i>	204
-----------------	-----

<i>sa</i>	204
-----------------	-----

<i>ttlBase</i>	205
----------------------	-----

<i>extendedControl</i>	205
------------------------------	-----

See also extendedControl

<i>hec</i>	205
------------------	-----

<i>protocolType</i>	205
---------------------------	-----

<i>serviceDataUnit</i>	205
------------------------------	-----

<i>fcs</i>	205
------------------	-----

bf

See ATT_STATION_SET

C

checksum

See topology checksum (TC) frame

checksumStatus

See topology checksum (TC) frame

CLASS_A0

See baseControl; sc

CLASS_A1

See baseControl; sc

CLASS_B

See baseControl; sc

CLASS_C

See baseControl; sc

co

See ATT_STATION_SET

control frame

ttl 206

baseControl 206

See also baseControl

da 206

sa 206

ttlBase 206

extendedControl 206

See also extendedControl

hec 207

controlType 207

CT_STATION_ATD 207

CT_TOPO_PROT 207

CT_TOPO_CHKSUM 207

CT_LRTT_REQ 207

CT_LRTT_RSP 207

CT_FDD 207

CT_OAM_ECHO_REQ 207

CT_OAM_ECHO_RSP 207

CT_OAM_FLUSH 207

CT_OAM_ORG 207

controlVersion 207

controlDataUnit 207

fcs 207

control frames

See ATD frame

See echo request/response frame

See fairness differential delay (FDD) frame

See flush frame

See loop round-trip time (LRTT) request frame

See loop round-trip time (LRTT) response frame

See organization-specific frame

See topology and protection (TP) frame

See topology checksum (TC) frame

CONTROL_MAX

See frame sizes

CONTROL_MIN

See frame sizes

controlDataUnit

See control frame

controlType

See control frame

controlVersion

See control frame

CT_FDD

See control frame; controlType

See fairness differential delay (FDD) frame

CT_LRTT_REQ

See control frame; controlType

See loop round-trip time (LRTT) request frame

CT_LRTT_RSP

See control frame; controlType

See loop round-trip time (LRTT) response frame

CT_OAM_ECHO_REQ

See control frame; controlType

See echo request/response frame

CT_OAM_ECHO_RSP

See control frame; controlType

See echo request/response frame

CT_OAM_FLUSH

See control frame; controlType

See flush frame

CT_OAM_ORG

See control frame; controlType

See organization-specific frame

CT_STATION_ATD

See ATD frame

See control frame; controlType

CT_TOPO_CHKSUM

See control frame; controlType

See topology checksum (TC) frame

CT_TOPO_PROT

See control frame; controlType

See topology and protection (TP) frame

cv

See topology checksum (TC) frame; checksumStatus

D

da

See basic data frame

