IEEE *Draft* P802.17/D2.0

Part 17: Resilient Packet Ring Access Method & Physical Layer Specifications—

Medium Access Control (MAC) Parameters, Physical Layer Interface, and Management Parameters

Sponsor

LAN MAN Standards Committee
of the
IEEE Computer Society

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 set down herein.

This draft, D2.0, has been modified as a result of comments made on D1.1, and is being circulated for Working Group Ballot by the P802.17 Working Group.

List of Special Symbols.

Editor's Note: *to be removed prior to final publication*

For the benefit of those who have received this document by electronic means, what follows is a list of special symbols that are produced using non-standard characters. If any of these symbols fail to print out correctly on your machine, the editors apologize, and hope that this table will at least help you to sort out the meaning of the resulting funny-shaped blobs and strokes. Note that ***this table will be removed*** *in the final publication, and is only intended to assist during draft creation and editing. The reader is directed to Clause 3 for normative definitions of symbols and operators that have significance for this draft.*

Special symbols and operators

Printed Character	Meaning	Frame 6.0 character code	Font
^	Boolean XOR	^	Times
!	Boolean NOT	ALT-033	Symbol
<	Less than	ALT-060	Symbol
≤	Less than or equal to	ALT-0163	Symbol
≠	Not equal to	ALT-0185	Symbol
≥	Greater than or equal to	ALT-0179	Symbol
>	Greater than	ALT-062	Symbol
⇐	Assignment operator	Ctrl-q \	Symbol
±	Plus or minus (a tolerance)	Ctrl-q 1	Times
°	Degrees (as in degrees Celsius)	ALT-0176	Symbol
∑	Summation	ALT-0229	Symbol
—	Big dash (Em dash)	Ctrl-q Shft-q	Times
–	Little dash (En dash)	Ctrl-q Shft-p	Times
†	Dagger	Ctrl-q Space	Times
‡	Double dagger	Ctrl-q ‘	Times
µ	Micro	Ctrl-q 5	Times
Ω	Omega	ALT-087	Symbol
λ	Lambda	ALT-0108	Symbol

Contents

Information technology—
Telecommunications and information exchange between systems—
Local and metropolitan area networks—Specific requirements—

Part 17: Resilient Packet Ring Access Method & Physical Layer Specifications

1. Overview

Editors' Notes: To be removed prior to final publication.

References:
None

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for WG review.
Draft 0.2, April 2002	Draft 0.2 for WG review, modified according to comments on D0.1.
Draft 0.3, June 2002	Draft 0.3 for WG review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for WG review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

Resilient packet ring (RPR) is a metropolitan area network (MAN) technology supporting data transfer among stations interconnected in a dual-ring configuration. RPR supports the attachment of up to 255 stations. The design of RPR is optimized for rings having up to 64 stations and a maximum circumference of 2000 kilometers.

NOTE—The 2000 kilometer figure above is a design goal and not a specific physical constraint.

Editors' Notes: *To be removed prior to final publication.*

The maximum number of stations may depend on the details of wrapped time-to-live aging protocols. Another option that is being considered is 128.

Key features of RPR include:

a) Unicast, multicast, and broadcast data transfer
b) Low-delay guaranteed rate, bounded delay committed rate, and best effort service classes
c) 50 millisecond service restoration following a single station or link failure
d) Reclamation of unused committed capacity
e) Ring capacity reuse downstream of unicast destination (spatial reuse)
f) Weighted fair access to available ring capacity
g) Reporting of multiple points of congestion (multi-choke)
h) Flow control per service class to regulate traffic introduced by clients
i) Point-of-congestion aware flow control (allowing per destination queuing in client)
j) Plug-and-play operation
k) Operations, administration, and maintenance suitable for the service provider environment
l) Fully distributed control architecture (no single point of failure)

1.1 Scope

This document is Part 17 of IEEE Std. 802, 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) Overview of the specification (Clause 1);
b) References (constituting provisions of this standard through reference in the text; Clause 2);
c) Definitions and notation (Clause 3);
d) Acronyms and their expansions (Clause 4);
e) MAC service interface specification and reference model (Clause 5);
f) MAC datapath sublayer specification (Clause 6);
g) PHY service interface specification (Clause 7);
h) Syntax and semantics of the MAC frame (Clause 8)
i) MAC fairness control entity specification (Clause 9);
j) MAC topology control entity specification (Clause 10);
k) MAC protection control entity specification (Clause 11);
l) MAC operation, administration, and maintenance (OAM) control entity specification (Clause 12);
m) MAC layer management entity (MLME) service interface specification (Clause 13);
n) Bibliography (informative; Annex A);
o) Reconciliation sublayer (RS) specification for 1Gb/s and 10Gb/s Ethernet (Annex B);
p) Reconciliation sublayer (RS) specification for SONET/DSH (Annex C);
q) Management information base (MIB) specification (Annex D);
r) IEEE 802.1D MAC bridging conformance (Annex E);
s) MAC cyclic redundancy check (CRC) calculations (informative; Annex F);
t) Illustrative C-code (informative; Annex G);
u) Spatial indications and shaping (informative; Annex H)
v) Datapath scenarios (informative; Annex I)
w) Topology discovery and protection scenarios (informative; Annex J)
x) Connectivity monitoring using echo request and response (informative; Annex K)

1.2 Layer model

The RPR layer model and its relationship to the open systems interconnect (OSI) reference model is illustrated in Figure 1.1. 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.

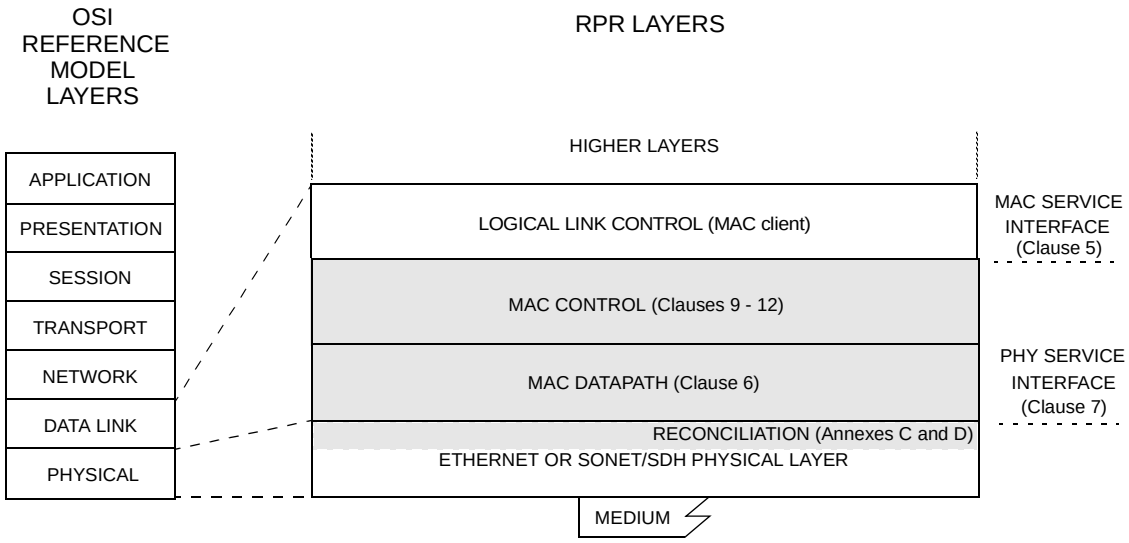


Figure 1.1—RPR layer model

NOTE—The MAC datapath sublayer occupies the same architectural layer position as the MAC sublayer described in figures 2-1 and 2-2 of IEEE Std 802.3-2000. The MAC control sublayer, described as optional in 802.3-2000, is mandatory in the RPR specification.

Editors' Notes: *To be removed prior to final publication.*

Figure 5-2 of D0.3 uses the name 'ring control sublayer' vs. the name "MAC control sublayer" and the name "access control" instead of "MAC datapath sublayer". It also shows the ReceiveFrame() function being issued by the MAC datapath to the MAC control, rather than from the MAC control to the MAC datapath (i.e. arrow points up instead of down).

The current model has MAC datapath function in both the MAC datapath sublayer and the MAC control sublayer. This naming may be confusing.

The MAC service interface supports the transfer of data from a MAC client to one or more remote peer MAC clients and the transfer of local control information from the MAC to the MAC client. The MAC control sublayer performs datapath activities not associated with a specific ringlet and control activities required to maintain the state of the MAC. The MAC control sublayer sends and receives frames with the MAC datapath sublayer. The MAC datapath sublayer supports access control and data transfer functions associated with a specific ringlet.

The PHY service interface is used by MAC datapath sublayer to transmit and receive frames on the physical medium. Distinct reconciliation sublayers specify mapping between specific PHYs and the medium independent interface (MII). This standard includes definition of reconciliation sublayers for the most commonly used PHYs and permits other reconciliation sublayers implementations that conform to the requirements in Clause 7.

1.3 Ring structure

RPR employs a dual-ring structure utilizing a pair of unidirectional counter-rotating ringlets, as illustrated in Figure 1.2. The ringlets share a common circular path but transmit signals in opposing directions. The ringlets are shown as ringlet0 and ringlet1 in Figure 1.2. The association between link and ringlet is not altered by changes in the state of links or stations on the ringlet.

Editors' Notes: *To be removed prior to final publication.*

Comment #101 (D0.3) requested the addition of 'domain' to indicate set of contiguous spans'. Is this inclusive or exclusive of the endpoint stations? The figure currently assumes 'exclusive'.

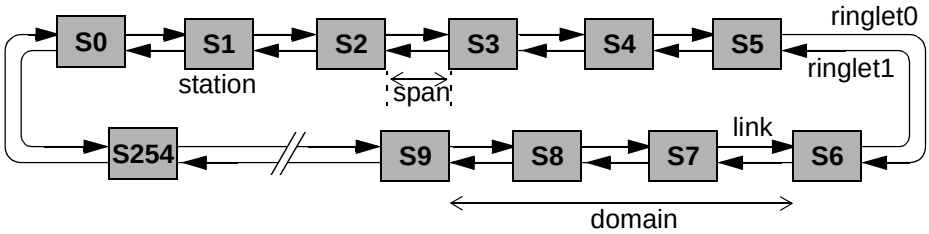


Figure 1.2—Dual-ring structure

Stations on the ring are identified by an IEEE 802 48-bit MAC address as specified in IEEE Std 802-2002, "IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture." All links on the ring operate at the same data rate, but may exhibit different delay properties. The portion of a ring bounded by adjacent stations is called a span. The span is composed of a pair of unidirectional links transmitting in opposite directions. A set of contiguous spans is known as a domain.

Station SY is said to be downstream from station SX when the output of station SX is the direct input of station SY. Thus, station S5 is the downstream neighbor of station S4 on ringlet0; similarly station S2 is the downstream neighbor of station S3 on ringlet1.

If station SY is downstream from station SX, then station SX is said to be upstream from station SY. Thus, station S3 is the upstream neighbor of station S4 on ringlet0; similarly station S4 is the upstream neighbor of station S3 on ringlet1.

Editors' Notes: *To be removed prior to final publication.*

Comment #31 (D0.3) requested removal of the sentence 'All links on the ring operate at the same data rate, but may exhibit differing delay properties.' The WG reached no resolution on this comment and it was carried forward.

1.4 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 and two MAC datapath entities, each of which is associated with a ringlet. The PHY transmitting on ringlet0 and receiving on ringlet1 is identified as the east PHY, as illustrated in Figure 1.3. 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.

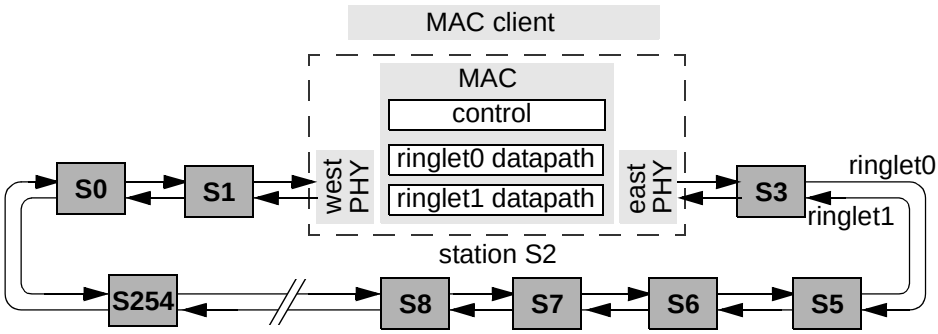


Figure 1.3—Station structure

NOTE—The RPR MAC only supports a single service interface and the service interface assumes there is a single client entity attached to that service interface. The client may contain multiple client sub-entities, but the definitions of such sub-entities are beyond the scope of this standard.

1.5 Ring operation

A ring supports the transfer of frames from a source station to a destination station associated with an individual MAC address (unicast) or to the set of stations associated with a group MAC address (multicast). A broadcast address is viewed as a special-case subset of a multicast address.

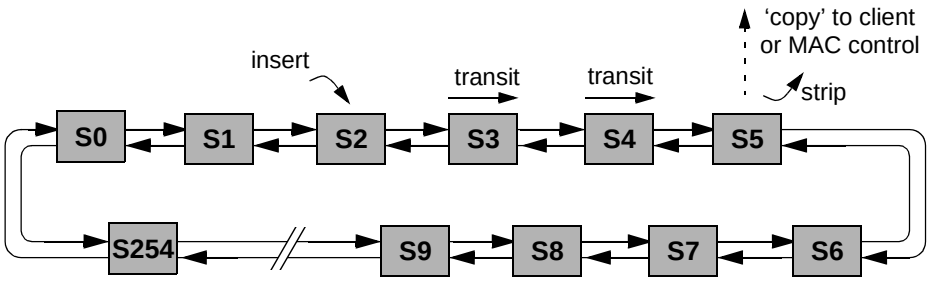


Figure 1.4—Unicast operation

In the case of a unicast transfer, a frame is inserted into the ringlet at the source station, passes through a sequence of transit stations, and is stripped from the ringlet at the destination station. The frame is also copied at the destination station for delivery to the local MAC client or MAC control entity. The frame is stripped before reaching the destination station if the time to live (TTL) value carried in the frame header expires during transit.

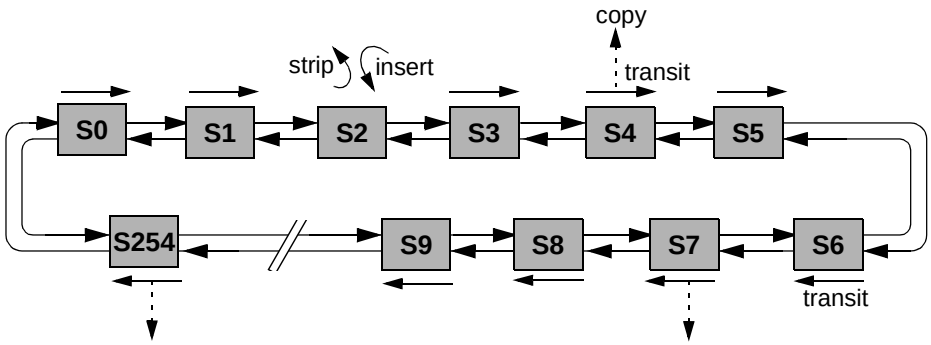


Figure 1.5—Multicast operation

In the case of a multicast transfer, the frame is inserted into one or both of the ringlets. If the frame is inserted into both ringlets, circulation of the frame proceeds independently on each ringlet. The frame transits a sequence of stations and is stripped from the ring on returning to the source station or on expiry of the TTL, whichever occurs first. The frame is copied to the local client or MAC control entity at each station with membership in the group identified by the destination address field of the frame header.

1.5.1 Concurrence

The insertion of individual frames is not synchronized among stations on the ring[1]. The transmission of a frame on an individual link is independent of frame transmissions on other links.

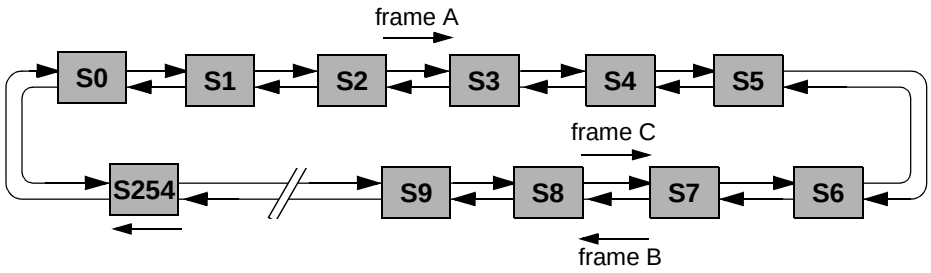


Figure 1.6—Concurrence

Figure 1.6 illustrates events that can occur concurrently, as listed below.

a) Frames A and B cross different spans of the same ringlet.
b) Frames B and C cross the same span via different ringlets.
c) Frames A and C cross different spans via different ringlets.

1.5.2 Spatial reuse

The stripping of unicast frames at the destination station implies that unicast transfers need only utilize capacity on links lying between the source and destination stations. Capacity on the remaining portion of the ringlet is available for other frame transfers. Figure 1.7 shows unicast frame A transferred from S1 to S3 via ringlet0. The frame is stripped from the ringlet at the destination S3. Some time after the arrival of frame A at S3, frame B is transferred from S4 to S5 on ringlet0. This frame can be viewed as occupying link capacity that would have been occupied by frame A had frame A not been stripped at S3. Similarly, frame C occupies the capacity that would have been occupied by frames A or B had they not been stripped at their respective destinations. The utilization of ringlet capacity on links downstream of the point of stripping, is known as spatial reuse.

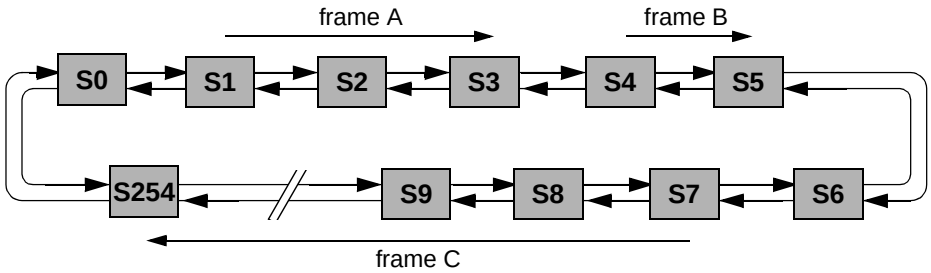


Figure 1.7—Spatial reuse

[1]As is the case, for example, with token-passing protocols such as token-ring.

Editors' Notes: *To be removed prior to final publication.*

Spatial reuse for multicast operations is an ongoing topic.
Possible wording (depending on the WG resolution of this topic) could be:
In the case of multicast, frames may be sent to all stations, for simplicity, or only the affected stations.

1.6 MAC service

The MAC service is defined by the semantics of the MAC service interface illustrated in Figure 1.8. A request (req) is associated with the direction from MAC client to MAC, while an indication (ind) is associated with the direction from MAC to 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.

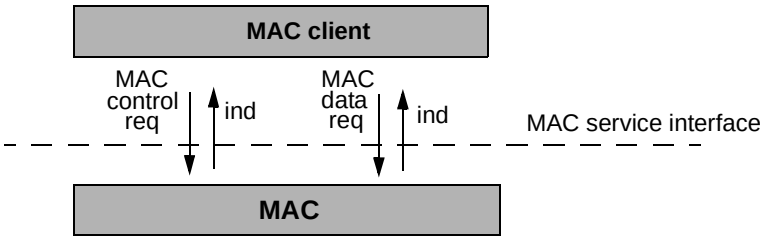


Figure 1.8—MAC service interface

1.6.1 MAC data primitives

Figure 1.9 illustrates the unicast transfer of a client-layer PDU from a client at S2 to a client at S4. In step one, a MAC data request is issued by the MAC client to the MAC at S2. The client-layer PDU is carried as a MAC-layer service data unit (SDU), identified as a parameter of the data request primitive. In step two, the SDU is encapsulated in a MAC header addressed to S4 and is transferred to S4 as a frame (MAC-layer PDU). In step three, the frame is received by the MAC at S4. The MAC header is removed and the remaining payload is transferred to the local MAC client via a MAC data indication primitive. As in step one, the primitive carries the SDU as a parameter. From the perspective of the client, the three steps are logically equivalent to the transfer of a client-layer PDU from one client to the other, as shown.

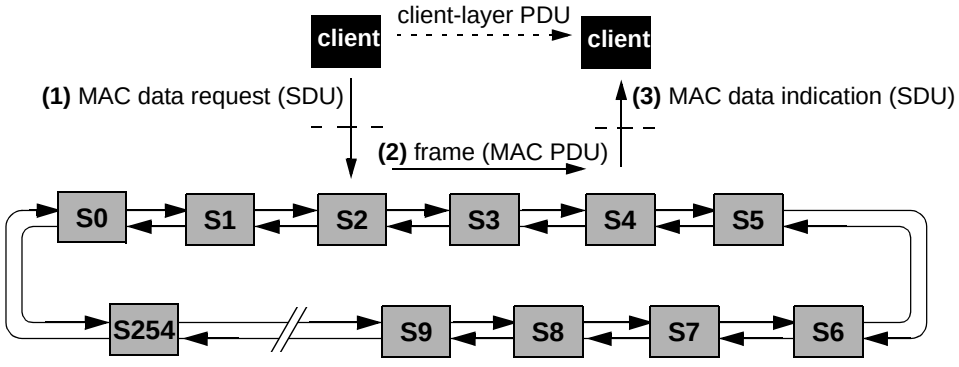


Figure 1.9—MAC data primitive

1.6.2 Service classes

In addition to the SDU, the MAC data primitive identifies a service class with which the data transfer is associated. The supported service classes are summarized in Table 1.1. The classA service provides low delay transfer of traffic up to its allocated rate. Traffic above the allocated rate is rejected. The classB service provides bounded delay transfer of traffic at or below a allocated committed rate (in-profile) and a best-effort transfer of data above the committed rate (out-of-profile). Ring capacity required to support the classA service and in-profile classB service is allocated via provisioning and these services can be characterized as provisioned services. The provisioning activity insures that the aggregate service commitment on the ring does not exceed the ring capacity. ClassC and out-of-profile classB services use capacity not required by provisioned services and are characterized as opportunistic services. A fairness algorithm (FA) is provided to allow sharing of the available capacity among stations in proportion to provisioned weights. The fair rate computed by the FA is used by a shaper or shapers associated with the service class in order to regulate access for traffic of that service class to the ring.