See control frame

See extended data frame

daExtended

See extended data frame

data

See ATT_ORG_SPECIFIC

DATA_MIN

See frame sizes

DEGRADE_FAIL

See PHY_LINK_STATUS.indication

<i>dependentID</i>	<i>extendedControl</i>
<i>See ATT_ORG_SPECIFIC</i>	<i>ef</i>212
<i>See organization-specific frame</i>	<i>fi</i>
<i>destination_address</i>	FI_NONE.....213
<i>See MA_DATA.indication</i>	FI_UNIDIR.....213
<i>See MA_DATA.request</i>	FI_BIDIR.....213
<i>destination_address_extended</i>	FI_RES.....213
<i>See MA_DATA.indication</i>	<i>ps</i>213
<i>See MA_DATA.request</i>	<i>so</i>213
E	<i>res</i>213
<i>echo request/response frame</i>	
<i>controlDataUnit</i>	
<i>responseControl</i>366	
<i>res</i>366	
<i>protectionMode</i>366	
<i>responseRinglet</i>366	
RR_RINGLET0.....366	
RR_RINGLET1.....366	
RR_REVERSE.....366	
RR_DEFAULT.....366	
<i>userData</i>366	
<i>ef</i>	
<i>See extendedControl</i>	
<i>ERROR</i>	
<i>See PHY_DATA.indication</i>	
<i>ese</i>	
<i>See topology and protection (TP) frame; protStatus</i>	
<i>esw</i>	
<i>See topology and protection (TP) frame; protStatus</i>	
<i>EXT_HDR_SIZE</i>	
<i>See frame sizes</i>	
<i>extended data frame</i>	
<i>ttl</i>204	
<i>baseControl</i>204	
<i>See also baseControl</i>	
<i>da</i>204	
<i>sa</i>204	
<i>ttlBase</i>205	
<i>extendedControl</i>205	
<i>See also extendedControl</i>	
<i>hec</i>205	
<i>daExtended</i>205	
<i>saExtended</i>205	
<i>protocolType</i>205	
<i>serviceDataUnit</i>205	
<i>fcs</i>205	
<i>extended_frame</i>	
<i>See MA_DATA.indication</i>	
F	
<i>fairness differential delay (FDD) frame</i>	
<i>controlDataUnit</i>	
<i>sequenceNumber</i>238	
<i>fddTimeStamp</i>238	
<i>fairness frame</i>	
<i>ttl</i>208	
<i>baseControl</i>208	
<i>See also baseControl</i>	
<i>saCompact</i>209	
<i>fairnessHeader</i>209	
<i>ffType</i>237	
<i>res</i>237	
<i>fairRate</i>209, 237	
<i>fcs</i>209	
<i>fairness_eligible</i>	
<i>See MA_DATA.indication</i>	
<i>FAIRNESS_SIZE</i>	
<i>See frame sizes</i>	
<i>fairnessHeader</i>	
<i>See fairness frame</i>	
<i>fairRate</i>	
<i>See fairness frame</i>	
<i>fcs</i>	
<i>See basic data frame</i>	
<i>See control frame</i>	
<i>See extended data frame</i>	
<i>See fairness frame</i>	
<i>See idle frame</i>	
<i>fddTimeStamp</i>	
<i>See fairness differential delay (FDD) frame</i>	
<i>fe</i>	
<i>See baseControl</i>	
<i>ffType</i>	
<i>See fairness frame; fairnessHeader</i>	
<i>fi</i>	
<i>See extendedControl</i>	
<i>FI_BIDIR</i>	
<i>See extendedControl; fi</i>	