Table 1.1—Service classes

class	service	rate	delay	rate limiting
A	low delay guaranteed rate	not exceeding guaranteed rate	low	provisioned
B	bounded delay committed rate	at or below committed rate (in-profile)	bounded	
		above committed rate (out-of-profile)	unbounded	subject to fairness
C	best-effort			

l

Editors' Notes: *To be removed prior to final publication.*

Draft 1.1 comment #12 requested delay-time clarification, but has not been resolved.

1.7 MAC control primitive

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.

1.7.1 Flow-control

One use of the MAC control primitive 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 (FA).

The MAC enforces bandwidth constraints, by refusing to allow its client to transmit more than the station's allowed rate.

For the allocated services, a status indication is sent from 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 nearest 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 (HOL).

1.8 MAC architecture

Figure 1.10 provides a single station view of the MAC architecture. The MAC entity associated with the station is shown to contain one instance of the MAC control sublayer function and two instances of the MAC datapath sublayer function. Each instance of the MAC datapath serves one of the two ringlets. The MAC control entity sends frames to, and receives frames from, each of the two MAC datapath instances.

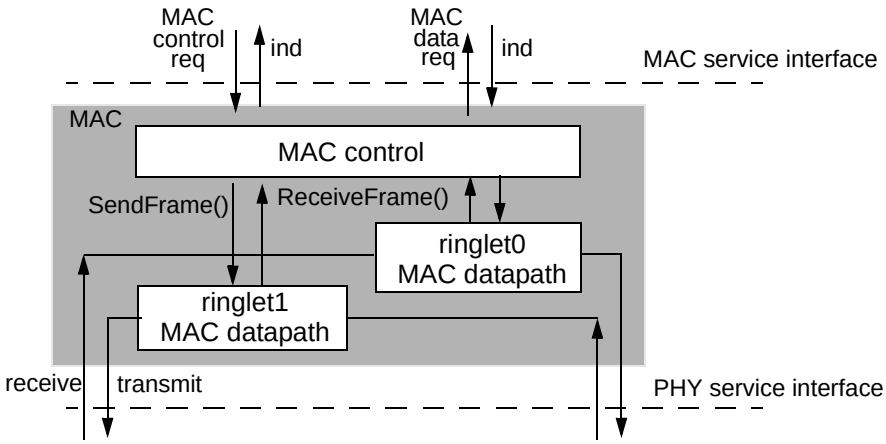


Figure 1.10—Single station view of MAC architecture

Figure 1.11 provides an end to end view of the MAC entities involved in a data transfer from S1 to S3 via S2 as a transit station. Data transfer exercises MAC control functions only at those stations where data is introduced by a client or data is delivered to a client. 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).

Editors' Notes: *To be removed prior to final publication.*

Description will be added to accompany this figure. e.g.
S1 ringlet independent: MAC service interface receive processing; encapsulation; ringlet selection
S1 ringlet0: shaping; staging; PHY transmission
S2 ringlet0: receive from west PHY; transit queuing; shaping; transmit on east PHY
S3 ringlet0: receive from west PHY; select frame for local delivery; select frame for strip;
S3 ringlet independent: frame collection; decapsulation; MAC service interface processing

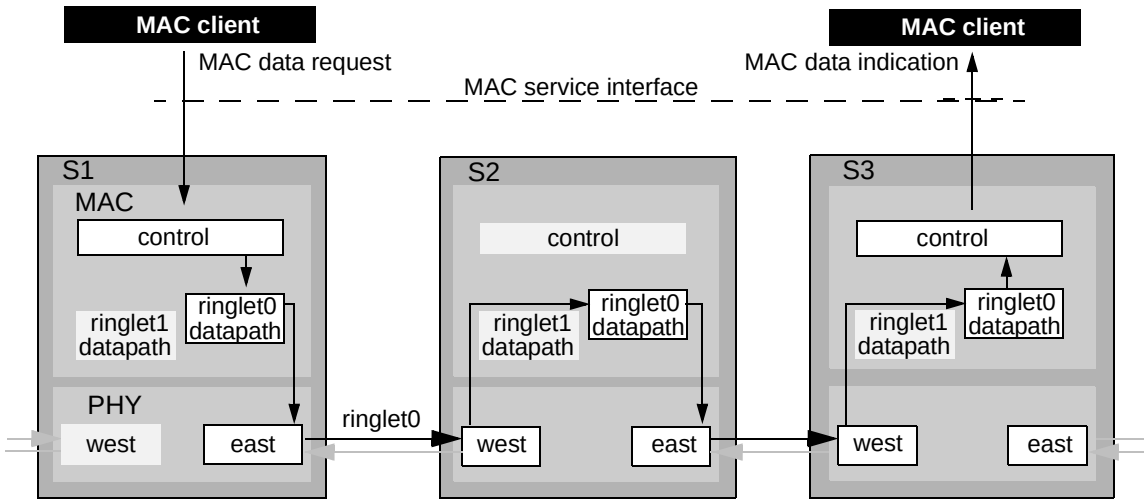


Figure 1.11—End-to-end view of MAC architecture

1.8.1 MAC control

Editors' Notes: *To be removed prior to final publication.*

The placement of ring selection will be clarified when material is available.

The MAC control sublayer, shown in Figure 1.12, supports control activities necessary to maintain the state of the MAC and datapath activities not identified with a particular ringlet. The control activities include fairness control, topology discovery, ringlet selection, protection, and operations, administration, and maintenance. The control activities are distributed among stations on the ring in order to survive any single point of failure. Control entities in a station must communicate with peer control entities in other stations using the services of the MAC datapath sublayer. The datapath activities of the MAC control sublayer include:

a) MAC service interface processing.
b) Ringlet selection and frame collection.
c) Encapsulation/decapsulation (including error processing).
d) Control frame processing.

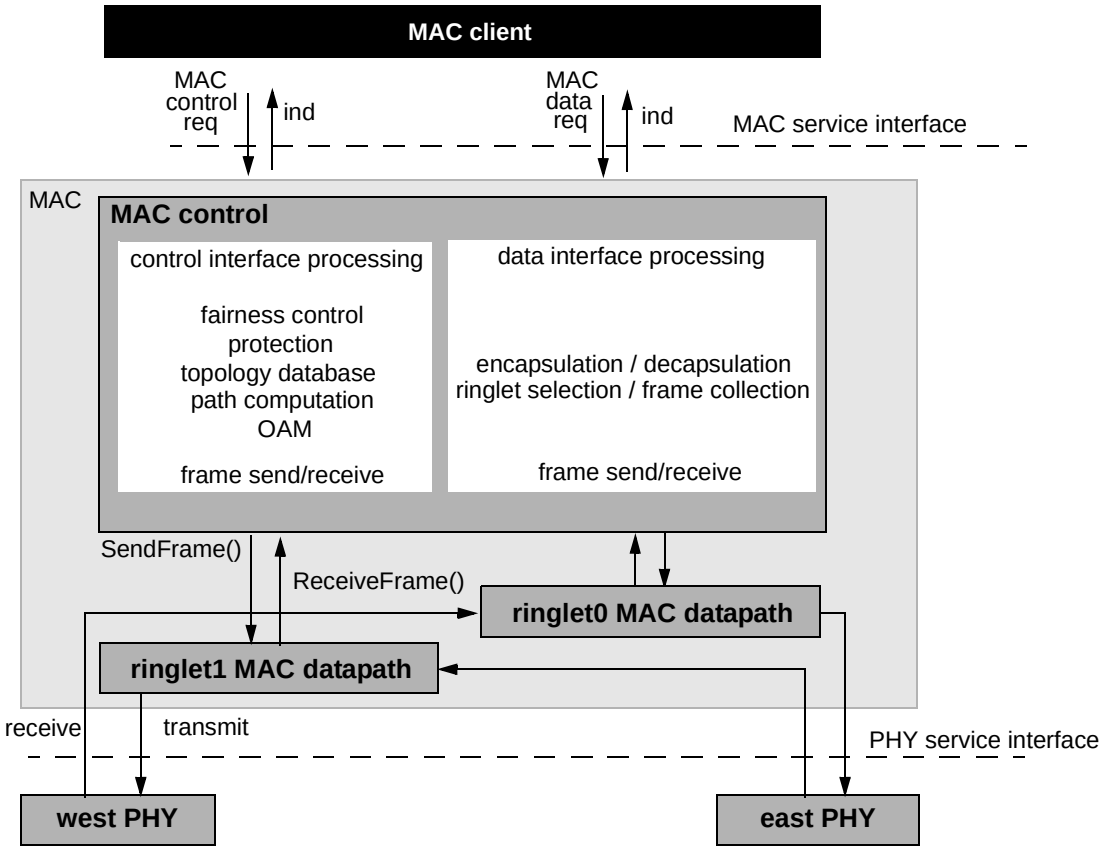


Figure 1.12—MAC control architecture

1.8.2 Ringlet selection

Editors' Notes: *To be removed prior to final publication.*

Add material here when available to describing ringlet selection as requested by comment #47.

1.9 MAC datapath

The MAC supports a distinct instance of the MAC datapath sublayer for each ringlet. Functions performed by the MAC datapath include (1) the shaping of traffic (per service class) to regulate access to the shared ring medium, (2) the staging of frames at the source station and the queuing of frames at transit stations allowing frames to be held until conditions for transmission are satisfied, (3) the selection of frames for delivery to the local client, and (4) the selection of frames for stripping from the ring. The datapath functions are illustrated in figure 1.13.

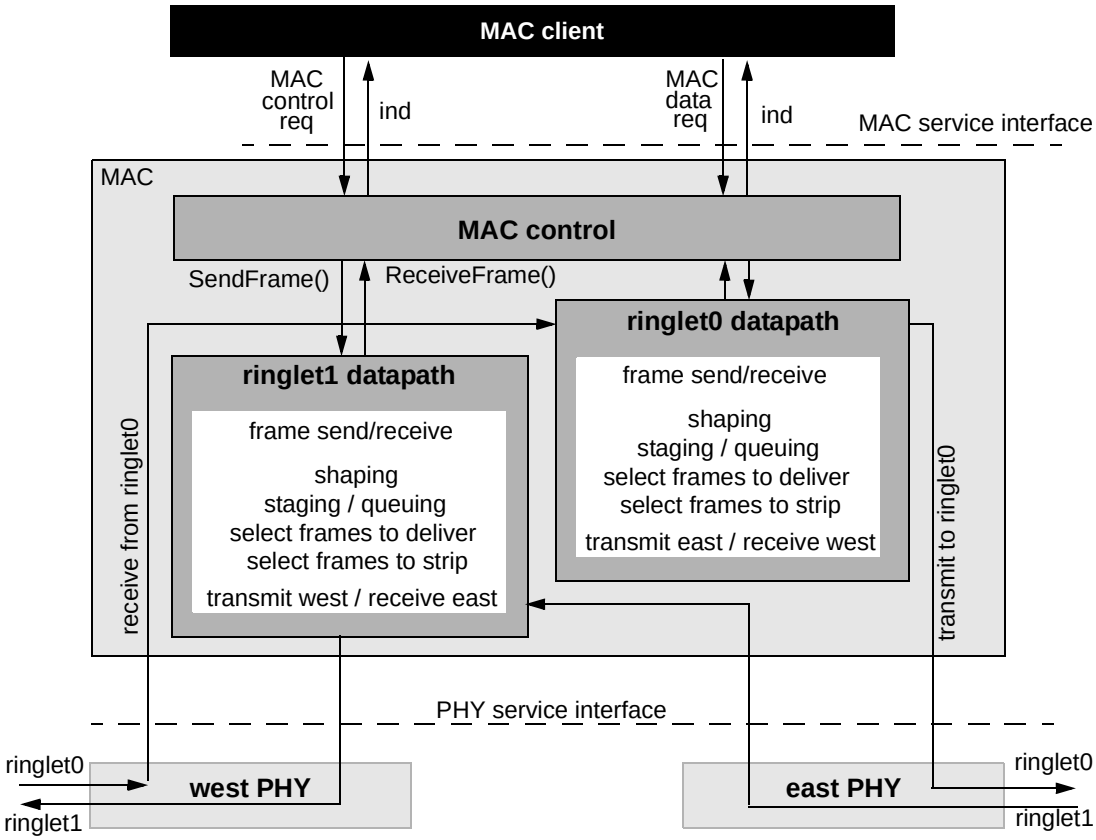


Figure 1.13—MAC datapath architecture

1.10 MAC data paths

The MAC datapath provides check hardware (that determine when frames are stripped) and transit queues (that hold ill-timed pass-through frames), as shown in Figure 1.14. Stations have the option of providing only a small primary transit queue (PTQ) or supplementing the primary transit queue with a typically larger secondary transit queue (STQ), as illustrated in the left and right of Figure 1.14.

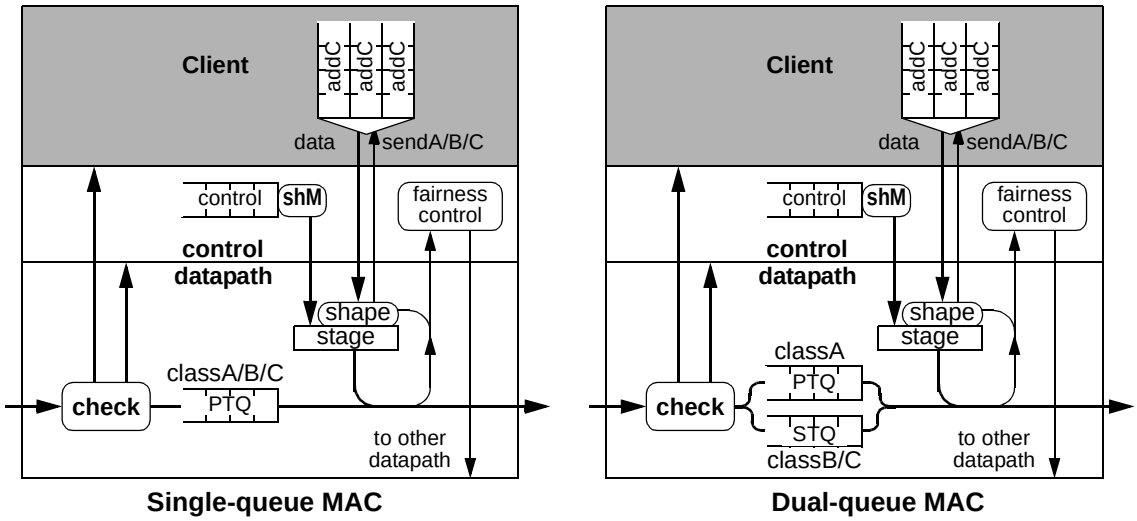


Figure 1.14—MAC data paths

Frames are transmitted from a MAC-resident *stage* buffer, 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 buffer, rather than when the staged frames are actually transmitted.

All traffic is shaped before transmission. Control frames are shaped to a low transmission rate, to limit the jitter impact on client-supplied classA transmissions. Client-supplied data frames are similarly 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 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. The classA and classB traffic can be similarly flow-controlled, although non-binary versions of the *sendA*/*sendB* indications have been deferred to future revisions of the standard.

1.11 Transparent bridging

1.11.1 Bridged frame formats

RPR is intended to support transparent bridging as well as efficient router-to-router communications. Ethernet frame forwarding involves supplementing the header with additional RPR-local fields, as illustrated in Figure 1.15.

The ring nature of the interconnect mandates the presence of effective RPR-local station identifies, *destinationStationID* and *sourceStationID*. The ***destinationStationID*** determines where the frame is removed; the ***sourceStationID*** is useful for system-maintenance purposes (topology discovery and bridge-route learning). For simplicity and efficiency, these station identifiers are specified as 8-bit hop-count distances, as opposed to 48-bit MAC addresses.

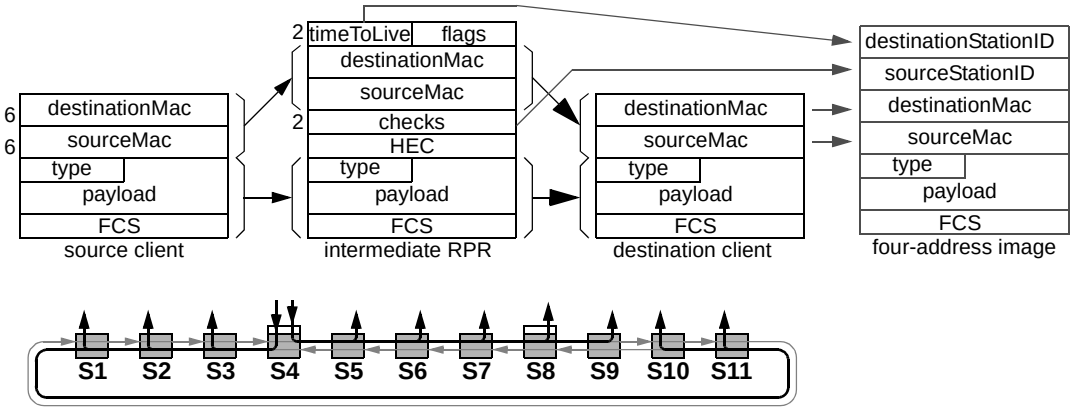
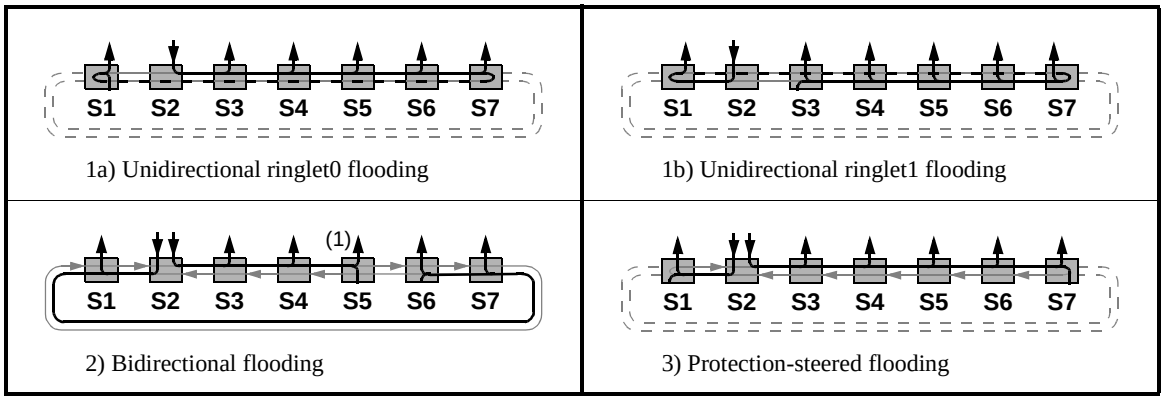


Figure 1.15—Bridged frame formats

1.11.2 Flooded transmissions

Transparent bridging is based on the concept of flooding remote frames to all bridges, to ensure that the frame can be flooded to all possible remote nets. Unidirectional flooding involves sending the frame to one's upstream station, on either of the (1a) ringlet0 or (1b) ringlet1 datapaths, as illustrated in Figure 1.16.

Figure 1.16—Flooded transmission



Bidirectional flooding involves (2) sending the frame over non-overlapping ringlet0 and ringlet1 segments, which balances the ringlet loading and while minimizing hop-count distances. This standard supports the use of either unidirectional or bidirectional transmissions, although this option selection is beyond the scope of this standard.

Protection-steered flooding (3) must also be supported, since this topology can result from a failed link. This is effectively a special case of bidirectional flooding, where the center-point location is selected to avoid the failed-link location.

1.11.3 Strict and relaxed transmissions

Two types of frame transmission are supported, as listed below. The adherence to these requirements by the RPR MAC is not only applicable to RPR MACs servicing 802.1D/Q compliant clients (i.e. bridges), but also RPR MACs servicing other clients (e.g., host, router, etc.).

a) Strict: Adheres to the 802.1 frame reorder, duplication, and loss requirements:
   1) There is no guarantee that service data units (SDUs) are delivered.
   2) Duplication of user data frames is not permitted.
   3) Reordering of frames with the same three parameters is not permitted:
      i) MAC address of the destination.
      ii) MAC address of the source.
      iii) User priority associated with the VLAN.

b) Relaxed: Normally adheres to the 802.1 frame reorder, duplication, and loss requirements. Duplication and reordering may occur during transient protection events and topology changes.

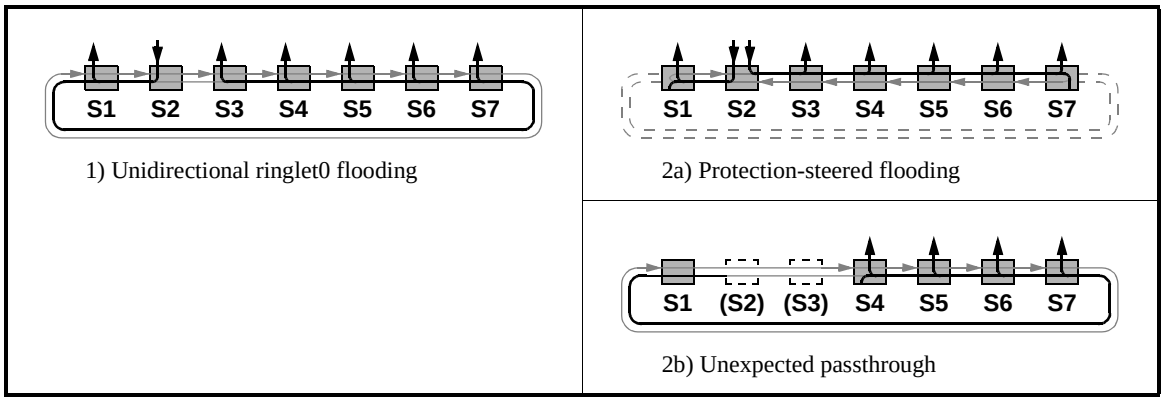
The complexity of supporting strict transmission is particularly burdensome during station or link failures. With relaxed mode, a minimal amount of reorder and/or duplication can be encountered, however the likelihood of frame loss is decreased.

The processing of strict and relaxed 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. Distinct discard strategies are required to address the two possible failure scenarios:

a) Severed link. A link goes from available-to-severed or severed-to-available states: frames are discarded until the distribution of new-topology is confirmed by RPR-local stations.
b) Passthrough: A station goes from operational-to-passthrough or passthrough-to-operational states: frames with inconsistent hop-count-distance parameters are discarded.

The severed-link strategy addresses link failure that force a (1) unidirectional loop transmissions to become (2a) bidirectional-steered chain transmissions, as illustrated in Figure 1.17. The passthrough strategy addresses failures that cause (2b) one or more stations to disappear during flooding operations.

Figure 1.17—Flooded transmission



1.12 Bandwidth allocation

Editors' Notes: *To be removed prior to final publication.*

The means of allocating the classA and classB traffic, assuring the allocation is consistent across the ring, and communicating the allocated amounts to the stations controlling the links over which the traffic transits is left to the OAM section drafters.

Note that 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.

Whether bandwidth is allocated globally uniform or differently for each link based on spatial awareness of the traffic being allocated is an implementation decision.

The use of CIR 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.

1.12.1 Reclaiming allocated bandwidth

The classA capacity is designated as reclaimable, although reclaiming of the classA0 portion is less efficient than reclaiming of the classA1 portion. Reclaimed traffic can be used to support classB excess rate traffic or classC traffic, when the capacity is not in use by the nominal classA owner.

The quantity of reclaimable capacity is designated so as not to interfere with classA service requirements.

1.12.2 Single-queue uniform allocation

A uniform-allocation single-queue S1 station has allocated levels of classA and classB traffic, as illustrated in Figure 1.18. In this simplest of allocation examples, a (1) uniform worst-case traffic flow is assumed. Uniform allocation makes no distinction between (1) classA traffic sent from S1-to-S2 and classA traffic sent from S1-to-S4. Similarly, uniform allocation makes no distinction between (2) 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 (3) cumulative bandwidth profile of client-visible bandwidth allocations. Within a single-queue MAC, the bandwidth advertised for (4) the classA service consists of subclassA0 allocated bandwidth, since a second transit queue is necessary to support subclassA1.

1.12.3 Dual-queue uniform allocation

A uniform-allocation dual-queue S1 station has allocated levels of classA and classB traffic, as illustrated in Figure 1.19. Uniform allocation makes no distinction between (1) classA traffic sent from S2-to-S3 and classA traffic sent from S2-to-S1. Similarly, the allocation of (2) classB traffic makes no distinction between traffic sent from S2-to-S4 and traffic sent from S2-to-S1.For each station, the sum of class profiles forms a (3) cumulative bandwidth profile of client-visible classA and classB bandwidth allocations. The (4) classA service consists of the allocation of subclassA0 and subclassA1 bandwidths, where the levels of supportable subclassA1 bandwidths are proportional to the size of the secondary transit queue.

Figure 1.18—Single-queue station-S1 allocations

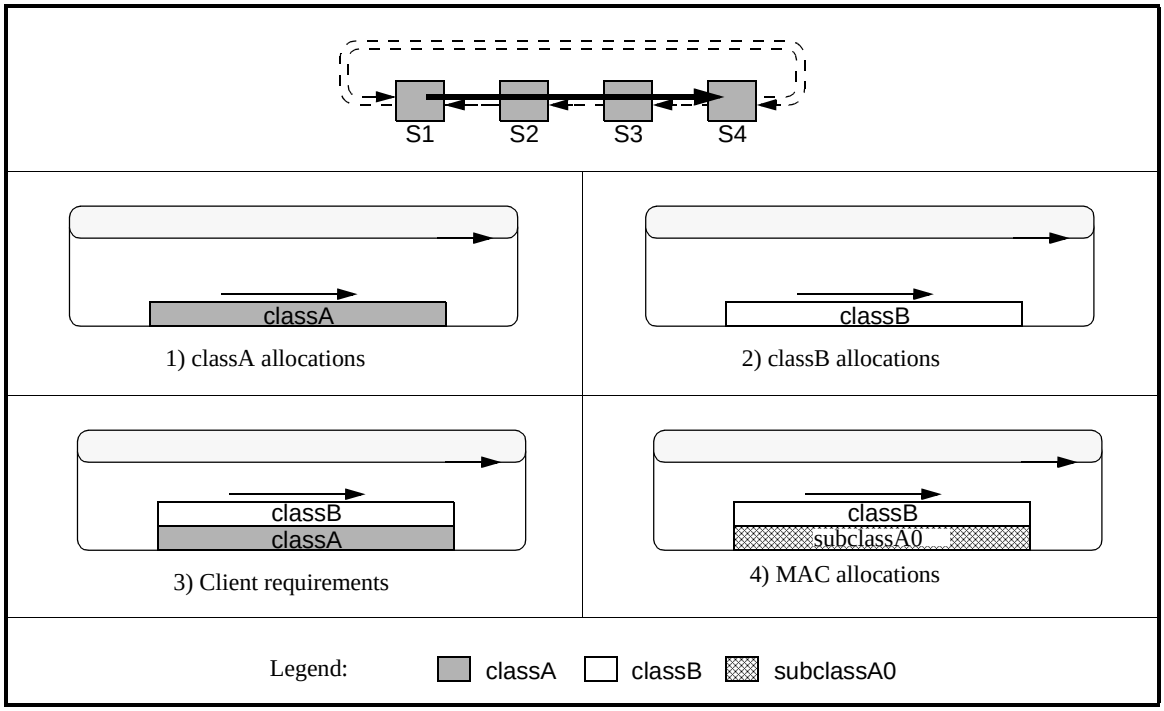
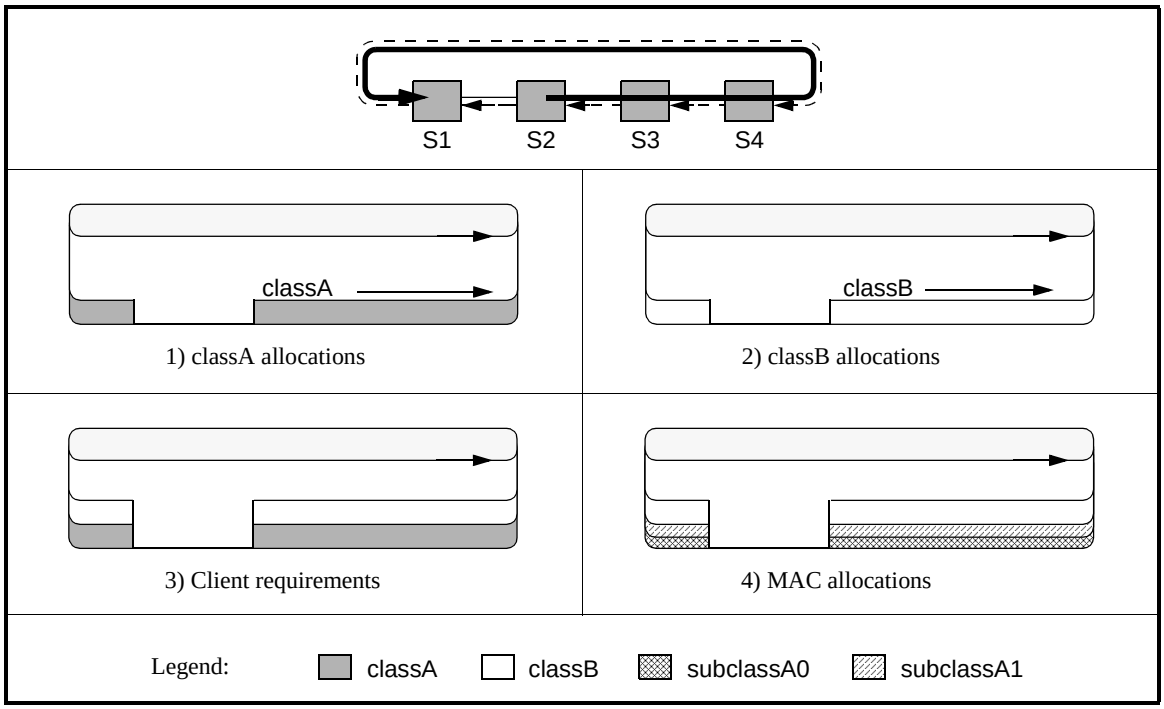


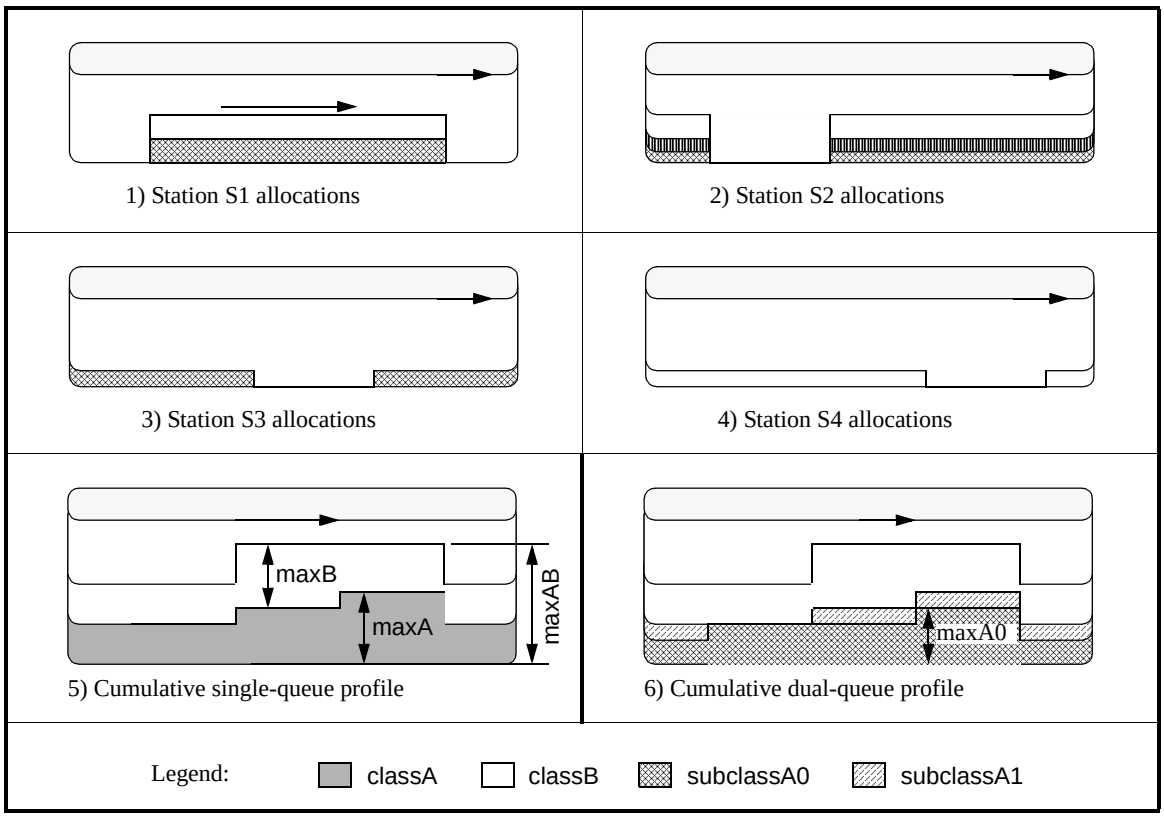
Figure 1.19—Dual-queue station-S2 allocations



1.12.4 Cumulative bandwidth allocations

Each station has its own (1,2,3,4) cumulative class profile, as illustrated in Figure 1.20. These can be combined (5) into a ringlet class profile. Within consistent ringlet profiles, the sum of subclassA0, subclassA1, and classB profiles (5) shall be less than any individual link capacity.

Figure 1.20—Cumulative bandwidth allocations



For uniform rate control, the cumulative class profile yields *maxA* and *maxA0*, the worst-case allocated classA and subclassA0 segments respectively. This information is used to rate-limit each station's classB and classC transmissions, with the intent of sustaining downstream classA transmissions.

Editors' Notes: *To be removed prior to final publication.*

The following needs to be better clarified, to better justify the conclusion that follows.

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* shall 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*.

1.13 Fairness

1.13.1 Equal-weighted fairness

A station is not permitted to use more than a weighted fair share of available capacity for the insertion of fairness eligible traffic when congestion has been detected on a ringlet. This restriction prevents a station from utilizing 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 1.21 illustrates an example of unregulated access with upstream stations advantage. In the unregulated case on the left, stations S1 and S2 can utilize all available capacity during periods of congestion, preventing stations further downstream from inserting opportunistic traffic. In the case of weighted fair access, shapers restrict the use of available capacity during periods of congestion, eliminating the access advantage of upstream stations.

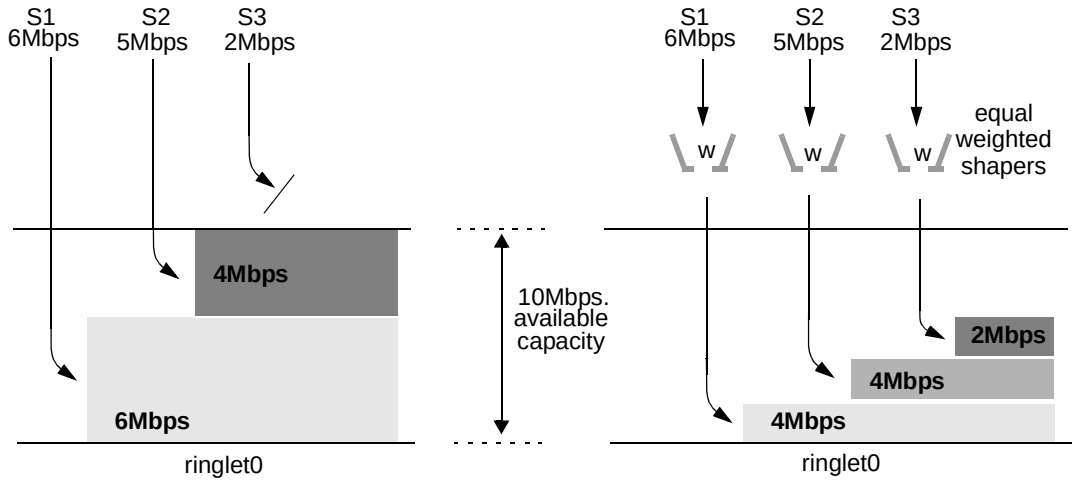


Figure 1.21—Unregulated vs. equal-weighted fair access

1.13.2 Unequal-weighted fairness

Fairness is not necessarily based on equal bandwidths, but 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 1.22.

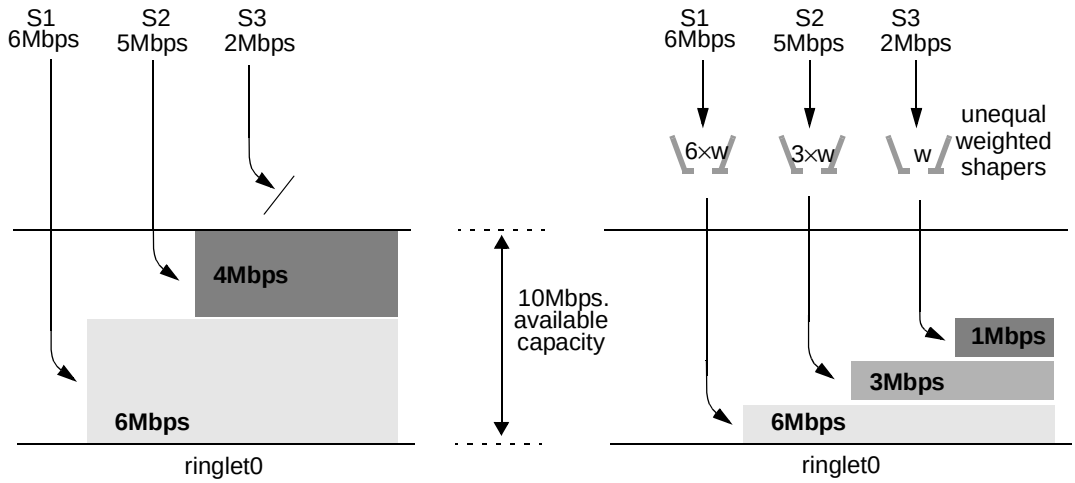


Figure 1.22—Unregulated vs. unequal-weighted fair access

1.13.3 Fairness message distribution

Shaping parameters for fairness eligible traffic are computed using a distributed fairness algorithm (FA). The FA relies on fairness messages that are circulated periodically on the ringlet opposing that of the associated data traffic, as illustrated in Figure 1.23. The single-choke fairness message is processed at each station and carries the identity of the most congested station encountered and the time averaged ingress rate, or fair rate, reported by that station. A second type of fairness message is broadcast periodically by all stations. This fairness frame is passed to the FCU for processing and the relevant information is passed to the client to support multi-choke fairness.

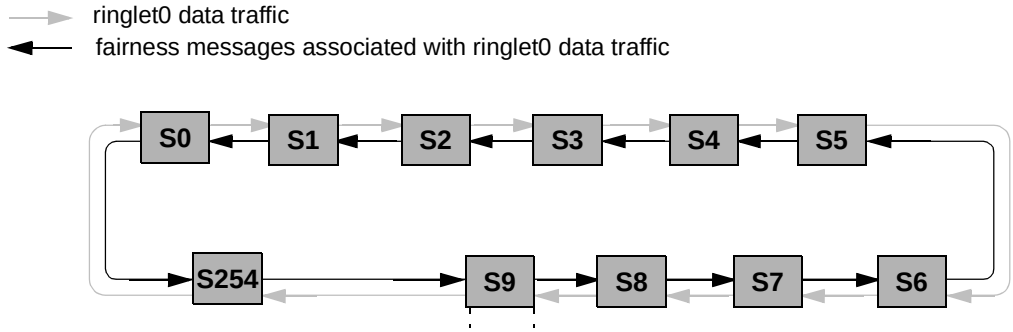


Figure 1.23—Fairness messages

1.14 Queuing options

The terms cut-through and store-and-forward describe options for the processing of pass-through traffic. Implementations use either protocol; store-and-forward is simpler but cut-through has the possibility of improved performance.

1.14.1 Store-and-forward

A store-and-forward implementation delays packet forwarding (1) while the frame header has been received, as illustrated by the sequence in Figure 1.24. Furthermore, the entire frame (2) is queued before any portion of the frame can be transmitted downstream. Once a full frame is available, that frame can be transmitted (3) downstream.

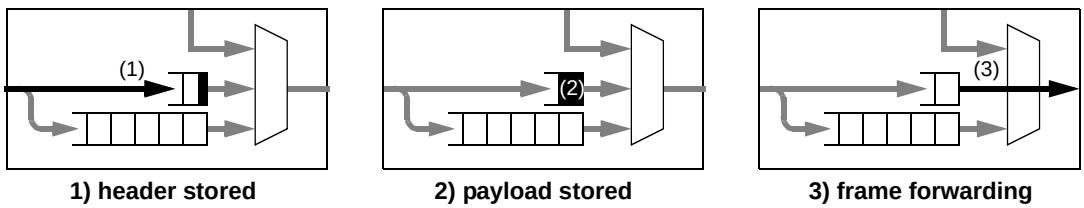


Figure 1.24—Store and forward

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.

1.14.2 Cut through

Cut-through processing allows packet forwarding to begin after the frame header has been received, as illustrated by the sequence in Figure 1.25. Transmission is stalled while the header portion of a packet (1) is

received. The header transmission then starts (2a) while the payload portion of the packet (2b) is being received. The transmission of the cut-through packet (3) continues until the entire frame has been transmitted.

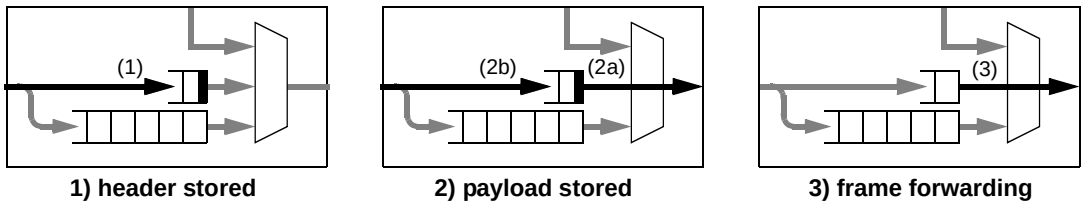


Figure 1.25—Cut-through

The advantage of a cut-through implementation is reduced latency: frame forwarding can begin after only the header has been received, even though its size may be unknown.

1.15 Protection

The RPR dual-ring topology ensures that an alternate path between source station and destination station(s) is available following the failure of a single span or station. Protection methods, involving passthrough, steering, and wrapping are available to limit service interruption to 50 milliseconds. Passthrough is optional and described in 1.15.1.

Steering is supported by all stations. Steered traffic is discarded at the point of failure. To avoid continued discards, the source station redirects unicast traffic to the ringlet that retains connectivity to the destination. Multicast or broadcast traffic is normally directed to both ringlets, so as to reach all stations on the ring.

Wrapping is an optional capability that is only activated when all stations on the ring support this capability. 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.

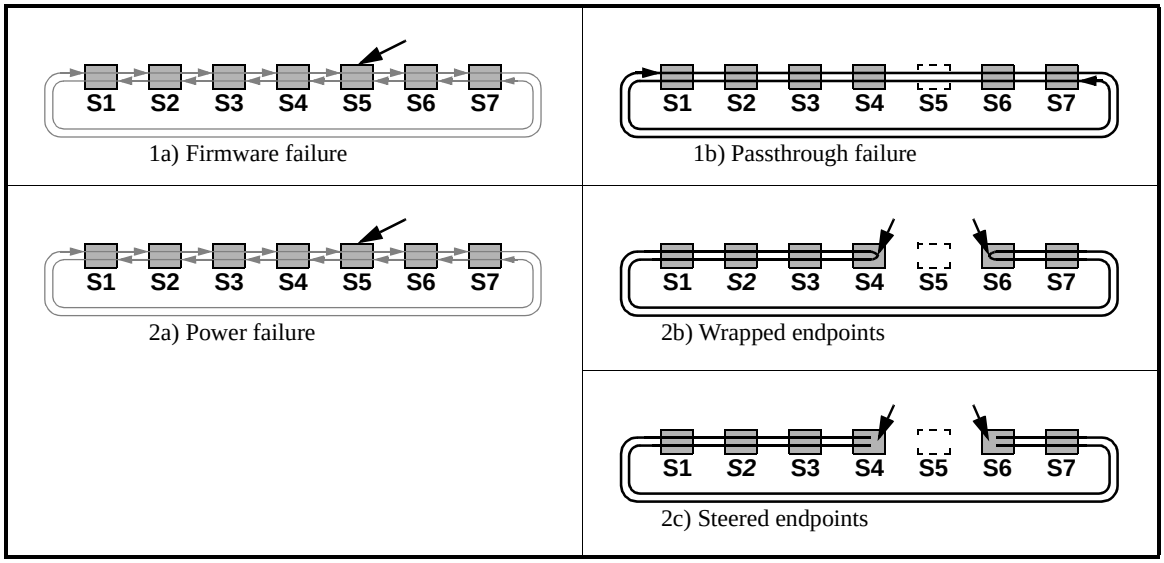
1.15.1 Protected stations

Stations can detect internal hardware/firmware corruption through consistency checks and heartbeat monitors. When failures are (1a) thus detected, the station has the option of entering a passthrough mode. While in passthrough mode, the station behaves as (1b) an active otherwise-invisible repeater, as illustrated in Figure 1.26.