FI_NONE	
<i>See extendedControl; fi</i>	
FI_RES	
<i>See extendedControl; fi</i>	
FI_UNIDIR	
<i>See extendedControl; fi</i>	
FLOOD_BIDIR	
<i>See MA_DATA.request</i>	
FLOOD_NONE	
<i>See MA_DATA.request</i>	
FLOOD_UNIDIR	
<i>See MA_DATA.request</i>	
flooding_form	
<i>See MA_DATA.request</i>	
flush frame	
<i>controlDataUnit</i>	
<i>res</i> 367	
<i>userData</i> 367	
frame sizes	
data frame sizes	
DATA_MIN 203	
EXT_HDR_SIZE 203	
REGULAR_MAX 203	
JUMBO_MAX 203	
control frame sizes	
CONTROL_MIN 205	
CONTROL_MAX 205	
fairness frame sizes	
FAIRNESS_SIZE 208	
idle frame sizes	
IDLE_SIZE 209	
frame_check_sequence	
<i>See MA_DATA.indication</i>	
<i>See MA_DATA.request</i>	
FS	
<i>See topology and protection (TP) frame; pse</i>	
<i>See topology and protection (TP) frame; psw</i>	
ft	
<i>See baseControl</i>	
FT_CONTROL	
<i>See baseControl; ft</i>	
<i>See control frame</i>	
FT_DATA	
<i>See baseControl; ft</i>	
<i>See data frame</i>	
FT_FAIRNESS	
<i>See baseControl; ft</i>	
<i>See fairness frame</i>	
FT_IDLE	
<i>See baseControl; ft</i>	
<i>See idle frame</i>	
H	
hec	
<i>See basic data frame</i>	
<i>See control frame</i>	
<i>See extended data frame</i>	
I	
IDLE	
<i>See topology and protection (TP) frame; pse</i>	
<i>See topology and protection (TP) frame; psw</i>	
idle frame	
<i>ttl</i> 209	
<i>baseControl</i> 209	
<i>See also baseControl</i>	
<i>saCompact</i> 210	
<i>idlePayload</i> 210	
<i>fcs</i> 210	
IDLE_SIZE	
<i>See frame sizes</i>	
<i>idlePayload</i>	
<i>See idle frame</i>	
indication_operand_list	
<i>See MA_CONTROL.request</i>	
INPUT_FRAME	
<i>See PHY_DATA.indication</i>	
J	
jp	
<i>See topology and protection (TP) frame; prefs</i>	
JUMBO_MAX	
<i>See frame sizes</i>	
L	
latencyTimestamp	
<i>See loop round-trip time (LRTT) request frame</i>	
<i>See loop round-trip time (LRTT) response frame</i>	
LENGTH	
<i>See PHY_DATA.indication</i>	
<i>See PHY_DATA.request</i>	
length	
<i>See ATD frame</i>	
LINK_STATUS	
<i>See PHY_LINK_STATUS.indication</i>	
loop round-trip time (LRTT) request frame	
<i>controlDataUnit</i>	
<i>latencyTimestamp</i> 295	
<i>tailLatencyIn</i> 295	
<i>tailLatencyOut</i> 295	
loop round-trip time (LRTT) response frame	
<i>controlDataUnit</i>	
<i>latencyTimestamp</i> 296	

loop round-trip time (LRTT) request frame	
<i>controlDataUnit</i>	
<i>tailLatencyIn</i>	296
<i>tailLatencyOut</i>	296
M	
MA_CONTROL.indication	
opcode	
OAM_ECHO_IND	60
OAM_FLUSH_IND	60
OAM_ORG_IND.....	60
TOPO_CHANGE	60
PROT_CHANGE.....	60
SENDA	60
SENDB	60
SENDc	60
SINGLE_CHOKE_IND	60
MULTI_CHOKE_IND.....	60
indication_operand_list.....	59
MA_CONTROL.request	
opcode	
OAM_ECHO_REQ	59
OAM_FLUSH_REQ	59
OAM_ORG_REQ.....	59
request_operand_list.....	59
MA_DATA.indication	
destination_address.....	57
source_address.....	57
mac_service_data_unit	57
frame_check_sequence.....	57
reception_status	
RECEIVE_OK.....	57
RECEIVE_FCS_ERROR	57
service_class	
SC_CLASSA	57
SC_CLASSB	57
SC_CLASSC	57
ringlet_id	
RI_0	57
RI_1	57
RI_DEFAULT	57
(null).....	57
fairness_eligible.....	58
strict_order.....	58
extended_frame.....	58
destination_address_extended	58
source_address_extended	58
MA_DATA.request	
destination_address	53
source_address.....	53
mac_service_data_unit	54
frame_check_sequence.....	54
service_class	
SC_CLASS_A.....	54
SC_CLASS_B	54
SC_CLASS_C	54
ringlet_id	
RI_0	54
RI_1	54
RI_DEFAULT	54
(null).....	54
mac_protection	54
mark_fe.....	54
strict_order.....	55
destination_address_extended	55
source_address_extended	55
flooding_form	
FLOOD_NONE.....	55
FLOOD_UNIDIR	55
FLOOD_BIDIR	55
(null)	55
mac_protection	
<i>See MA_DATA.request</i>	
mac_service_data_unit	
<i>See MA_DATA.indication</i>	
<i>See MA_DATA.request</i>	
mark_fe	
<i>See MA_DATA.request</i>	
MS	
<i>See topology and protection (TP) frame; pse</i>	
<i>See topology and protection (TP) frame; psw</i>	
mu	
<i>See ATT_STATION_SET</i>	
MULTI_CHOKE_IND	
<i>See MA_CONTROL.indication</i>	
N	
NO_DEFECT	
<i>See PHY_LINK_STATUS.indication</i>	
NOT_READY	
<i>See PHY_READY.indication</i>	
O	
OAM_ECHO_IND	
<i>See MA_CONTROL.indication</i>	
OAM_ECHO_REQ	
<i>See MA_CONTROL.request</i>	
OAM_FLUSH_IND	
<i>See MA_CONTROL.indication</i>	
OAM_FLUSH_REQ	
<i>See MA_CONTROL.request</i>	