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

Passthrough mode is, of course, not always possible; when a station suffers a power loss, active repeating functions can also be lost. Thus, recovery actions sometimes involve having adjacent stations isolate the failed stations. Failed-station isolation can involve “wrapping” (2b) endpoint traffic if all stations are wrap capable. Alternatively, isolation can involves (2c) discarding endpoint traffic, but arranging to “steer” traffic so that endpoint is never reached.

Figure 1.26—Isolating failed stations



1.15.2 Protected links

A single link may detect SIGNAL_FAILURE or SIGNAL_DEGRADE conditions or can be directed to generate a FORCED_SWITCH or MANUAL_SWITCH condition. These protection conditions affect the resulting RPR topology, as illustrated in Figure 1.27 and Table 1.2, with the following table-entry descriptions.

Figure 1.27—Protection topologies

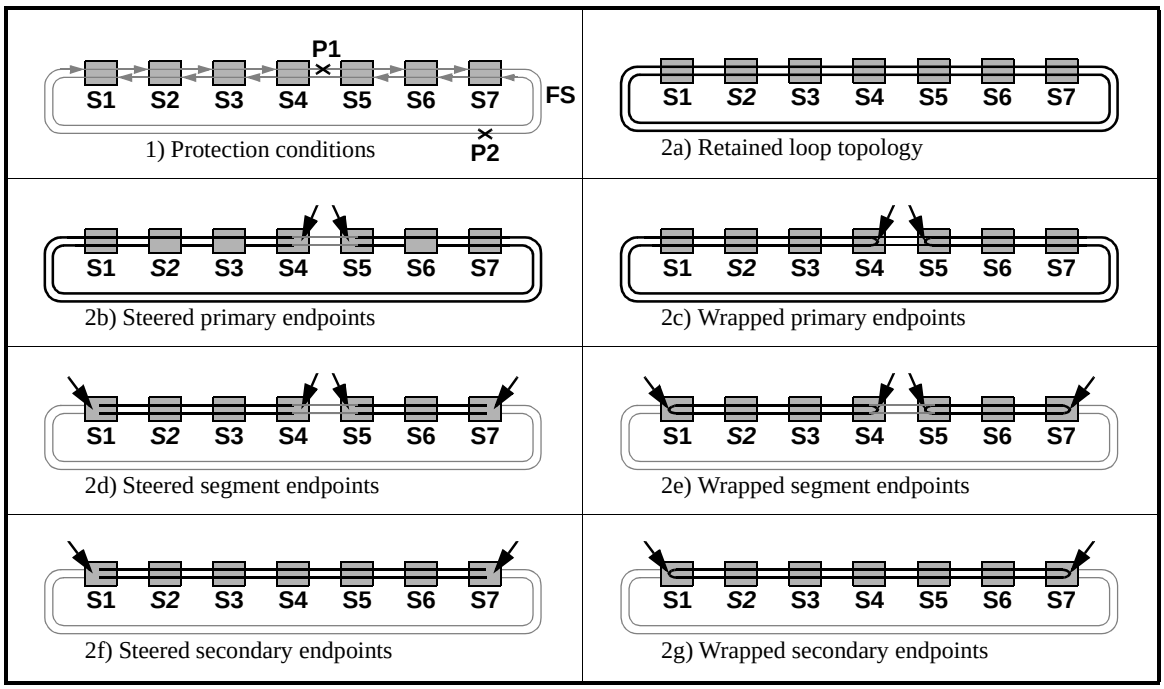


Table 1.2—Protection conditions

P1 condition	P2 condition	Row	Table 1.2 topology
SIGNAL_FAIL	SIGNAL_FAIL, FORCED_SWITCH	1	2d, 2e
	—	2	2b, 2c
FORCED_SWITCH	SIGNAL_FAIL, FORCED_SWITCH	3	2d, 2e
	—	4	2b, 2c
SIGNAL_DEGRADE	SIGNAL_FAIL, FORCED_SWITCH	5	2f, 2g
	SIGNAL_DEGRADE	6	2a
	—	7	2b, 2c
MANUAL_SWITCH	SIGNAL_FAIL, FORCED_SWITCH, SIGNAL_DEGRADE	8	2f, 2g
	MANUAL_SWITCH	9	2a
	—	10	2b, 2c

The protection conditions within the preceding figure and table are described below. These are listed in order of severity, where the most-severe condition nominally dominates.

SIGNAL_FAILURE — A major signal degradation condition that forces a link to be unused.
FORCED_SWITCH — A management directive that forces a link to be unused.
SIGNAL_DEGRADE — A minor signal degradation condition that coerces a link to be unused.
MANUAL_SWITCH — A management directive that coerces a link to be unused.

Row 1.2-1: Row 1.2-3: Two faults force a pair of breaks, thereby segmenting the RPR loop topology. Sustaining communications is hampered by the unavailability of faulted links.

Row 1.2-2: Row 1.2-4: A primary fault forces a break, converting the RPR loop into a chain topology.
Row 1.2-5: Row 1.2-8: A secondary fault forces a break, converting the RPR loop into a chain.
The intent is to sustain communications, while avoiding use of the faulted link.

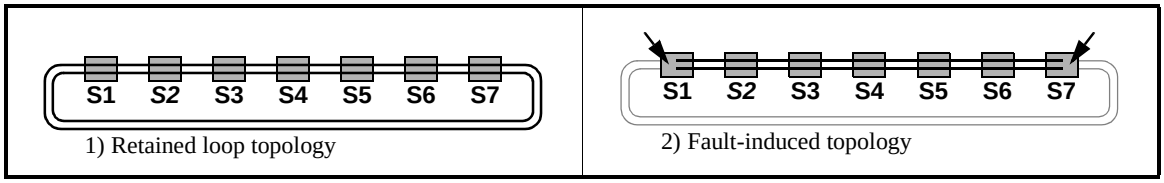
Row 1.2-6: Row 1.2-9: Multiple same-level warnings force no breaks, retaining the RPR loop topology.
The intent is to sustain communications, rather than avoiding use of the less-desirable links.

Row 1.2-7: Row 1.2-10: A single protection warning forces a break, converting the RPR loop into a chain.
The intent is to sustain communications, while avoiding use of the less-desirable link.

1.16 Topology discovery

A distributed topology discovery algorithm provides the means to construct a topology image replicated at each station. The topology image represents (1) a loop of stations or (2) a chain of stations resulting from a ring broken at one or more points as illustrated in Figure 1.28.

Figure 1.28—Loop and chain topologies



The topology discovery algorithm describes rules for the broadcast of a topology status message on the ring. The message contains information about the originating station and the current topology image of that station. Status messages are generated (1) periodically and (2) on detection of a change in station or ring status. The local topology image is determined to be complete when the topology data base is consistent.

1.17 Operations, administration, and maintenance (OAM)

Editor's Note: *To be removed prior to final publication.*

This, and some other sections, of the overview have not been completed. Contributions are solicited.

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

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

1.17.1 RPR echo transactions

1.17.1.1 Echo transaction overview

A echo is a form of confirmed transaction used to verify link integrity. The RPR client may request an echo operation (called a echo) to a specified destination with the intent of checking the reachability of that RPR destination. This echo request generates a echo response, and that response returns on the other ring. The client has the option of sending echo frames using classA, classB, or classC delivery services.

Editor's Note: *To be removed prior to final publication.*

An Nth echo request may be received before the first N echo response has been returned, so that any finite sized N-entry buffer is insufficient to guarantee lossless echoes. How should this issue be addressed?

All forms of echo-request are sent from the requester to the responder; the addressed responder station returns an echo-response. The path for the response (ringlet0 or ringlet1) depends on the arrival path and the specified mode, as illustrated in the following subclauses.

1.17.1.2 Echo response routing

The echo-request transfers information from the requester to the responder. The request may be sent on ringlet0 (1a) or ringlet1 (1b), as illustrated in Figure 1.29. The echo type and the received-ringlet effect whether the response is returned on (2a) ringlet0 or (2b) ringlet1, as illustrated in Table 1.3.

Figure 1.29—Echo paths

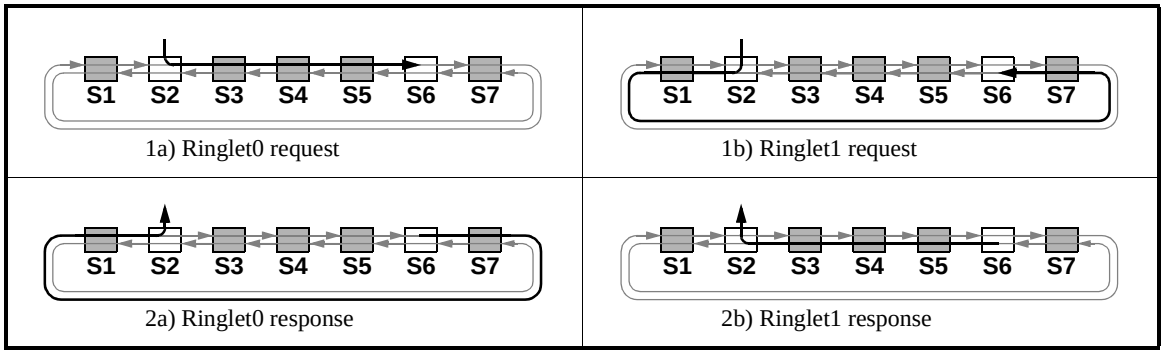


Table 1.3—Echo response routing

echo type	Request	Response
DEFAULT	Ringlet0 request (1a)	Ringlet1 response (2b)
	Ringlet1 request (1b)	Ringlet0 response (2a)
RINGLET0	Ringlet0 request (1a)	Ringlet0 response (2a)
	Ringlet1 request (1b)	
RINGLET1	Ringlet0 request (1a)	Ringlet1 response (2b)
	Ringlet1 request (1b)	

The intent of the DEFAULT type is to provide a simple mechanism for testing requester-to-responder connectivity, by sending the response in the reverse direction from the received request. The RINGLET0 and RINGLET1 types support testing of selected-ringlet communication paths.

2. Normative references

Editors' Notes: To be removed prior to final publication.

References:
None

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for WG review.
Draft 0.2, April 2002	Draft 0.2 for TF review, modified according to comments on D0.1.
Draft 0.3, June 2002	Draft 0.3 for TF review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for WG review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

Editors' Notes: *To be removed prior to final publication.*

The references appearing in this section are copied from the 'References:' portion of the prolog in each clause or annex.

It has been officially determined by the IEEE staff that publicly available OIF documents may be referenced by IEEE standards. (Note that both ITU-T and ANSI - to which the IEEE is a subsidiary body - reference OIF documents.) Please do not submit any further comments on referencing OIF documents.

Note that all of the references in this section (and their counterparts throughout the draft) are routinely updated by the IEEE editorial staff prior to issue of the standard. Therefore, there is no need to continually update references to other standards, unless absolutely required for technical issues. There is also no need to submit comments against reference versions unless an actual technical error results from the use of an older version. In particular, references to "IEEE 802.1D" and "IEEE 802.1Q" are automatically assumed to be to the most recent versions of these standards, whatever they may be.

The following standards contain provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid international standards. The registration authority for Internet numbering, including, for example MIB extensions, is the Internet Assigned Numbers Authority (IANA).[1]

IEEE Std 802-2002, IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture

IEEE Std 802.1D-1998, Information Technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Part 3: Media Access Control (MAC) Bridges

Editors' Notes: *To be removed prior to final publication.*

IEEE Std 802.1D-1998 is now amended by IEEE 802.1T and IEEE 802.1W.

[1]The contact information for IANA is as follows:
Internet Assigned Numbers Authority, 4676 Admiralty Way, Suite 330, Marina del Rey, CA 90292, USA.
+1-310-823-9358 (phone); +1-310-823-8649 (facsimile); iana@iana.org (e-mail); http://www.iana.org (Web)

IEEE Std 802.1F-1993 (Reaff 1998), IEEE Standards for Local and Metropolitan Area Networks: Common Definitions and Procedures for IEEE 802 Management Information

IEEE Std 802.1Q-1998, IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks

IEEE Std 802.3-2000, 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

IEEE Draft P802.3ae/D5.0, Supplement to Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications—Media Access Control (MAC) Parameters, Physical Layer, and Management Parameters for 10 Gb/s Operation

IETF RFC1662: PPP in HDLC Like Framing, W. Simpson, July 1994

IETF RFC2570: Introduction to Version 3 of the Internet-standard Network Management Framework, J. Case et. al., April 1999

IETF RFC2574: User based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), U. Blumenthal et. al., April 1999

IETF RFC2575: View based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), B. Wijnen 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

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

ISO/IEC 7498-1 1994, Information technology—Open Systems Interconnection—Basic Reference Model: The Basic Model

ISO/IEC 9899: 1999, Programming languages - C

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

ISO/IEC 15802-2: 1995 [ANSI/IEEE Std 802.1B-1992 and IEEE Std 802.1k-1993], Information technology —Telecommunications and information exchange between systems—Local and metropolitan area networks —Common specifications—Part 2: LAN/MAN Management

ISO/IEC 15802-3: 1998 [IEEE Std 802.1D, 1998 Edition), Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications —Part 3: Media Access Control (MAC) bridges

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

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

ITU-T Recommendation G.7041, Generic Framing Procedure

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

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

3. Terms, definitions, and notation

Editors' Notes: To be removed prior to final publication.

References:
None

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for WG review.
Draft 0.2, April 2002	Draft 0.2 for TF review, modified according to comments on D0.1.
Draft 0.3, June 2002	Draft 0.3 for TF review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for WG review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

Editors' Notes: *To be removed prior to final publication.*

Definitions for 'revertive' and 'non-revertive' are to be added to this clause when available from Clause 10.

3.1 Terms and definitions

3.1.1 agent: [802.3-2000 1.4.30 (modified)] A network management entity (NME) used to configure a station and/or collect data describing the operation of that station.

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

3.1.3 allocated bandwidth: Bandwidth that may be subject to reclamation, but which will be made available when requested.

3.1.4 available capacity: Ring capacity not required to support provisioned service and, consequently, available to support opportunistic service.

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

3.1.6 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.

3.1.7 bit error ratio (BER): [802.3-2000 1.4.47] The ratio of the number of bits *received* in error to the total number of bits *transmitted*.

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

3.1.9 broadcast: [802.5-1998 1.3.10] The act of sending a *frame* addressed to all *stations* on the network.

3.1.10 broadcast address: [ISO/IEC2382-25 25.01.13] A *group address* that identifies the set of all *stations* on the *network*.

3.1.11 committed information rate (CIR): [ITU I.233.1 A.8 (modified)] The rate at which the network agrees to transfer data averaged over a minimum interval of time (*Tc*).

3.1.12 committed rate measurement interval (Tc): [ITU I.233.1 A.7 (modified)] The time interval during which the user is allowed to send only the committed amount of data (*Bc*) and the excess amount of data (*Be*).

3.1.13 congested: The indication of whether the local station considers itself to be congested in trying to add traffic.

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

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

3.1.16 control frame: A *frame* carrying only *MAC* control information.

3.1.17 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.

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

3.1.19 data frame: A frame in which data is encapsulated by the MAC datapath for transfer between clients.

3.1.20 delay[1]: The time required to transfer information from one point to another.

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

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

3.1.23 domain: A set of contiguous spans including the associated stations.

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

3.1.25 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.

3.1.26 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.

3.1.27 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 must have substantially equal access to the available capacity of all links. (See also weighted fairness.)

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

3.1.29 fair rate: a rate having the property that a station has neither advantage nor disadvantage in adding data to the ringlet simply by virtue of its location on the ringlet.

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

[1]Delay and latency are synonyms for the purpose of this specification. Delay is the preferred term.

3.1.31 Flooding Scope (FS): 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.

3.1.32 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.

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

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

3.1.35 group address: [ISO/IEC2382-25 25.01.15] An *address* that identifies a group of *stations* on a *network.*

3.1.36 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.

3.1.37 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'.

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

3.1.39 in-profile: Traffic that is within allowed traffic parameters.

3.1.40 individual address: [ISO/IEC2382-25 25.01.14] An *address* that identifies a particular *station* on a *network.*

3.1.41 insert[2]**:** The placement of a *frame* on a ringlet by the source *station*.

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

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

3.1.44 layer management entity (LME): [8802-6-1994] The entity in a layer that performs local management of a layer. The LME provides information about the layer, effects control over it, and indicates the occurrence of certain events within it. Note that the term LME can be prefixed with the name of the layer as in the case of the MAC LME (MLME).

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

3.1.46 local area network (LAN): [adapted from IEEE 100 (C/DIS) 1278.2-1995, 1278.3-1996[3]] A communications network designed for a user premises, typically not exceeding a few kilometers in length, and characterized by moderate to high data transmission rates, low delay, and low bit error rates.

3.1.47 logical link control (LLC) sublayer: [C/LM 8802-5-1992s] 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.*

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

---

[2] (C/LM) 11802-4-1994 is not applicable. That definition uses the term to specify device, rather than frame, insertion in the ring.

[3] 'moderate sized geographic area' replaced by 'user premises, typically not exceeding a few kilometers in length,'

3.1.49 MAC layer management entity (MLME): see LME.

3.1.50 major priority inversion: Any reclamation-caused effect that violates any equal or higher class' guarantee.

3.1.51 management information base (MIB): [802.3-2000 1.4.163] A repository of information to describe the operation of a specific network device.

3.1.52 maximum transfer unit (MTU): [IEEE 100 610.7-1995 (modified)] The largest frame (comprising payload and all header and trailer information) that can be transferred across the network.

3.1.53 medium (plural: **media):** [IEEE 100 (C/LM) 8802-6-1994, 802.5-1998 1.3.34] The material on which *information signals* may be carried; e.g., optical fiber, coaxial cable, and twisted-wire pairs.

3.1.54 medium access control (MAC) sublayer: [IEEE 100 (C/LM) 8802-5-1995] The portion of the data link layer that controls and mediates the access to the network medium. In this standard, the MAC sublayer comprises the MAC datapath sublayer and the MAC control sublayer.

3.1.55 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.

3.1.56 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.

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

3.1.58 multicast address: [ISO/IEC2382-25 25.01.16]. A *group address* that identifies a subset of the *stations* on a *network.*

3.1.59 multi choke: The characteristic of an algorithm, whereby it can work with multiple choke or congestion points (of different values) at the same time.

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

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

3.1.62 opportunistic: The ability to utilize capacity that becomes available.

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

3.1.64 out-of-profile: Traffic that exceeds specified traffic parameters.

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

3.1.66 passthru: A condition of a MAC such that frames pass through it as if it were part of the segment between the stations on either side of it.

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

3.1.68 physical layer (PHY): [(C/LM) 8802-5-1995] 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*.

3.1.69 plug-and-play: The requirement that a *station* perform *transit*, *strip*, and *ring* control activities without operator intervention except for what may be needed for connection to the *ring*. The station may additionally *copy* and *insert frames*.

3.1.70 police: The enforcement of traffic parameters.

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

NOTE—This term should be prefixed with information to distinguish the type of port.

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

3.1.73 protocol data unit (PDU): [802.5-1998 1.3.46] Information delivered as a unit between peer layer entities that contains control information and, optionally, data.

3.1.74 protocol implementation conformance statement (PICS): 1.3.47: A statement of which capabilities and options have been implemented for a given Open Systems Interconnection (OSI) protocol.

3.1.75 QTag prefix: [802.3-2000 1.4.222 modified] The first four bytes of an Ethernet-encoded tag header, as defined by IEEE P802.1Q.

3.1.76 quality of service (QoS): One or a combination of measurable properties defining the requirements of a given data service.

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

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

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

3.1.80 reserved bandwidth: The amount of bandwidth that must be kept available (i.e., is not subject to reclamation). This is a subset of the allocated bandwidth.

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

3.1.82 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*.

3.1.83 ringlet0: The RPR ringlet where frames are launched in the clockwise direction.

3.1.84 ringlet1: The RPR ringlet where frames are launched in the counter clockwise direction.

Editors' Notes: *To be removed prior to final publication.*

The above definitions for "ringlet0" and "ringlet1" have implementation-specific connotations.

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

3.1.86 service data unit (SDU): [802.5-1998 1.3.59] Information delivered as a unit between adjacent layer entities that may also contain a PDU of the *upper layer.*

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

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

3.1.89 single choke: The characteristic of an algorithm, whereby it can accommodate only one choke or congestion point at one time.

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

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

3.1.92 span: The portion of a *ring* bounded by two adjacent stations.

3.1.93 spatial reuse: The concurrent transfer of data frames on nonoverlapping portions of a ringlet.

3.1.94 station: [IEEE 100 1073.3.1-1994, 1073.4.1-1994, 8802-5-1995 (modified)] A device attached to a *network* for the purpose of *transmitting* and *receiving* information on that *network*.

3.1.95 station management (SMT): [802.5-1998 1.3.66] The conceptual control element of a station that interfaces with all of the layers of the station and is responsible for the setting and resetting of control parameters, obtaining reports of error conditions, and determining if the station should be connected to or disconnected from the medium.

3.1.96 steering: A protection method where the source station directs a unicast frame to the ringlet having connectivity to the destination station or a multicast/broadcast frame to one or both ringlets.

3.1.97 strict data frame: A frame having its strictOrder bit set.

3.1.98 strip: The removal of a *frame* from a ringlet.

3.1.99 tagged frame: [802.3-2000 1.4.269 (modified)] A frame that contains a QTag Prefix.

3.1.100 time-to-live (TTL): A value carried in a frame that is decremented at each hop transited by the frame. This value is initially set by the MAC creating the frame, and indicates the maximum number of hops that the frame can make before it is removed from the ring.

3.1.101 topology: The arrangement of *stations* and links forming a network.

3.1.102 traffic class: A grouping for traffic according to processing rules within the MAC datapath sublayer.

3.1.103 transfer: [ISO/IEC2382-09 9.03.01 (modified)] 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.

3.1.104 transit: The passing of a *frame* through a *station* via the *ring*

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

3.1.106 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.

3.1.107 transmit (transmission): [(C/LM) 802.5-1989s, 8802-5-1995 (modified)] The action of a *station* placing a *frame* on the *medium*.

3.1.108 transparent bridging: [(C/LM) 8802-5-1995] A *bridging* mechanism that is transparent to the *end stations*.

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

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

3.1.111 unknown unicast: The act of flooding a frame throughout the network when the destination is not found in the learning database.

3.1.112 unreserved: A designation for traffic which is not reserved, whether allocated or not.

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

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

3.1.115 virtual destination queuing (VDQ): The maintenance of a dedicated transmit-queue for each destination.

3.1.116 virtual LAN (VLAN): [IEEE 100 (C/LM 802.1Q-1998)] A subset of the active *topology* of a *bridged local area network*.

3.1.117 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 shall be taken to imply weighted fairness.

3.1.118 wide area network (WAN): [IEEE 100 (C/DIS) 1278.2-1995] A communications network designed for large geographic areas. Sometimes called *long-haul* network.

3.1.119 wrapping: The *transmission* of a *frame* on the opposing ringlet in order to route around a fault in the ringlet.

3.2 Globally used variables

Editor's Note: *To be removed prior to final publication.*

Editors and WG members who observe that a variable is used across more than one clause are requested to submit a comment directing that it be added to this subclause.

3.2.1 *activeStations*: The number of stations active (as measured by having at least 1 transited fairness eligible frame) on the outgoing link of a ringlet during an agingInterval. (Clause 9)

3.2.2 *addRate*: Calculated. The rate, in units of bytes per AGECOEF *agingIntervals*, at which FE-marked traffic is transmitted to the ringlet from the add queues of the local station. (Clause 9)

3.2.3 *addRateCongested*: Calculated. The rate, in units of bytes per AGECOEF *agingIntervals*, at which FE-marked traffic intended for destinations downstream of the *congestionPoint*, is transmitted to the ringlet from the add queues of the local station. (Clause 9)

3.2.4 *advertisementInterval*: The interval at which FCMs are sent. (Clause 9)

3.2.5 *agingInterval*: The interval at which aging and low pass filtering functions are performed. (Clause 9)

3.2.6 *allowedRate*: Calculated. The rate, in units of bytes per AGECOEF *agingIntervals*, at which a station is allowed to add client-supplied FE-marked traffic to the ringlet. (Clause 9)

3.2.7 *allowedRateCongested*: Calculated. The rate, in units of bytes per AGECOEF *agingIntervals*, at which a station is allowed to add client-supplied FE-marked traffic intended for destinations downstream of the *congestionPoint*, to the ringlet. (Clause 9)

3.2.8 *congestionPoint*: The most congested downstream station on the ringlet reporting a congestion condition. (Clause 9)

3.2.9 *fwRate*: Calculated. The rate, in units of bytes per AGECOEF *agingIntervals*, at which FE-marked traffic is transmitted to the ringlet from the transit queues of the local station. (Clause 9)

3.2.10 *fwRateCongested*: Calculated. The rate, in units of bytes per AGECOEF *agingIntervals*, at which FE-marked traffic intended for destinations beyond the *congestionPoint*, is transmitted to the ringlet from the transit queues of the local station. (Clause 9)

3.2.11 *highThreshold*: The threshold that indicates that a MAC is experiencing some congestion. (Clause 9)

3.2.12 *hopsToCongestion*: Calculated. The number of hops between the local station and the *congestionPoint*. (Clause 9)

3.2.13 *LINK_RATE*: The actual maximum transmission rate for the outbound link. (Clause 9)

3.2.14 *localStation*: The station in which the local MAC resides. (Clause 9)

3.2.15 *lowThreshold*: The threshold that indicates that a MAC may soon start experiencing congestion. (Clause 9)

3.2.16 *MAX_STATIONS*: The maximum number of stations in an RPR ring (255). (Clause 9)

3.2.17 *nrXmitRate*: Calculated. The rate, in units of bytes per AGECOEF *agingIntervals*, at which FE-marked traffic intended for destinations downstream of the *congestionPoint*, is transmitted to the ringlet from the transit queues of the local station. (Clause 9)

3.2.18 *nrXmitRate*: Calculated. The rate, in units of bytes per AGECOEF *agingIntervals*, at which non-reserved (i.e. not in subclassA0) traffic is transmitted to the ringlet from the add queues and transit queues of the local station. (Clause 9)

Editor's Note: *To be removed prior to final publication.*

There seems to be two definitions for nrXmitRate. Which one is correct?

3.2.19 *numStations*: The number of stations that are known to be on the ringlet. (Clause 9)

3.2.20 *reservedRate*: The total amount of class A traffic that is reserved (i.e. the total amount of all subclassA0 traffic) on the outgoing link for all stations, including the local station, specified as the number of bytes per AGECOEF *agingIntervals*. (Clause 9)

3.2.21 *unreservedRate*: The rate available for the outbound link that is not reserved. (Clause 9)

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).

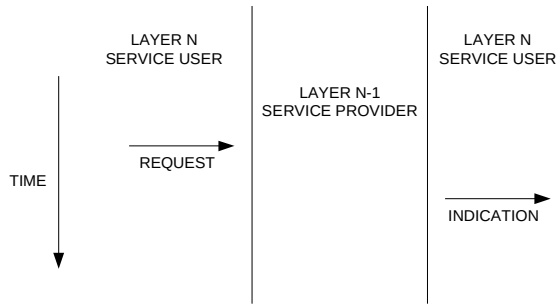


Figure 3.1—Service Definitions

Specific implementations may 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 both because they will 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:

a) REQUEST. The request primitive is passed from layer N to layer N-1 to request that a service be initiated.
b) 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 may be logically related to a remote service request, or may 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 detail 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 shall be passed to qualify the functions invoked by the primitive. Parameters indicate information available in a user/provider interaction; in any particular interface, some parameters may 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 may 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, whether embodied in formal state tables or diagrams, contain the authoritative statement 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 diagrams representing the state machines, over the descriptive text. This does not override, however, any explicit description in the text that has no parallel in the state diagrams. Where a state machine is described by both a state table and a state diagram, the table is normative and the diagram informative.

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, while any realistic implementation may place heavier emphasis on efficiency and suitability to a particular implementation technology. It is the functional behavior of any unit that must 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

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. No time period is associated with the transition from one state to the next.

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 will transition, but only if the condition is satisfied. When placed in the condition column, the symbol "—" signifies the default condition (i.e., operative when no other condition is active); when placed in the action column, it signifies that no action is to be performed. Conditions shall be evaluated in order, top to bottom, and the first condition that evaluates to a true result shall be used to determine the transition to the next state. If no condition evaluates to a true result, then the state machine shall remain in the current state.

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

Table 3.1 provides an informative example of the state table notation.

Table 3.1—State table notation example

Current state		Row	Next state	
state	**condition**		**action**	**state**
FIRST	sizeOfMacControl > spaceInPTQ	1	—	FIRST
	passM == 0	2		
	—	3	Transmit from MAC control queue	FINAL
FINAL	selected transfer completes	4	—	FIRST
	—	5	—	FINAL

Row 3.1-1: The size of the queued MAC Control frame must be less than the PTQ space.
Row 3.1-2: In the absence of MAC Control transmission credits, no action is taken.
Row 3.1-3: MAC Control transmissions have precedence over client transmissions.

Row 3.1-4: When the transmission completes, start over from the initial state.
Row 3.1-5: Until the transmission completes, remain in this state.

3.4.2 State diagram notation

State machines may also be represented diagrammatically. In state diagrams, each state that the function can assume is represented by a rectangle (see Figure 3.2). These are divided into two parts by a horizontal line. In the upper part the state is identified by a name in capital letters. The lower part contains the name of any message or signal that may be generated by the function. Actions are described by short phrases or assignments. Conditional actions are permitted.

All permissible transitions between the states of a function are represented graphically by arrows between them. A transition that is global in nature (for example, an exit condition from all states to the IDLE or RESET state) is indicated by an open arrow leading to the name of the target state. Labels on transitions are qualifiers that must be fulfilled before the transition will be taken; if no transition has a fulfilled qualifier, then the function will remain in the present state. The label UCT designates an unconditional transition. Italics should be used when referencing specific logical symbols or data values depicted in state diagrams within any accompanying descriptive text.

State transitions and sending and receiving of messages occur instantaneously. When a state is entered and the condition to leave that state is not immediately fulfilled, the state executes continuously, sending the messages and executing the actions contained in the state in a continuous manner. The symbol "⇐" (left arrow) or "=" denotes assignment of the value following the symbol to the term preceding the symbol.

Flow charts are provided for informative purposes only, unless expressly stated otherwise. Only formal state tables or diagrams are always normative; in the event of any discrepancy, formal state tables and diagrams always take precedence over flow charts.

Figure 3.2 provides an informative example of the state diagram notation.

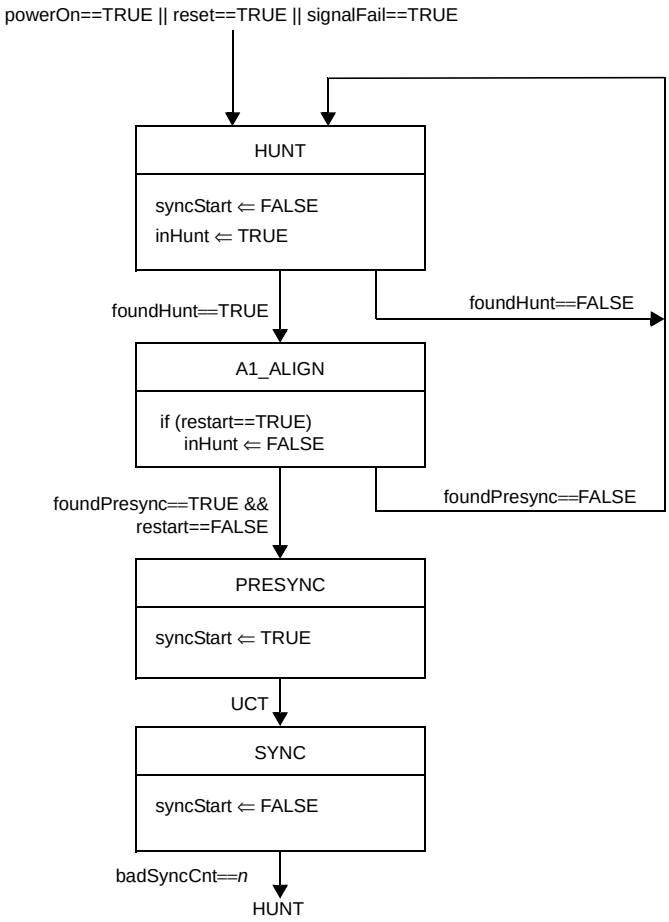


Figure 3.2—State diagram notation example

3.5 Arithmetic and logical operators

In addition to commonly accepted notation for mathematical operators, Table 3.2 summarizes the symbols used to represent arithmetic and logical (Boolean) operations.

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 “$1A_{16}$” or “11010_2”.

Table 3.2—Special symbols and operators

Printed Character	Meaning
&&	Boolean AND
||	Boolean OR
^	Boolean XOR
!	Boolean NOT (negation)
≤ or <=	Less than or equal to
≥ or >=	Greater than or equal to
==	Equal to
≠ or !=	Not equal to
⇐ or =	Assignment operator
//	Comment delimiter

3.7 Field notations

3.7.1 Use of italics

Field names or variable names (such as *level* or *MACAddress*), and sub-fields within variables (such as *thisState.level*) shall be italicized within text, 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 shall also be italicized. Variable or field names shall not be italicized within tables or figures, however, since their special meaning is implied by their column heading. Names used as nouns (e.g., subclassA0) shall also not be italicized.

3.7.2 Field conventions

This document describes values that are packetized or MAC-resident, such as those illustrated in Table 3.3.

Table 3.3—Names of fields and subfields

Name	Description
newCRC	Field within a register or frame
thisState.level	Sub-field within field thisState
thatState.rateC[n].c	Sub-field within array element rateC[n]

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

A sub-field of a field is referenced by suffixing the field name with the sub-field name, separated by a period. For example, *thisState.level* refers to the sub-field *level* of the field *thisState*. This notation may be

continued in order to represent sub-fields of sub-fields (e.g., *thisState.level.next* is interpreted to mean the sub-field *next* of the sub-field *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.ringControl* is used to indicate the *ringControl* 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 shall not be generated; values within reserved fields (whether zero or nonzero) shall be ignored.

3.7.3 Field value conventions

This document describes values of fields. For clarity, names are associated with each of these defined values, as illustrated in Table 3.4. A symbolic name, consisting of upper case letters with underscore separators, allows other portions of this document to reference the value by its symbolic name, rather than binary or hexadecimal value.

Table 3.4—*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. For devices conforming to this version of this standard, reserved-value fields shall not be generated by devices; a reserved-value field shall be processed as though the field value were not supported. The intent is to ensure default behaviors for future-specified features.

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 3.3, 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.

Figures shall be 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.

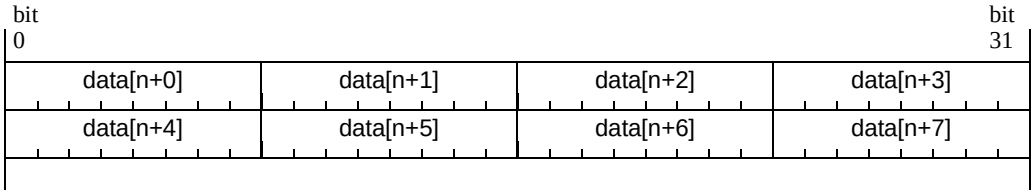


Figure 3.3—Bit numbering and ordering

3.9 Byte sequential formats

Figure 3.4 provides an illustrative example of the conventions to be used for drawing frame formats and other byte sequential representations. These representations shall be 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 shall be drawn contiguously such that the transmission order across fields is from top to bottom. The example shows that *timeToLive*, *flags*, and *destinationMACAddress* are 1-, 1- and 6-byte fields, respectively, transmitted in order starting with the *timeToLive* field first. As illustrated on the right hand side of Figure 3.4, 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.

NOTE—Only the left-hand diagram in Figure 3.4 is required for representation of byte-sequential formats. The right-hand diagram is provided for explanatory purposes only, for illustrating how a multi-byte field within a byte sequential representation is expected to be ordered.

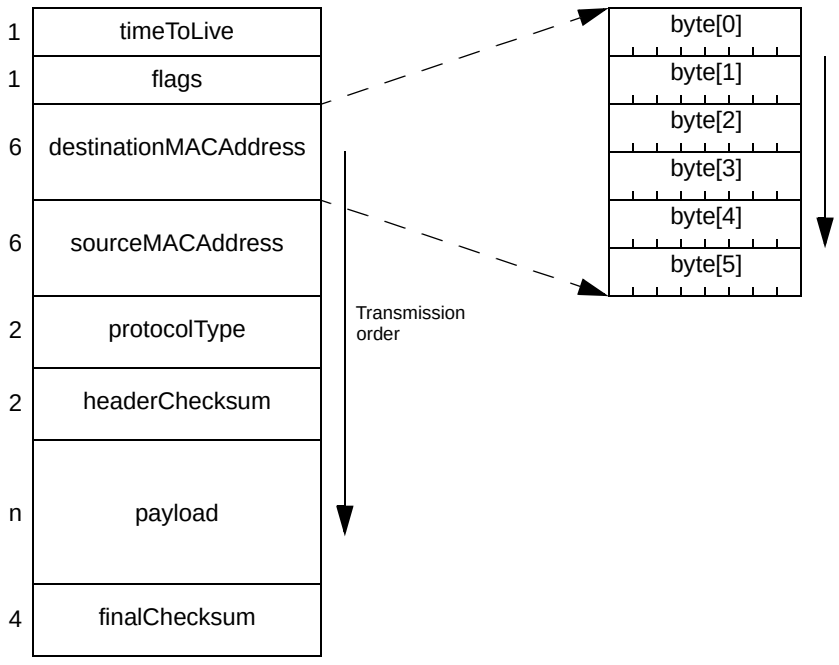


Figure 3.4—Byte sequential field format illustrations

4. Abbreviations and acronyms

Editors' Notes: To be removed prior to final publication.

References:
None

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for WG review.
Draft 0.2, April 2002	Draft 0.2 for TF review, modified according to comments on D0.1.
Draft 0.3, June 2002	Draft 0.3 for TF review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for WG review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

Editors' Notes: *To be removed prior to final publication.*

The entries appearing in this section are copied from the 'Abbreviations:' portion of the prolog in each clause or annex.

Each acronym in this section should be defined (i.e., spelled out in full) when first used in this draft Standard, as specified in 13.6 of the IEEE Standards Style Manual. For example, the first use of BER should be spelled out as follows: " . . . bit error ratio (BER) . . .". All Section Editors are requested to review their clauses and ensure that this rule is uniformly followed.

This standard contains the following abbreviations and acronyms:

AIS	alarm indication signal
BER	bit error ratio
CC	continuity check
CIR	committed information rate
CoS	class of service
CRC	cyclic redundancy check
DA	destination address
dLOC	loss of continuity failure defect
EIR	excess information rate
FA	fairness algorithm
FE	fairness eligible
FCM	fairness control message
FCS	frame check sequence
FCU	fairness control unit
FE	fairness eligible
FIFO	first in first out
FS	forced switch
GERS	gigabit ethernet reconciliation sublayer
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
IND	indication
ISO	International Organization for Standardization
ITU	International Telecommunication Union
LAN	local area network
LCP	link control protocol
LLC	logical link control
LME	layer management entity
LOC	loss of continuity failure
LOF	loss of frame
LOS	loss of signal
LR	line rate
MAC	medium access control
MAN	metropolitan area network
MC	multi choke
MFS	maximum frame size
MIB	management information base
MLME	MAC layer management entity
MS	manual switch
MTU	maximum transfer unit
NE	network element
OAM	operations, administration, and maintenance
OSI	open systems interconnect
PCS	physical coding sublayer
PDU	protocol data unit
PHY	physical layer
PICS	protocol implementation conformance statement
PLME	physical layer management entity
PMA	physical medium attachment
PMD	physical medium dependent
POS	packet over SONET
PPM	parts per million
PPP	point to point protocol
PTQ	primary transit queue
QoS	quality of service
R	reserved
RDI	remote defect indication
REQ	request
RFC	request for comment
RI	ringlet identifier
RPR	resilient packet ring
RS	reconciliation sublayer
RTT	roundtrip time
SA	source address
SC	single choke (Clause 9)
SC	service class (Clause 6)

Editors' Notes: *To be removed prior to final publication.*

The use of the same acronym (SC) with different meanings is highly deprecated. It is strongly recommended that an alternative acronym be found for one or the other term.

SD	signal degrade
SDH	synchronous digital hierarchy
SDU	service data unit
SF	signal fail
SLA	service level agreement
SME	station management entity
SNMP	simple network management protocol
SONET	synchronous optical network
SPI	system packet interface
SRS	SONET reconciliation sublayer
STQ	secondary transit queue
TTL	time-to-live
UCT	unconditional transition
VLAN	virtual LAN
VDQ	virtual destination queuing
WAN	wide area network
WDM	wavelength division multiplexing
WE	wrap eligible
WIS	WAN interface sublayer
WTR	wait to restore
XAUI	10 gigabit attachment unit interface
XGMII	10 gigabit media independent interface

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

Editors' Notes: To be removed prior to final publication.

References:
None

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for WG review.
Draft 0.2, April 2002	Draft 0.2 for TF review, modified according to comments on D0.1.
Draft 0.3, June 2002	Draft 0.3 for WG review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for WG review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

5.1 Scope

This clause provides an overview, specifies the services provided by the MAC sublayer, including the MAC control sublayer and the MAC datapath sublayer (see shaded portions of Figure 5.1), and provides a reference model for the MAC sublayer. Higher-layer clients may 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.

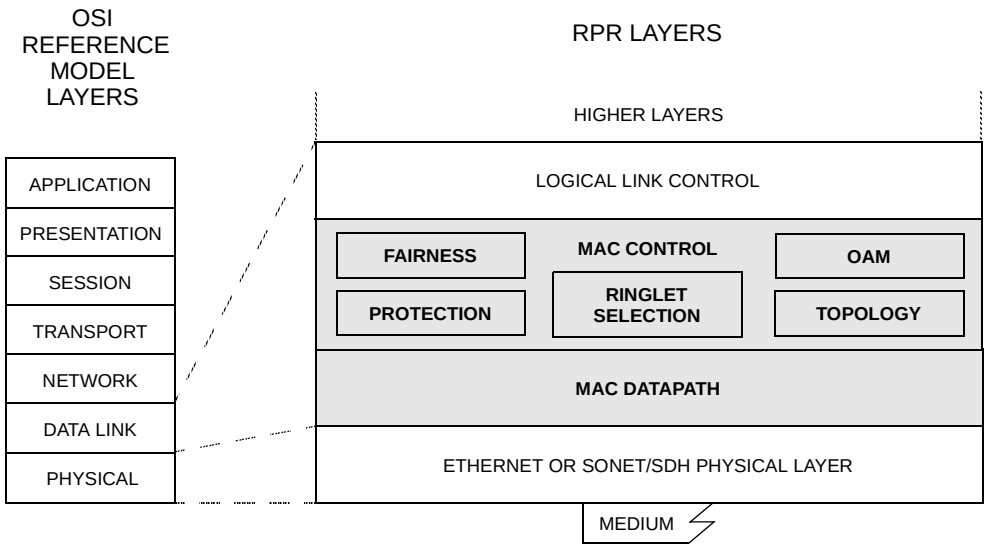


Figure 5.1—RPR layer diagram

5.2 Overview of MAC services

The services provided by the MAC sublayer allow:

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 resilient packet ring parameters with local MAC entities.
c) The relay entity in a bridge to exchange data with local MAC entities in the bridge.

The RPR MAC provides two types of frame transmission service, both of which shall be supported:

1) Strict: Adheres to the 802.1 frame reorder, duplication, and loss requirements. That is,

   i) There is no guarantee that all Service Data Units (SDUs) are delivered.
   ii) Reordering of frames with a given user priority for a given combination of destination address and source address is not permitted.
   iii) Duplication of user data frames is not permitted.

   In general, the complexity associated with supporting strict transmission is particularly required during station or link failure operations.

2) Relaxed: Adheres to the 802.1 frame reorder, duplication, and loss requirements during "normal" ring operation. 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:
   i) After ring (link or station) restoration events.
   ii) Topology and status database of stations on the ring are not synchronized.
   iii) Station failure resulting in passthru behavior. That is, frames are sent through the transit path without *timeToLive* decrement or any other frame processing/stripping rules being applied.
   iv) Compound ring (link or station) failures resulting in segmented chains.
   v) Rapid cascading ring failures.