OAM_ORG_IND	<i>pse</i>
<i>See MA_CONTROL.indication</i>	<i>See topology and protection (TP) frame; protStatus</i>
OAM_ORG_REQ	<i>psw</i>
<i>See MA_CONTROL.request</i>	<i>See topology and protection (TP) frame; protStatus</i>
OK	R
<i>See PHY_DATA.indication</i>	READY
organizationEUI	<i>See PHY_READY.indication</i>
<i>See ATT encodings; organization-specific organization-specific frame</i>	READY_STATUS
controlDataUnit	<i>See PHY_READY.indication</i>
<code>oui</code> 367	RECEIVE_FCS_ERROR
<code>dependentID</code> 367	<i>See MA_DATA.indication</i>
<code>userData</code> 367	RECEIVE_OK
<i>See ATT_ORG_SPECIFIC</i>	<i>See MA_DATA.indication</i>
<i>See organization-specific frame</i>	reception_status
OUTPUT_FRAME	<i>See MA_DATA.indication</i>
<i>See PHY_DATA.request</i>	REGULAR_MAX
P	<i>See frame sizes</i>
parity	request_operand_list
<i>See baseControl</i>	<i>See MA_CONTROL.request</i>
PHY_DATA.indication	res
INPUT_FRAME..... 192	<i>See ATT_STATION_SET</i>
STATUS	<i>See echo request/response frame; responseControl</i>
OK 192	<i>See extendedControl</i>
ERROR 192	<i>See fairness frame; fairnessHeader</i>
LENGTH 192	<i>See flush frame</i>
PHY_DATA.request	<i>See topology checksum (TC) frame; checksumStatus</i>
OUTPUT_FRAME..... 191	res1
LENGTH 191	<i>See ATD frame</i>
PHY_LINK_STATUS.indication	res2
LINK_STATUS	<i>See ATD frame</i>
NO_DEFECT 192	responseControl
SIGNAL_FAIL..... 192	<i>See echo request/response frame</i>
SIGNAL_DEGRADE..... 192	responseRinglet
DEGRADE_FAIL 192	<i>See echo request/response frame; responseControl</i>
PHY_READY.indication	ri
READY_STATUS	<i>See baseControl</i>
READY 193	RI_0
NOT_READY 193	<i>See MA_DATA.indication</i>
prefs	<i>See MA_DATA.request</i>
<i>See topology and protection (TP) frame</i>	RI_1
PROT_CHANGE	<i>See MA_DATA.indication</i>
<i>See MA_CONTROL.indication</i>	<i>See MA_DATA.request</i>
protectionMode	RI_DEFAULT
<i>See echo request/response frame; responseControl</i>	<i>See MA_DATA.indication</i>
protocolType	<i>See MA_DATA.request</i>
<i>See basic data frame</i>	RINGLET_0
<i>See extended data frame</i>	<i>See baseControl; ri</i>
protStatus	
<i>See topology and protection (TP) frame</i>	
ps	
<i>See extendedControl</i>	