The MAC sublayer presents a service interface for the exchange of MAC client PDUs between MAC client entities. The MAC service interface supports service classes denoted classA, classB, and classC.

For all service classes, the MAC service interface provides per ringlet indications to the MAC client of whether traffic can or cannot currently be accepted. For service class classC, the MAC service interface also provides the number of hops to the nearest congested station. Each service class is rate controlled to prevent the client from transmitting more traffic than was allocated or allowed by fairness, as applicable. The service classes are described in 6.4.

The information that flows between the MAC and the client layer is specified formally in the primitives described in 5.3.

5.2.1 Service class classA

ClassA service provides an allocated, guaranteed data rate and a low end-to-end delay and jitter bound. Within this class, there is a mechanism to reserve some or all of the allocated bandwidth.

> ***Editors' Notes:*** *To be removed prior to final publication.*
>
> *The means of allocating and reserving bandwidth will be documented in the LME clause. The means of advertising the reserved bandwidth will be documented in the Topology clause. This service description will be amended to refer to the appropriate text when it is available.*

Allocated bandwidth is invisible to the RPR fairness algorithm. ClassA traffic is not subject to the fairness algorithm at ingress to the ring or when transiting through the ring. Therefore, the Fairness Eligible (FE) bit in the RPR header must always be set to 0 on classA traffic.

ClassA traffic moves through the primary transit path in each station as it propagates around the ring. Operation of the primary and secondary transit paths, and single-queue and dual-queue models, are described in Clause 6.

5.2.2 Service class classB

ClassB service provides an allocated, guaranteed data rate, optional additional data rate that is not allocated or guaranteed, and bounded end-to-end delay and jitter for the traffic within the allocated rate.

ClassB has similarities to the classA service in that frame transmission rates within the allocated rate profile are guaranteed a bounded delay and jitter, although with higher bounds than for classA frames. However, traffic is treated differently with respect to the fairness algorithm depending on whether it meets its rate profile. 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. This traffic is essentially invisible to the fairness algorithm in the same way that classA packets are.

ClassB traffic also has similarities to classC service, explained below, in that traffic beyond the allocated rate profile is subject to the fairness algorithm, and is marked by the MAC as such with the fairness eligible (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.

In a single-queue implementation, classB traffic moves through the primary transit path, regardless of whether the frame is marked fairness eligible or not. In a dual-queue implementation, classB traffic moves through the secondary transit path, regardless of whether the frame is marked fairness eligible or not.

5.2.3 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 is always subject to the fairness algorithm, and is marked by the MAC as such with the fairness eligible (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.

5.3 MAC services to the client layer

Four service primitives are defined for the client interfaces.

— MA_DATA.request
— MA_DATA.indication
— MA_CONTROL.request
— MA_CONTROL.indication

The service primitives are shown in context in Figure 5.2.

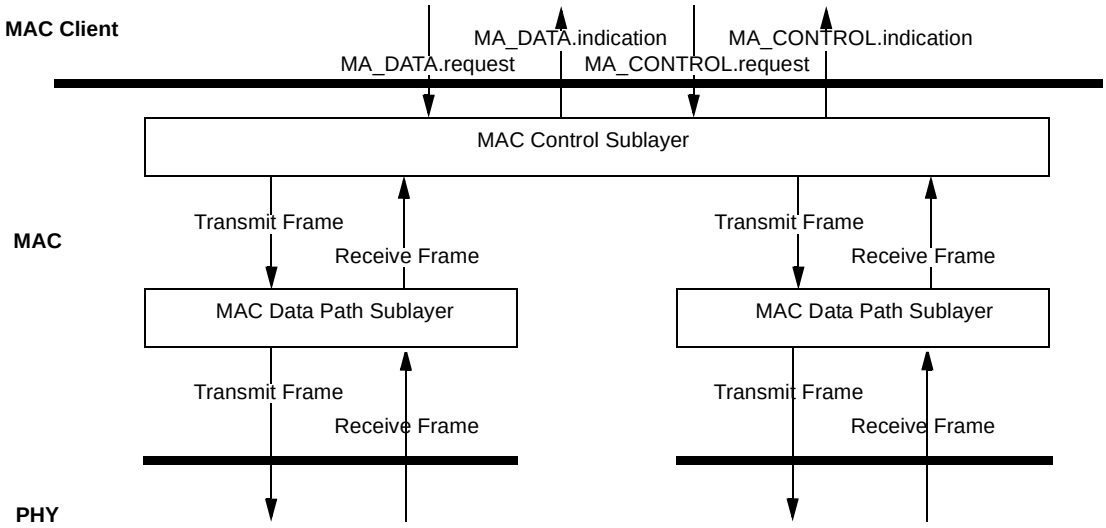


Figure 5.2—MAC service model

The primitives MA_DATA.request, MA_DATA.indication, MA_CONTROL.request, and MA_CONTROL.indication described in this subclause are mandatory.

5.3.1 MA_DATA.request

5.3.1.1 Function

This 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.

5.3.1.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
MA_DATA.request (
           destinationAddress,
           sourceAddress, [optional]
           mSDU,
           serviceClass,
           ringletID, [optional]
           MACProtection, [optional]
           markFE [optional])
```

The parameters of the MA_DATA.request are described below:

destinationAddress

This may specify either an individual or group MAC address, different from the local MAC address, to be used to create the *destinationMacAddress* field of the transmitted frame, as described in 8.2.3.

sourceAddress

If present, this must specify an individual MAC address to be used to create the *sourceMacAddress* field of the transmitted frame, as described in 8.2.4. If the *sourceAddress* parameter is omitted, the local MAC sublayer entity will insert a value associated with that entity.

mSDU

This provides the payload to be delivered. Specifically, the *mSDU* parameter specifies the MAC service data unit to be transmitted by the MAC sublayer entity, as described in 8.2.9. There is sufficient information associated with *mSDU* for the MAC sublayer entity to determine the length of the data unit.

serviceClass

This indicates the class of service requested by the MAC client, as described in 6.4 and 8.2.2.4.

ringletID

This indicates the ringlet choice of the client, as described in 6.3 and 8.2.2.1. If the *ringletID* parameter is omitted, the MAC shall use its default algorithm to determine which ringlet to use.

MACProtection

This indicates a choice of whether the MAC shall provide protection for the frame, as described in 6.3. If the *MACProtection* parameter is omitted, the MAC shall provide protection for the frame.

markFE

This indicates a request to mark and treat a classB frame as fairness eligible regardless of how it would have been marked or treated otherwise. This is provided to allow a client to choose which of the packets presented to the MAC are considered fairness eligible, such as when handling multiple flows of traffic, each with their own client-based rate allocations. The treatment of fairness eligible frames is discussed in Clause 6, and the indication of fairness eligibility is discussed in 8.2.2.2. If the *markFE* parameter is omitted, the default frame handling described in Clause 6 is performed.

5.3.1.3 When generated

This primitive is invoked by the client entity whenever data is to be transferred to a peer entity or entities.

5.3.1.4 Effect of receipt

The receipt of this primitive shall cause the MAC entity to insert all MAC specific fields, and any fields that are unique to the particular medium access method, and pass the properly formed frame to the lower protocol layers for transfer to the peer MAC sublayer entity or entities.

5.3.1.5 Additional comments

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

5.3.2 MA_DATA.indication

5.3.2.1 Function

This primitive defines the transfer of data from the MAC sublayer entity to the MAC client entity.

5.3.2.2 Semantics of the service primitive

The semantics of the primitive are as follows:

```
MA_DATA.indication (
           destinationAddress,
           sourceAddress,
           mSDU,
           receptionStatus,
           ringletID,
           serviceClass,
           fairnessEligible)
```

The parameters of the MA_DATA.request are described below:

destinationAddress
: This may be either an individual or group address, as specified by the *destinationMacAddress* field of the incoming frame, as described in 8.2.3.

sourceAddress
: This is an individual address as specified by the *sourceMacAddress* field of the incoming frame, as described in 8.2.4.

mSDU
: This specifies the MAC service data unit as received by the local MAC entity, as described in 8.2.9. There is sufficient information associated with *mSDU* for the MAC client entity to determine the length of the data unit.

receptionStatus
: This is used to pass the status of the received frame to the MAC client entity; as described in Table 5.1. *receptionStatus* can take the value of 1 only if the MIB variable *rprIfDropBadFcsControl* has been configured to FALSE.

ringletID
: This indicates, to MAC clients which optionally use the information, which ringlet the *mSDU* was received from, as described in 8.2.2.1.

serviceClass
: This indicates the service class at which the packet was sent, as described in 8.2.2.4.

fairnessEligible
: This indicates the setting of the *fairnessEligible* bit in the frame header, as described in 8.2.2.2.

Table 5.1—receptionStatus values

receptionStatus values	Description
0	no errors
1	FCS error
all others	reserved

5.3.2.3 When generated

The MA_DATA.indication is passed from the MAC sublayer entity (through the MAC Control sub-layer) 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, broadcast or multicast). A client may elect to accept or discard frames that have FCS errors.

5.3.2.4 Effect of receipt

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

5.3.2.5 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.indicate primitive to be sent to the originating client.

5.3.3 MA_CONTROL.request

This primitive defines the transfer of control requests from the MAC client to the MAC Control sublayer. Its purpose is to allow the MAC client to control the local MAC. 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) may be indirectly generated as a result of this request.

5.3.3.1 Function

This primitive defines the transfer of control commands from a MAC client entity to the local MAC Control sublayer entity.

5.3.3.2 Semantics of the service primitive

The semantics of the primitive are as follows:

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

The opcode indicates the control operation requested by the MAC client entity. The operations are described in Table 5.2.

Table 5.2—Control request opcodes

Opcode name	Meaning	Operands	Specified in
OAM_ECHO_REQ	Request to transmit an echo request frame	echo request parameters	12.3.1
OAM_FLUSH_REQ	Request to transmit a flush frame	flush parameters	12.3.2
all others	TBD	—	—

5.3.3.3 When generated

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

5.3.3.4 Effect of receipt

The effect of receipt of this primitive by the MAC control sublayer is opcode-specific.(See Clause TBD.)

5.3.4 MA_CONTROL.indication

5.3.4.1 Function

This primitive defines the transfer of control status indications from the MAC control sublayer to the MAC client.

5.3.4.2 Semantics of the service primitive

The semantics of the primitive are as follows:

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

The elements of the indication_operand_list are opcode-specific, and specified in Table 5.3.

Table 5.3—Control indication opcodes

Opcode name	Meaning	Operands	Specified in
OAM_ECHO_IND	Receipt of an echo reply frame	echo payload and parameters	12.3.1
OAM_FLUSH_IND	Receipt of a flush frame	flush payload and parameters	12.3.2
TOPO_CHANGE	Topology change	topology and status database	10.2.6
PROT_CHANGE	Protection change	topology and status database	10.2.6
SENDA	sendA change	true/false, ringletID	6.6.1
SENDB	sendB change	true/false, ringletID	6.6.1
SENDC	sendC change	hopsToCongestion, ringletID	6.6.1
SC_FCM_IND	Receipt of SC-FCM	allowedRate, allowedRateCongested, hopsToCongestion, ringletID	9.6
MC_FCM_IND	Receipt of MC-FCM	fairnessMessageType, controlValue, sourceAddress, TTL, ringletID	9.6
all others	TBD	—	—

5.3.4.3 When generated

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

5.3.4.4 Effect of receipt

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

5.4 MAC reference model

The MAC reference model for steering systems and for wrapping system that are not wrapped is shown in Figure 5.3.

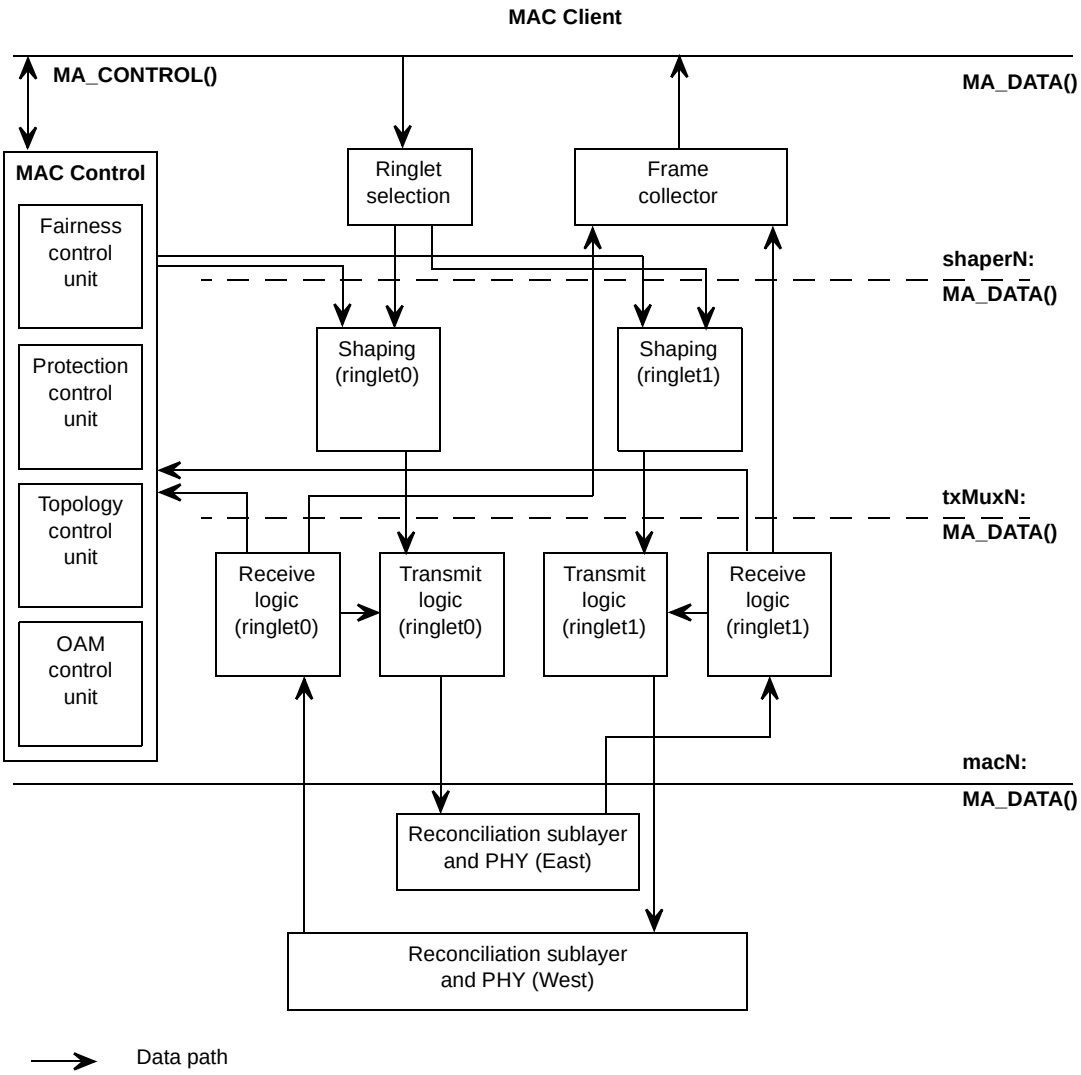


Figure 5.3—MAC reference model, showing internal MAC functions

The MAC reference model for wrapping systems that use center wrapping is shown in Figure 5.4.

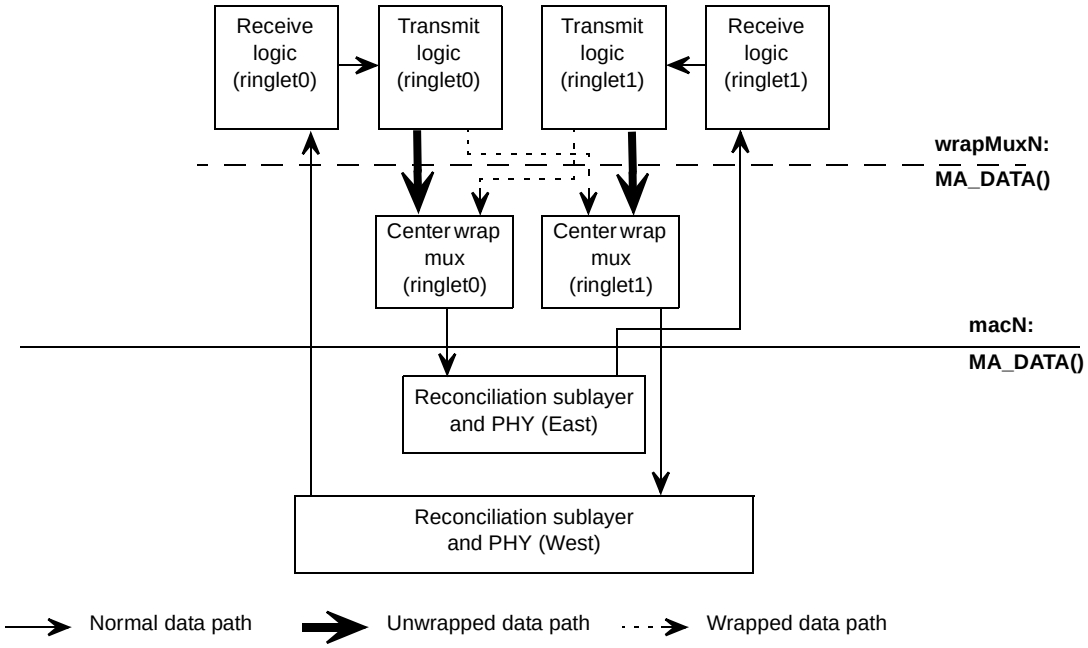


Figure 5.4—MAC reference model continued, showing center wrap data path

The MAC reference model for wrapping systems that use edge wrapping is shown in Figure 5.5

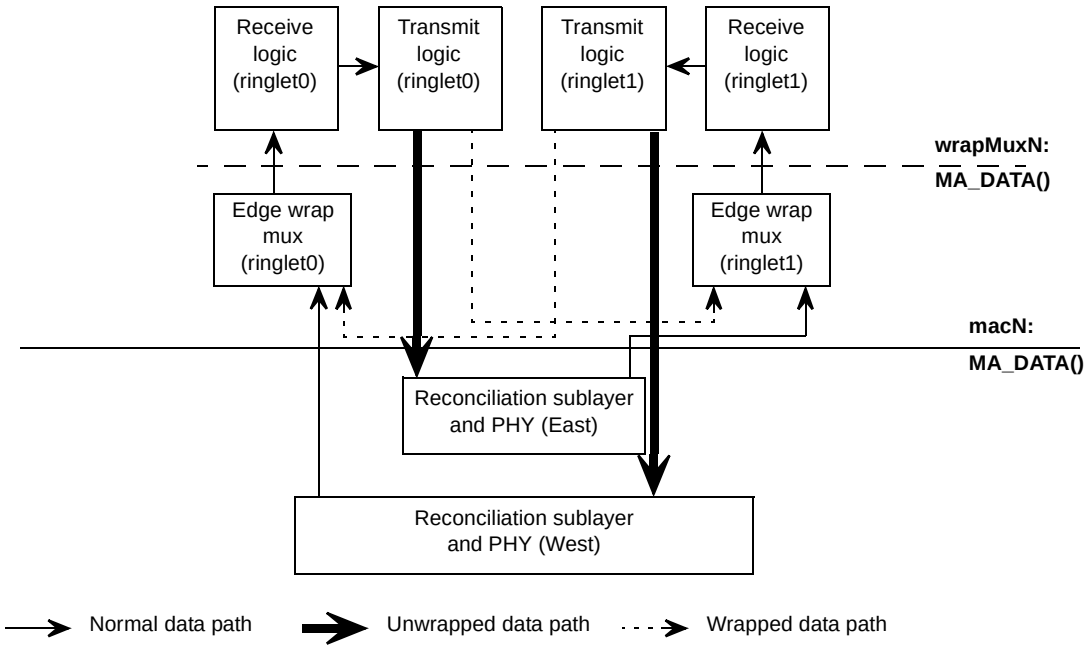


Figure 5.5—MAC reference model continued, showing edge wrap data path

5.4.1 PHY

The 1Gb/s Ethernet, 10 Gb/s Ethernet, and SONET/SDH physical layers are referenced, and the mappings for GFP and byte-synchronous HDLC are described in relationship to the referenced standard. It is under-

stood that different physical layers provide different services. The differences are reconciled by each PHY-specific reconciliation sublayer.

5.4.2 Reconciliation sublayer

The reconciliation sublayer 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. The reconciliation sublayer, described in Clause 7, provides a common interface for MAC/PHY data and also control of the PHY. Specific reconciliation sublayers are described in Annex B and Annex C.

5.4.3 RPR medium access control

The RPR medium access control layer 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, mapping of frames to the correct transit queue, rate control, fairness, protection, topology discovery, and ringlet selection.

5.4.3.1 Receive

The receive function 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. The receive logic is described in 6.8.

5.4.3.2 Transmit

The transmit function transmits frames to the reconciliation sublayer from the transit queue(s), the MAC control sublayer, and the MAC client. The transmit logic is described in 6.9.

5.4.3.3 Ringlet selection

Ringlet selection may 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, as described in 6.3.

5.4.3.4 Rate control

The rate control function governs the rate at which frames are transmitted from the client and from the transit queues, and 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.

ClassA, classB, and classC are shaped using token buckets. These shapers are used to indicate to the MAC client to cease making MA_DATA.requests for a particular service class. The functions of the shapers are described in detail in 6.6. The indications to the client are achieved through the sendA, sendB, and sendC signals described in Table 5.3 and 6.7.1.

Dynamic control of bandwidth to accommodate bursty traffic is accomplished through the fairness algorithm's bandwidth control mechanism described in Clause 9. This mechanism provides parameters to the token bucket algorithm (described in Clause 6) to ensure that the rate of frame insertion into the ring is both smoothed in time and consistent with avoiding ring overload and consequent packet loss. In addition, the optional multi-choke mechanism enables the client to provide packets to the MAC at an optimal rate even when more than one link is congested.

5.4.3.5 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 may not always be the same for all stations. The fairness algorithm ensures weighted dynamic distribution of available link bandwidths to source stations using those links.

5.4.3.6 Protection

By default, traffic is protected from failures in the ring and ring equipment. The MAC client may choose not to protect a given frame. The protection function provides the protection state machine and manages the protection database for the local MAC, and coordination of this control with the other MAC sublayers on the ring.

5.4.3.7 Topology

The topology function of the MAC sublayer manages the topology database. It also provides the topology state machine 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.

5.4.4 MAC layer management entity (MLME)

The MAC layer management entity 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.

5.4.5 Operations, administration, and maintenance (OAM)

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

6. Medium access control data path

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.3, June 2002	Adopted by TF and introduced into P802.17.
Draft 1.0, August 2002	Draft 1.0 for WG review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

6.1 Datapath overview

Editors' Notes: *To be removed prior to final publication.*

The following items remain to be decided:
— ringlet selection for wrapped stations,
— exact jitter bounds for classA and classB
— details for all the variables and functions used in state tables.

This clause describes per-ringlet behavior, unless explicitly mentioned otherwise. Ringlet selection, a unit of the MAC control sublayer, selects which ringlet's data path to use. The remainder of the clause operates with the chosen ringlet's data path. The selection of the data path, and the operations done on the selected data path are together treated as the MAC datapath, as shown in the shaded regions of Figure 6.1.

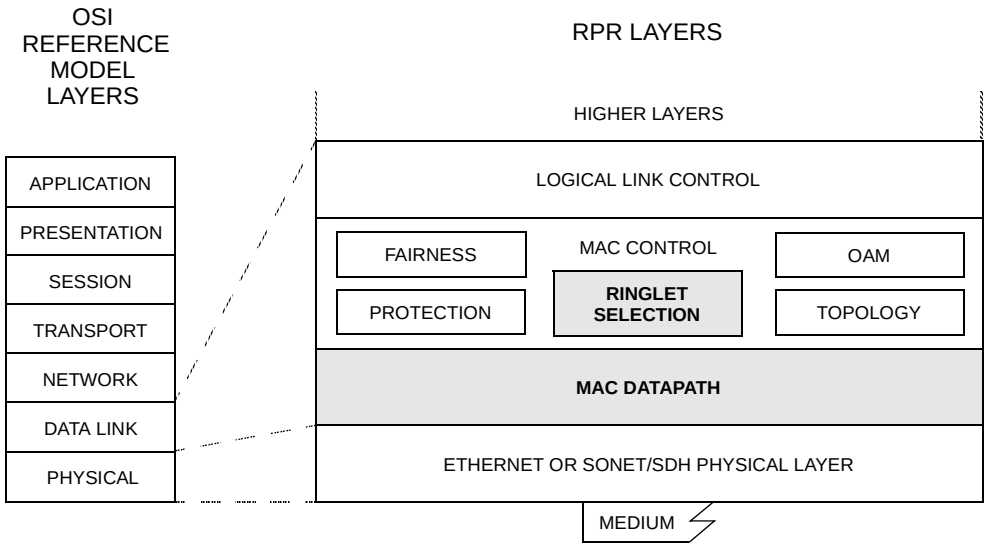


Figure 6.1—RPR layer diagram, MAC datapath portions

6.2 Variables and terminology used

6.2.1 Datapath terms and variables

This clause defines the following terms and variables:

> ***Editors' Notes:*** *To be removed prior to final publication.*
>
> *Most of these terms or variables are intended for the shaping subclauses. Although they were approved, the shaping subclauses were not. So they define things that are not yet described. However, these definitions are essential to the future contributions and comments that will be used to produce the shaping subclauses, and hence have been left in this section.*

6.2.1.1 *addRateCongestedOK*: Calculated. A boolean value indicating if the current *addRateCongested* is within acceptable limits.

6.2.1.2 *addRateOK*: Calculated. A boolean value indicating if the current *addRate* is within acceptable limits.

6.2.1.3 clientA0: Definition. The size of a client supplied classA frame transmitted as a subclassA0 frame.

6.2.1.4 clientA1: Definition. The size of a client supplied classA frame transmitted as a subclassA1 frame.

6.2.1.5 clientB: Definition. The size of a client supplied classB frame.

6.2.1.6 clientC: Definition. The size of a client supplied *FE* frame traveling short of the congestion point.

6.2.1.7 clientCc: Definition. The size of a client supplied *FE* frame traveling beyond the congestion point.

6.2.1.8 *creditA0*: Calculated. The total current credits accumulated by shaperA0.

6.2.1.9 *creditA1*: Calculated. The total current credits accumulated by shaperA1.

6.2.1.10 *creditB*: Calculated. The total current credits accumulated by shaperB.

6.2.1.11 *creditC*: Calculated. The total current credits accumulated by shaperC.

6.2.1.12 *creditCc*: Calculated. The total current credits accumulated by shaperCc.

6.2.1.13 *creditD*: Calculated. The total current credits accumulated by shaperD.

6.2.1.14 *creditI*: Calculated. The total current credits accumulated by shaperI.

6.2.1.15 *creditM*: Calculated. The total current credits accumulated by shaperM.

6.2.1.16 decSize: Definition. The amount by which the credits allocated to a shaper are decremented (usually the size of a transmitted frame).

6.2.1.17 *hiLimitA0*: Configured. The high threshold for shaperA0 credits.

6.2.1.18 *hiLimitA1*: Configured. The high threshold for shaperA1 credits.

6.2.1.19 *hiLimitB*: Configured. The high threshold for shaperB credits.

6.2.1.20 *hiLimitC*: Configured. The high threshold for shaperC credits.

6.2.1.21 *hiLimitCc*: Configured. The high threshold for shaperCc credits.

6.2.1.22 *hiLimitD*: Configured. The high threshold for shaperD credits.

6.2.1.23 *hiLimitI*: Configured. The high threshold for shaperI credits.

6.2.1.24 *hiLimitM*: Configured. The high threshold for shaperM credits.

6.2.1.25 *idleThreshold*: Constant. The threshold that indicates that a MAC is experiencing an adverse rate mismatch. When the PTQ depth exceeds this value, the incoming link is considered adversely rate mismatched.

6.2.1.26 incSize: Definition. The amount by which the credits allocated to a shaper are incremented (usually at a constant rate).

6.2.1.27 *loLimitA0*: Constant. The low threshold for shaperA0 credits.

6.2.1.28 *loLimitA1*: Constant. The low threshold for shaperA1 credits.

6.2.1.29 *loLimitB*: Constant. The low threshold for shaperB credits.

6.2.1.30 *loLimitC*: Constant. The low threshold for shaperC credits.

6.2.1.31 *loLimitCc*: Constant. The low threshold for shaperCc credits.

6.2.1.32 *loLimitD*: Constant. The low threshold for shaperD credits.

6.2.1.33 *loLimitI*: Constant. The low threshold for shaperI credits.

6.2.1.34 *loLimitM*: Constant. The low threshold for shaperM credits.

6.2.1.35 macA: Definition. The size of a MAC supplied classA frame.

6.2.1.36 macI: Definition. The size of a MAC supplied idle frame.

6.2.1.37 *passA0*: Calculated. The output value of shaperA0.

6.2.1.38 *passA1*: Calculated. The output value of shaperA1.

6.2.1.39 *passB*: Calculated. The output value of shaperB.

6.2.1.40 *passC*: Calculated. The output value of shaperC.

6.2.1.41 *passCc*: Calculated. The output value of shaperCc.

6.2.1.42 *passD*: Calculated. The output value of shaperD.

6.2.1.43 *passI*: Calculated. The output value of shaperI.

6.2.1.44 *passM*: Calculated. The output value of shaperM.

6.2.1.45 promiscuous: Definition. A MAC behavior that causes it to receive data frames regardless of the destinationMacAddress.

6.2.1.46 promiscuous mode: Definition. A setting in which a MAC receives data frames promiscuously.

6.2.1.47 *rateA0*: Configured. The allocated rate for subclassA0 client add traffic.

6.2.1.48 *rateA1*: Configured. The allocated rate for subclassA1 client add traffic.

6.2.1.49 *rateB*: Configured. The allocated rate for classB client add traffic.

6.2.1.50 *rateD*: Calculated. The allowed rate for unreserved client add traffic.

6.2.1.51 *rateI*: Configured. The calculated rate for MAC idle add traffic.

6.2.1.52 *rateM*: Configured. The allocated rate for classA MAC control add traffic.

6.2.1.53 *sendA*: Calculated. The indication generated to the client to indicate whether it may send classA traffic.

6.2.1.54 *sendB*: Calculated. The indication generated to the client to indicate whether it may send allocated classB traffic.

6.2.1.55 *sendC*: Calculated. The indication generated to the client to indicate whether it may send fairness eligible traffic.

6.2.1.56 shaperA0: Definition. The shaper for subclassA0 client add traffic.

6.2.1.57 shaperA1: Definition. The shaper for subclassA1 client add traffic.

6.2.1.58 shaperB: Definition. The shaper for classB client add traffic.

6.2.1.59 shaperC: Definition. The shaper for *FE* client add traffic traveling short of the congestion point.

6.2.1.60 shaperCc: Definition. The shaper for *FE* client add traffic traveling beyond the congestion point.

6.2.1.61 shaperD: Definition. The shaper for sustaining downstream subclassA0 traffic.

6.2.1.62 shaperI: Definition. The shaper for MAC idle add traffic.

6.2.1.63 shaperM: Definition. The shaper for classA MAC control add traffic.

6.2.2 Other calculated variables

This clause defines the calculation of the following variables which are defined in Clause 9:

a) *addRate*,
b) *addRateCongested*,
c) *fairnessEligible*,
d) *fullThreshold*,
e) *fwRate*,
f) *fwRateCongested*,
g) *highThreshold*,
h) LINK_RATE, and

i) *lowThreshold*.

6.2.3 Other variables used

This clause also uses the following variables defined in Clause 9:

a) *allowedRate*,
b) *allowedRateCongested*, and
c) *hopsToCongestion*.

6.3 Ringlet selection

This subclause describes how a ringlet is chosen. The remainder of this clause following this subclause (i.e. 6.4 and later subclauses) assume the ringlet has already been selected.

Ringlet selection is the first stage of a 3-stage process of transmitting a frame, initiated by receipt of a MA_DATA.request, as described in 5.3.1. After the ringlet has been chosen, the subsequent stages of transmitting the frame, stage queue processing (described in 6.9.1) and transmit selection (described in 6.9.8 or 6.9.11), act only within the datapath of the chosen ringlet. The functionality and state tables in 6.4 and later subclauses are hence duplicated for each ringlet.

6.3.1 Position of ringlet selection

Ringlet selection is a function of the MAC control sublayer, as shown in Figure 6.1. Ringlet selection is performed for client add traffic, and for OAM echo and flush frames. Ringlet selection occurs prior to shaping or any other action on the add traffic inside the MAC.

6.3.2 Ringlet selection actions

The primary action for ringlet selection is to choose the appropriate ringlet for each client data frame. The choice is based on the *ringletID* parameter, the *MACProtection* parameter, and the topology and status database. The client may exercise complete control, no control, or partial control over the path(s) that a frame takes.

When protection is active, ringlet selection optionally re-steers add frames for steering rings or for local attachment failure in wrapping rings using a center wrap scheme. Additionally, for steering rings, when ringlet selection chooses to send a multicast frame on 2 ringlets, ringlet selection performs the frame replication and sends the frame to both chosen 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.

Ringlet selection also determines whether to set the *wrapEligible* (*WE*) bit, based on the optional *MACProtection* parameter.

NOTE—This allows the co-existence of steered and wrapped frames from the same client, on wrapping rings.

Additionally, once the ringlet choice has been made, the TTL is set based on the destination's location on the chosen ringlet(s), as determined from the topology and status database.

NOTE—Whether the TTL is set to the exact hop count to the destination, *numStations*, MAX_STATIONS, or any other value equal to or greater than the hop count to the destination is a local implementation choice.

6.3.3 Relationship to other clauses

Ringlet selection makes use of the topology and status database, which is described in 10.2.6. 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 method. The protection method of the ring is used to determine whether to use steering or wrapping as the protection method.

6.3.4 Client control of ringlet selection

Ringlet selection is specified via the optional *ringletID* parameter and the optional *MACProtection* parameter of the MA_DATA.request primitive for each packet supplied by the client.

The values for ringletID are:

RINGLET_0
RINGLET_1
DEFAULT_RINGLET

If unspecified, the value for *ringletID* shall default to DEFAULT_RINGLET.

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 DEFAULT_RINGLET, the client can not know in advance which ringlet will be chosen, and therefore should follow the send indications from both ringlets. This can be represented as:

$$clientSendX = \text{minimum}(ringlet0sendX, ringlet1sendX) \quad (6.1)$$

See 6.6.1 and 6.? for a description of the send indications.

NOTE—A client using DEFAULT_RINGLET, either by choice or by default, may experience head of line blocking while waiting for the opposite ringlet from which its frame will be sent to become free.

The values for *MACProtection* are:

PROTECTED
UNPROTECTED

If unspecified, the value for *MACProtection* shall default to PROTECTED.

6.3.4.1 RINGLET_0, PROTECTED

A frame with a *ringletID* value of RINGLET_0 and a *MACProtection* value of PROTECTED will be 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 will be sent on ringlet1.

For multicast frames being sent on wrapping rings, the ringlet decision is the same as for unicast frames. For steering rings, multicast frames may be sent either or both directions, as necessary to reach all of the destination addresses.

For wrapping rings, the *wrapEligible* bit will be set. For steering rings, the *wrapEligible* bit has no meaning and will not be set.

6.3.4.2 RINGLET_0, UNPROTECTED

A frame with a *ringletID* value of RINGLET_0 and a *MACProtection* value of UNPROTECTED will be sent on ringlet0, regardless of state of ringlet0 ringlet. There is no guarantee that any UNPROTECTED frame will reach any unicast or multicast destination beyond any link with a non-IDLE protection state.

For unicast addresses, the frame may make it to the destination, depending upon whether any links with a non-IDLE protection state exist before the destination.

For multicast addresses, the frame may 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 *wrapEligible* bit will not be set.

6.3.4.3 RINGLET_1, PROTECTED

A unicast frame with a *ringletID* value of RINGLET_1 and a *MACProtection* value of PROTECTED will be 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 will be sent on ringlet0.

For multicast frames being sent on wrapping rings, the ringlet decision is the same as for unicast frames. For steering rings, multicast frames may be sent either or both directions, as necessary to reach all of the destination addresses.

For wrapping rings, the *wrapEligible* bit will be set. For steering rings, the *wrapEligible* bit has no meaning and will not be set.

6.3.4.4 RINGLET_1, UNPROTECTED

A frame with a *ringletID* value of RINGLET_1 and a *MACProtection* value of UNPROTECTED will be sent on ringlet1, regardless of state of ringlet1 ringlet. There is no guarantee that any unprotected frame will reach any unicast or multicast destination beyond any link with a non-IDLE protection state.

For unicast addresses, the frame may make it to the destination, depending upon whether any links with a non-IDLE protection state exist before the destination.

For multicast addresses, the frame may 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 *wrapEligible* bit will not be set.

6.3.4.5 DEFAULT_RINGLET, PROTECTED

A frame with a *ringletID* value of DEFAULT_RINGLET and a *MACProtection* value of PROTECTED will be sent on either or both ringlets (for some steering based protection scenarios), 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 addressed with the same destination address, until protection or topology changes.

For wrapping rings, if the frame is not replicated, the *wrapEligible* (*WE*) bit is set. For wrapping rings, if the frame is replicated, the *wrapEligible* (*WE*) bit is not set. For steering rings, the *wrapEligible* bit has no meaning and will not be set.

6.3.4.6 DEFAULT_RINGLET, UNPROTECTED

A frame with a *ringletID* value of DEFAULT_RINGLET and a *MACProtection* value of UNPROTECTED will be sent on either or both ringlets, as chosen by the MAC, with no allowance for unicast or multicast addresses beyond any links with a non-IDLE protection state.

For unicast addresses, the frame may make it to the destination, depending upon whether any links with a non-IDLE protection state exist before the destination.

For multicast addresses, the frame may 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 addressed with the same destination address, until topology changes.

For all rings, the *wrapEligible* bit is not set.

6.3.4.7 Setting of TTL for unicast frames

Stations shall set the TTL for unicast data frames to no less than the hop count to the desired station and to no more than MAX_STATIONS.

NOTE—Smaller values are acceptable if they are known to reach the desired destination. For example, one likely value would be *numStations*; and another would be the hop count to the desired destination in the chosen direction.

NOTE—Setting of TTL for OAM echo and flush frames is handled the same as for data frames. The setting of TTL for frames other than data frames is described in the clauses describing those frames.

6.3.4.8 Setting of TTL for multicast frames

Multicast data frames that are replicated for sending bidirectionally on both ringlets shall have their TTLs for each direction set to no more than the hop count in each direction to the point of protection.

Multicast data frames that are not replicated shall have their TTL set to no less than the hop count to the desired station and to no more than MAX_STATIONS.

NOTE—Smaller values, such as *numStations* or the hop count to the desired destination in the chosen direction, are acceptable if they are known to reach the desired destinations.

NOTE—The setting of TTL for frames other than data frames is described in the clauses describing those frames.

6.3.5 Ringlet selection state machine

This subclause specifies the state machine for selecting the ringlet and protection values for client frames being added.