RINGLET_1
See baseControl; ri

ringlet_id
See MA_DATA.indication
See MA_DATA.request

ringlet0ReservedBw
See ATT_STATION_BW

ringlet1ReservedBw
See ATT_STATION_BW

rprIfIndex
See ATT_INTERFACE_INDEX

RR_DEFAULT
See echo request/response frame; responseControl; responseRinglet

RR_REVERSE
See echo request/response frame; responseControl; responseRinglet

RR_RINGLET0
See echo request/response frame; responseControl; responseRinglet

RR_RINGLET1
See echo request/response frame; responseControl; responseRinglet

S

sa
See basic data frame
See control frame
See extended data frame

saCompact
See fairness frame
See idle frame

saExtended
See extended data frame

sc
See baseControl

SC_CLASS_A
See MA_DATA.request

SC_CLASS_B
See MA_DATA.request

SC_CLASS_C
See MA_DATA.request

SC_CLASSA
See MA_DATA.indication

SC_CLASSB
See MA_DATA.indication

SC_CLASSC
See MA_DATA.indication

SD
See topology and protection (TP) frame; pse
See topology and protection (TP) frame; psw

secondaryMacAddress0
See ATT_SECONDARY_MACS

secondaryMacAddress1
See ATT_SECONDARY_MACS

SENDA
See MA_CONTROL.indication

SENDB
See MA_CONTROL.indication

SENDC
See MA_CONTROL.indication

seqnum
See topology and protection (TP) frame; pfs

sequenceNumber
See fairness differential delay (FDD) frame

service_class
See MA_DATA.indication
See MA_DATA.request

serviceDataUnit
See basic data frame
See extended data frame

SF
See topology and protection (TP) frame; pse
See topology and protection (TP) frame; psw

SIGNAL_DEGRADE
See PHY_LINK_STATUS.indication

SIGNAL_FAIL
See PHY_LINK_STATUS.indication

SINGLE_CHOKE_IND
See MA_CONTROL.indication

so
See extendedControl

source_address
See MA_DATA.indication
See MA_DATA.request

source_address_extended
See MA_DATA.indication
See MA_DATA.request

stationIpAddress
See ATT_MANAGE_ADDRESS

stationName
See ATT_STATION_NAME

STATUS
See PHY_DATA.indication

strict_order
See MA_DATA.indication
See MA_DATA.request

T

tailLatencyIn
See loop round-trip time (LRTT) request frame
See loop round-trip time (LRTT) response frame

tailLatencyOut
See loop round-trip time (LRTT) request frame
See loop round-trip time (LRTT) response frame

TOPO_CHANGE
See MA_CONTROL.indication

topology and protection (TP) frame	
<i>controlDataUnit</i>	
<i>protStatus</i>	292
<i>esw</i>	292
<i>ese</i>	293
<i>psw</i>	293
IDLE.....	293
WTR.....	293
MS.....	293
SD.....	293
SF.....	293
FS.....	293
<i>pse</i>	293
IDLE.....	293
WTR.....	293
MS.....	293
SD.....	293
SF.....	293
FS.....	293
<i>pref</i>	292
<i>wc</i>	293
<i>jp</i>	293
<i>seqnum</i>	293
topology checksum (TC) frame	
<i>controlDataUnit</i>	
<i>checksumStatus</i>	294
<i>res</i>	295
<i>cv</i>	295
<i>checksum</i>	294
<i>ttl</i>	
<i>See basic data frame</i>	
<i>See control frame</i>	
<i>See extended data frame</i>	
<i>See fairness frame</i>	
<i>See idle frame</i>	
<i>ttlBase</i>	
<i>See basic data frame</i>	
<i>See control frame</i>	
<i>See extended data frame</i>	
<i>type</i>	
<i>See ATD frame</i>	
U	
<i>userData</i>	
<i>See echo request/response frame</i>	
<i>See flush frame</i>	
<i>See organization-specific frame</i>	
W	
<i>wc</i>	
<i>See topology and protection (TP) frame; pref</i>	
<i>we</i>	
<i>See baseControl</i>	
<i>WTR</i>	
<i>See topology and protection (TP) frame; pse</i>	
<i>See topology and protection (TP) frame; psw</i>	