6.3.5.1 Inputs

destinationAddress
: The *destinationAddress* parameter value passed in the MA_DATA.request() primitive, as defined in 5.3.1.

frame
: The RPR data frame constructed from the MA_DATA.request() primitive, as defined in 5.3.1, and formatted as specified in 8.2.

MACProtection
: The *MACProtection* parameter value passed in the MA_DATA.request() primitive, as defined in 5.3.1.

markFE
: The *markFE* parameter value passed in the MA_DATA.request() primitive, as defined in 5.3.1.

ringletID
: The *ringletID* parameter value passed in the MA_DATA.request() primitive, as defined in 5.3.1.

topologyCapabilities
: The topology capabilities, as defined in 10.2.6.

6.3.5.2 Constants

PROTECTED
: A *MACProtection* value defined in 6.3.4.

RINGLET_0
: A *ringletID* value defined in 6.3.4 and 8.2.2.1.

RINGLET_1
: A *ringletID* value defined in 6.3.4 and 8.2.2.1.

UNPROTECTED
: A *MACProtection* value defined in 6.3.4.

6.3.5.3 Variables

frame0
: A copy of frame, to send on ringlet0.

frame1
: A copy of frame, to send on ringlet1.

6.3.5.4 Functions

unicast
: To indicate if a destination MAC address is a unicast address.
Values:
TRUE: The address is a unicast address.
FALSE: The address is not a unicast address.

reachable0

To indicate if a destination MAC address is reachable on ringlet0, as determined from the topology and status database described in 10.2.6.

Values:

TRUE: The address is reachable on ringlet0.

FALSE: The address is not reachable on ringlet0.

reachable1

To indicate if a destination MAC address is reachable on ringlet1, as determined from the topology and status database described in 10.2.6.

Values:

TRUE: The address is reachable on ringlet1.

FALSE: The address is not reachable on ringlet1.

6.3.5.5 Ringlet selection state table

The ringlet selection state machine specified in Table 6.1, where rows are evaluated in top-to-bottom order, implements the functions necessary for client frame admission and the associated frame header processing. 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 machine is described in 3.4.

Table 6.1—Ringlet selection states

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	ringletID == RINGLET_0	1	—	RINGLET0
	ringletID == RINGLET_1	2	—	RINGLET1
	—	3	—	<RINGLET0, RINGLET1>
RINGLET0	MACProtection == UNPROTECTED	4	frame.RI = RINGLET_0; frame.WE = STEERABLE;	TRANSMIT
	topologyCapabilities.WC == FALSE	5	—	STEER0
	—	6	—	WRAP0
RINGLET1	MACProtection == UNPROTECTED	7	frame.RI = RINGLET_1; frame.WE = STEERABLE;	TRANSMIT
	topologyCapabilities.WC == FALSE	8	—	STEER1
	—	9	—	WRAP1
STEER0	unicast(destinationAddress) && reachable0(destinationAddress)	10	frame.RI = RINGLET_0; frame.WE = STEERABLE;	TRANSMIT
	unicast(destinationAddress) && reachable1(destinationAddress)	11	frame.RI = RINGLET_1; frame.WE = STEERABLE;	TRANSMIT

Table 6.1—Ringlet selection states *(continued)*

Current state		Row	Next state	
state	**condition**		**action**	**state**
	unicast(destinationAddress)	12	—	WAIT
	reachable0(destinationAddress)	13	frame.RI = RINGLET_0; frame.WE = STEERABLE;	TRANSMIT
	reachable1(destinationAddress)	14	frame.RI = RINGLET_1; frame.WE = STEERABLE;	TRANSMIT

	—	15	replicate(frame);
			frame0.RI = RINGLET_0;
			frame0.WE = STEERABLE;
			frame1.RI = RINGLET_1;
			frame1.WE = STEERABLE;
				TRANSMIT

WRAP0	—	16	frame.RI = RINGLET_0; frame.WE = WRAPABLE;	TRANSMIT
STEER1	unicast(destinationAddress) && reachable1(destinationAddress)	17	frame.RI = RINGLET_1; frame.WE = STEERABLE;	TRANSMIT
	unicast(destinationAddress) && reachable0(destinationAddress)	18	frame.RI = RINGLET_0; frame.WE = STEERABLE;	TRANSMIT
	unicast(destinationAddress)	19	—	WAIT
	reachable1(destinationAddress)	20	frame.RI = RINGLET_1; frame.WE = STEERABLE;	TRANSMIT
	reachable0(destinationAddress)	21	frame.RI = RINGLET_0; frame.WE = STEERABLE;	TRANSMIT

	—	22	replicate(frame);
			frame0.RI = RINGLET_0;
			frame0.WE = STEERABLE;
			frame1.RI = RINGLET_1;
			frame1.WE = STEERABLE;
				TRANSMIT

WRAP1	—	23	frame.RI = RINGLET_1; frame.WE = WRAPABLE;	TRANSMIT
TRANSMIT	—	24	follow state machine for stage queue selection for chosen ringlet(s)	WAIT
WAIT	MA_DATA.request()	25	—	START
	—	26	—	WAIT

Row 6.1-1: Client has requested RINGLET_0.
Row 6.1-2: Client has requested RINGLET_1.
Row 6.1-3: Client has either requested DEFAULT_RINGLET or has made no ringletID request. The next state is dependent upon the implemented algorithm and is beyond the scope of this standard.

Row 6.1-4: Client has requested UNPROTECTED. Prepare the frame for transmit on ringlet0.
Row 6.1-5: Client has requested PROTECTED and this is a steering ring.
Row 6.1-6: Client has requested PROTECTED and this is a wrapping ring.

Row 6.1-7: Client has requested UNPROTECTED. Prepare the frame for transmit on ringlet1.
Row 6.1-8: Client has requested PROTECTED and this is a steering ring.
Row 6.1-9: Client has requested PROTECTED and this is a wrapping ring.

Row 6.1-10: The destination MAC address is a unicast address and the topology and status database indicates that it is reachable on ringlet0. Prepare the frame for transmit on ringlet0.
Row 6.1-11: The destination MAC address is a unicast address and the topology and status database indicates that it is reachable on ringlet1. Prepare the frame for transmit on ringlet1.
Row 6.1-12: The destination MAC address is a unicast address and the topology and status database indicates that it is not reachable on ringlet0 or ringlet1. Ignore the frame and continue.
Row 6.1-13: The destination MAC address is a multicast address and the topology and status database indicates that all destinations are reachable on ringlet0. Prepare the frame for transmit on ringlet0. This is an optional condition that is not required to be supported.
Row 6.1-14: The destination MAC address is a multicast address and the topology and status database indicates that all destinations are reachable on ringlet1. Prepare the frame for transmit on ringlet1. This is an optional condition that is not required to be supported.
Row 6.1-15: The destination MAC address is a multicast address and the topology and status database indicates that all destinations are not reachable on ringlet0 or ringlet1. Replicate the frame and prepare one copy of the frame for transmit on ringlet0 and the other copy of the frame for transmit on ringlet1.

Row 6.1-16: Prepare the frame for transmit on ringlet0.

Row 6.1-17: The destination MAC address is a unicast address and the topology and status database indicates that it is reachable on ringlet1. Prepare the frame for transmit on ringlet1.
Row 6.1-18: The destination MAC address is a unicast address and the topology and status database indicates that it is reachable on ringlet0. Prepare the frame for transmit on ringlet0.
Row 6.1-19: The destination MAC address is a unicast address and the topology and status database indicates that it is not reachable on ringlet1 or ringlet0. Ignore the frame and continue.
Row 6.1-20: The destination MAC address is a multicast address and the topology and status database indicates that all destinations are reachable on ringlet1. Prepare the frame for transmit on ringlet1. This is an optional condition that is not required to be supported.
Row 6.1-21: The destination MAC address is a multicast address and the topology and status database indicates that all destinations are reachable on ringlet0. Prepare the frame for transmit on ringlet0. This is an optional condition that is not required to be supported.
Row 6.1-22: The destination MAC address is a multicast address and the topology and status database indicates that all destinations are not reachable on ringlet1 or ringlet0. Replicate the frame and prepare one copy of the frame for transmit on ringlet0 and the other copy of the frame for transmit on ringlet1.

Row 6.1-23: Prepare the frame for transmit on ringlet1.

Row 6.1-24: Transmit the frame(s) on the ringlet(s) selected in the RI field by passing control to the stage queue state machine for the chosen ringlet (see Table 6.18).

Row 6.1-25: Client has requested the transmission of a frame.
Row 6.1-26: There is no current frame transmission request from the client.

6.4 Service classes

Client data is classified into 3 service classes, as listed in Table 6.2. ClassA traffic is allocated with a committed information rate (CIR), and provides the lowest MAC end-to-end delay and jitter[1] 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. EIR classB and classC traffic is marked as *fairnessEligible* (*FE*) 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.

Editors' Notes: *To be removed prior to final publication.*

*The exact jitter bounds have not been calculated. ClassA jitter is on the order of N * MTU, where N is the number of stations on the ring. ClassB CIR jitter is on the order of RTT.*

Comments are requested with more precise bounds definitions/calculations.

Table 6.2—Service classes

Name	Bandwidth	Delay and jitter	Example use
classA	allocated	low	real time
classB	allocated	bounded	near real time
	opportunistic	unbounded	
classC	opportunistic		best effort

The configuration of the allocations of classA and classB traffic is described in Clause 13.

Control traffic shares the same service classes as used by client data. Control frames are sent as classA, unless otherwise specified in the definition of the control frame.

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.

Internal to the MAC, classA traffic is partitioned into two subclasses: subclassA0 and subclassA1. This partitioning is done in order 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, based on ring circumference and the size of the secondary transit queue (STQ) in that station, which determine how much classB and classC traffic can be queued while a congested station is signaling

[1]MAC 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.

upstream stations to decrease excess traffic. Single-queue implementations always allocate 100% of classA traffic to subclassA0, and 0% to subclassA1. The MAC advertises, via the topology message, a classA allocation equal to the internal subclassA0 amount. Bandwidth allocated to subclassA1 can be easily reclaimed by traffic of classB and classC when not being used by the nominal classA owner.

Given the amount of subclassA1 traffic that is desired to be added by a station, the sizing of the secondary transit queue (STQ) can be estimated by the formula:

sizeSTQ >= 2.3 x *RTT* x *rateA1* (6.1)

Alternatively, given a known STQ size, the amount of subclassA1 traffic that can be supported can be estimated by the formula:

rateA1 <= *sizeSTQ* x 0.43 / *RTT* (6.2)

Both of these formulas are approximations and assume the worst cases.

The value of *RTT* used in the above formulas includes the per hop delay at each station as each station waits to send a fairness message such that:

*RTT = numStations * advertisementInterval* + 5us * *ringkm* (6.3)

where *ringkm* is the length (or circumference) of the ring in kilometers.

NOTE—Equation 6.1 (and therefore also equation 6.2) is derived as follows, where:
sizeSTQ is the size in bytes of the STQ,
lowThreshold is the number of bytes before the *lowThreshold* of the STQ is hit,
highThreshold is the number of bytes before the *highThreshold* of the STQ is hit,
rateA1 is the allocated rate in bytes/microsecond of the subclassA1 traffic,
LR is the line rate in bytes/microsecond, and
RTT is the round trip time in microseconds for propagation of a fairness value around a ring.

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. Based on this, equation 6.1 is derived as shown below.

The first term, *lowThreshold*, is the amount the STQ will be allowed to fill before reporting congestion.

The second term, 2**RTT**(*rateA1*+*rateB*), is the 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 FE traffic from said station. The time to the farthest station is somewhat less than *RTT*. The time back from the farthest station is also somewhat less than *RTT*. Rounding up to a full *RTT* for each gives 2**RTT*. Multiplying by the rate of the local subclassA1 and classB traffic gives 2**RTT**(*rateA1*+*rateB*).

The third term, (*highThreshold*-*lowThreshold*)*((*LR*-(*rateA1*+*rateB*))/LR), is the amount of classC traffic that the local station is expected to add during the time between congestion detection (*lowThreshold*) and prevention of adding of FE traffic (*highThreshold*). The amount of bytes that could be added in this time, *highThreshold*-*lowThreshold*, is multiplied by the ratio of traffic that is expected to be added, (*LR*-(*rateA1*+*rateB*))/*LR*. Assuming the worst case, all classA traffic is subclassA1, then all remaining bandwidth, *LR*-(*rateA1*+*rateB*), could be filled by classC traffic.

Under the worst case assumptions that all add traffic is subclassA1 and that *rateA1* equals *LR*, and with the simplistic assumption that *lowThreshold* will be set to *sizeSTQ*/8 and *highThreshold* will be set to *sizeSTQ*/4, this first stage formula simplifies as shown in the subsequent steps.

a) *sizeSTQ* = *lowThreshold* + 2**RTT**(*rateA1*+*rateB*) + (*highThreshold*-*lowThreshold*)*((*LR*-(*rateA1*+*rateB*))/*LR*)

b) Since *lowThreshold* = *sizeSTQ*/8 and *highThreshold* = *sizeSTQ*/4, we can subtract *lowThreshold* from each side of the equation and simplify the right side of the equation, giving:
sizeSTQ * 7/8 = 2**RTT***rateA1* - ((*sizeSTQ*/8)*((*LR*-*rateA1*)/*LR*))/2

c) which can be solved to:
sizeSTQ = 8/7 * 2**RTT***rateA1* - ((*sizeSTQ*/8)*((*LR*-*rateA1*)/*LR*))/2

d) Assuming *rateA1* = *LR* (the worst case), the right side of the equation can be simplified to:
sizeSTQ = 8/7 * 2**RTT***rateA1* - ((*sizeSTQ*/8)*((0)/*LR*))/2

e) *sizeSTQ* = 8/7 * 2**RTT***rateA1*

f) *sizeSTQ* = 2.3**RTT***rateA1*

The above formulas can be modified to take into account variants such as fixed values for *lowThreshold* and *highThreshold*, different assumptions about ratio of subclassA1 traffic (instead of 100%), statistical approximations of the likelihood of achieving all the worst case scenarios, additional traffic in the transit queues of the in-between stations, etc.

Internal to the MAC, classB traffic is partitioned into two modes: classB-CIR and classB-EIR. This partitioning is done in order to differentially handle the classB traffic that is added by the client within the allocated classB rate and that which is in excess to the allocated rate. The MAC client requests classB traffic, not one of the internal modes. Rather than directly choose one of the modes, the client may either let the MAC choose based on the presence or absence of sendB, or it may force a classB frame to be considered only for classB-EIR (see 6.7.1 and 6.9.1). 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.

The fairness eligible traffic is opportunistic rather than allocated, in that it opportunistically uses bandwidth available from the unallocated remainder of the total link bandwidth or unused allocated bandwidth that is reclaimable, as described in 6.4.1. As such, the fairness eligible traffic provides no minimum-bandwidth or maximum-jitter guarantees. A weighted fairness algorithm is used to partition fairness eligible traffic among contending stations.

6.4.1 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 allocated subclassA1 and classB traffic can be done by any station that determines through the fairness algorithm that it is permitted to add fairness eligible traffic (see 6.6.1). The fairness algorithm and the datapath shapers guarantee that fairness eligible traffic may be added in amounts that will cause no more than minor priority inversions.

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

6.5 Data paths

A MAC transits frames that do not originate or terminate at that MAC. Each MAC data path (one per ringlet) has one or two transit queues to support classes of service precedence as shown in Figure 6.2. There are two types of MAC transit queueing designs: single-queue and dual-queue (see 6.9.6 and 6.9.9). 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 (PTQ), 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.

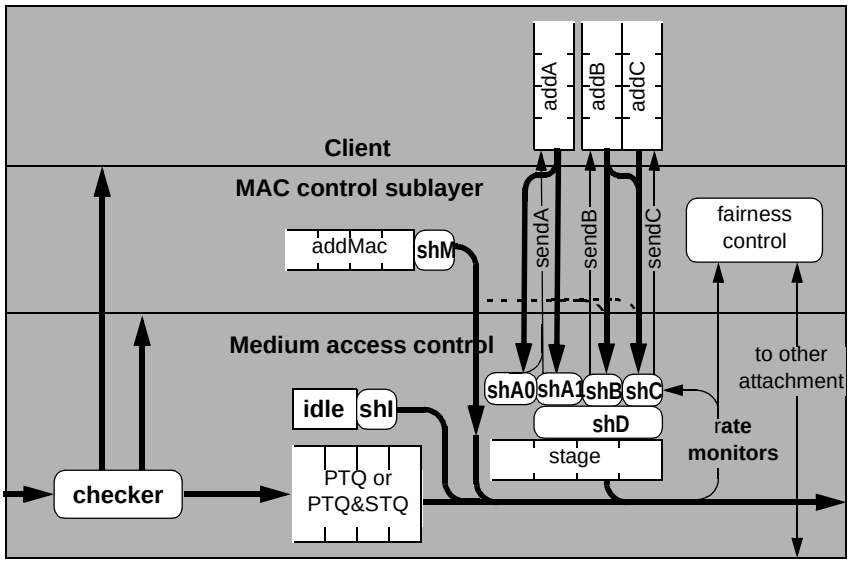


Figure 6.2—MAC data paths

The client labels its frames as classA, classB, or classC. The classA client add traffic flows through shaperA0 or shaperA1, as determined by the MAC. The classB client add traffic flows through shaperB or shaperC, as determined by the MAC. The classC client add traffic flows through shaperC. The shapers are described in 6.6. Frames added by the MAC control sublayer are usually labeled classA and shaped by shaperM, but can be labeled classB or classC and directed to the shapers for those classes (e.g. see 12.?). Additionally, all unreserved add traffic flows through shaperD, as described in 6.6. Optionally, rate synchronization idle frames are added by the MAC and shaped by shaperI, as described in 6.X. The correspondence between the services provided by the MAC and the transit paths used is summarized in Table 6.3, for implementations utilizing single and dual transit queues.

Table 6.3—Service mapping

Service class	Implementation type	Transit queue
A	single-queue dual-queue	Primary Primary
B	single-queue dual-queue	Primary Secondary
C	single-queue dual-queue	Primary Secondary

Accepted client traffic is placed into a stage queue as described in 6.9.1. The stage queue is a logical construct that may or may not correspond to any physical FIFO.

6.6 Rate control

All add traffic is rate controlled to maintain service class guarantees. (No direct rate control is applied to the transit traffic.) This also has the effect of limiting the strict precedence of transmit decisions such that each service class gets its fair share of transmissions.

6.6.1 Add queue flow control

The MAC does not maintain queues for shaping the client's add traffic, but provides per-ringlet, per-class flow control indications to the client. Transmission of client traffic is enabled with the send indications listed in Table 6.4.

Table 6.4—Send indication summary

description	send indication	inputs	subclause
classA send indication	sendA	passA0 passA1 passD	6.6.2
classB CIR send indication	sendB	passB passD	6.6.3
classB EIR and classC send indication	sendC	passC passD addRateOK add_rate_congestded_ok	6.6.4

Editors' Notes: *To be removed prior to final publication.*

There has been some interest in describing sendA and sendB as vector indications, like sendC, instead of binary indications, as currently defined. This has not been done as my understanding of the WG decision is that this is not desired to be explained in the standard, but should be possible to do outside of the standard. Please submit a comment against the text in the subclauses below if you disagree with this interpretation.

In addition to the above, all classes of traffic are effectively rate limited by the transmission selection algorithms described in 6.9.8 and 6.9.11.

6.6.2 sendA indication

This subclause describes the flow-control indication generated by the MAC to control client supplied classA traffic. The sendA indication assumes the availability of a passA0 or a passA1 indication from shaperA0 or shaperA1, respectively (see 6.7.5).

sendA has values of zero (FALSE) and nonzero (TRUE) during congested and uncongested conditions, respectively, as illustrated in Figure 6.3.

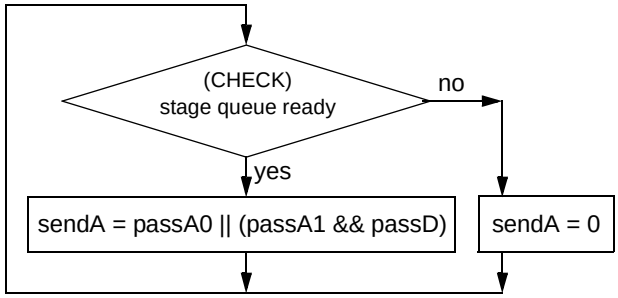


Figure 6.3—sendA indication

The sendA indication state machine specified in Table 6.5, where rows are evaluated in top-to-bottom order, implements the functions necessary for generating the sendA indication. 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 machine is described in 3.4.

Table 6.5—sendA indication

Current state		Row	Next state	
state	**condition**		**action**	**state**
CHECK	stageQueue can accept client frame	1	sendA = passA0 || (passA1 && passD);	CHECK
	—	2	sendA = 0;	

NOTE—passA0 is the output of shaperA0 (see Table 6.11). passA1 is the output of shaperA1 (see Table 6.11). *passD* is the output of shaperD (see Table 6.16).

Row 6.5-1: Setting sendA to the logical OR of the passA0 and passA1 (ANDed with passD) values enables throttled classA transmissions when the stage queue and the PHY are available. shaperA provides the passA permission indication (see 6.7.5) based on the measured client-supplied transmission rate of classA traffic.
Row 6.5-2: For store and forward stage queue implementations, at least one MTU of stage-queue storage is required to safely buffer client-supplied frames. For cut through stage queue implementations, the logical stage queue must be capable of immediately transmitting the client frame.

6.6.3 sendB indication

This subclause describes the flow-control indication generated by the MAC to control client supplied classB CIR traffic. The sendB indication assumes the availability of passB and passD indications from shaperB (see 6.7.6) and shaperD (see 6.7.8), respectively.

sendB has values of zero (FALSE) and nonzero (TRUE) during congested and uncongested conditions, respectively, as illustrated in Figure 6.4.

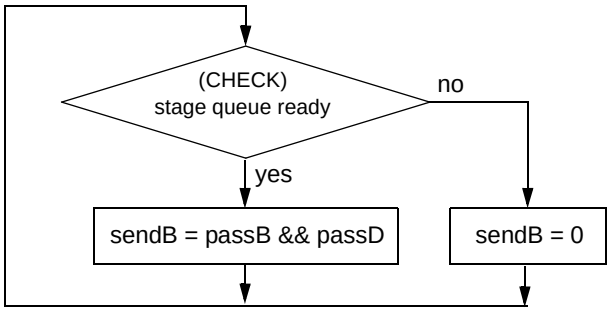


Figure 6.4—sendB indication

The sendB indication state machine specified in Table 6.6, where rows are evaluated in top-to-bottom order, implements the functions necessary for generating the sendB indication. 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 machine is described in 3.4.

Table 6.6—sendB indication

Current state		Row	Next state	
state	**condition**		**action**	**state**
CHECK	stageQueue can accept client frame	1	sendB = passB && passD;	CHECK
	—	2	sendB = 0;	

NOTE—*passB* is the output of shaperB (see Table 6.13). *passD* is the output of shaperD (see Table 6.16).

Row 6.6-1: Setting sendB to the logical AND of the passB and passD values enables throttled classB transmissions when the stage queue and the PHY are available. One shaper provides the passB permission indication (see 6.7.5) based on the measured client-supplied transmission rate of classB traffic. The other shaper provides the passD permission indication (see 6.7.8) based on the need to sustain downstream subclassA0 traffic.
Row 6.6-2: For store and forward stage queue implementations, at least one MTU of stage-queue storage is required to safely buffer client-supplied frames. For cut through stage queue implementations, the logical stage queue must be capable of immediately transmitting the client frame.

6.6.4 sendC indication

This subclause describes the flow-control indication generated by the MAC to control client supplied classB EIR and classC traffic. The sendC indication assumes the availability of passC, passCc, and passD indications from shaperC (see 6.7.7), shaperCc (see 6.7.7), and shaperD (see 6.7.8), respectively. These are combined with the addRateOK, addRateCongestedOK, and hopsToCongestion values calculated in Clause 9 to provide the value for sendC.

NOTE—The addition of addRateOK and addRateCongestedOK to the values of the shapers is to provide additional limitations beyond those dictated solely by rate control.

NOTE—The sendC value is different from the sendA and sendB values in that it is not a binary value, but provides a maximum hop count that fairness eligible frames may travel. This allows an implementation to selectively throttle fairness eligible traffic based on the hop count to its target.

Fairness eligible add traffic is rate controlled, via the sendC indication, to 2 rates. It is always controlled to a static rate configured for the station, allowedRate. In addition, any fairness eligible traffic that is passing the congestion point is limited to a dynamic rate, allowedRateCongested. The congestion point is the point reported in the single-choke fairness control messages. Fairness eligible traffic is rate controlled to the lesser of the station's static rate and the dynamic rate provided by the fairness algorithm. A sendC indication of 0 indicates that the static configured rate for the station has been exceeded; a sendC indication of greater than 0 and less than the ring size indicates the number of hops to the congestion point; and a sendC indication of (at least) the ring size indicates no congestion.

sendC has values of zero (FALSE) and nonzero (TRUE or hop-count) during congested and uncongested conditions, respectively, as illustrated in Figure 6.5.

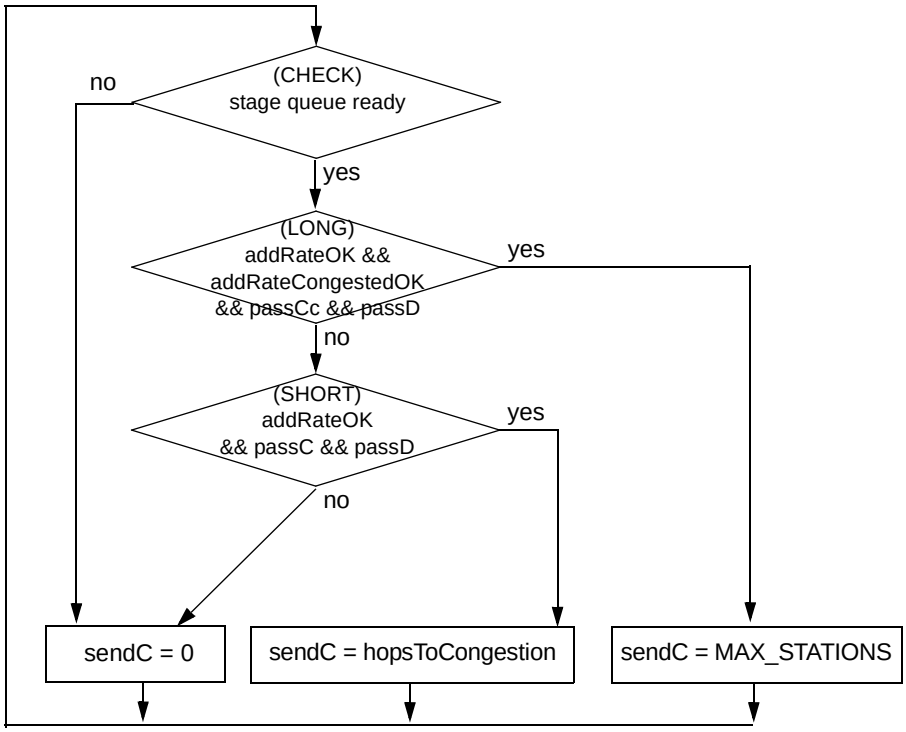


Figure 6.5—sendC indication

The sendC indication state machine specified in Table 6.7, where rows are evaluated in top-to-bottom order, implements the functions necessary for generating the sendC indication. 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 machine is described in 3.4.

Table 6.7—sendC indication

Current state		Row	Next state	
state	**condition**		**action**	**state**
CHECK	stageQueue can accept client frame	1	—	LONG
	—	2	sendC = 0	CHECK
LONG	addRateCongestedOK && addRateOK && passCc && passD	3	sendC = MAX_STATIONS	CHECK
	—	4	—	SHORT
SHORT	addRateOK && passC && passD	5	sendC = hopsToCongestion	CHECK
	—	6	sendC = 0	

NOTE—*passC* and *passCc* are the calculated outputs of shaperC (see Table 6.14) and shaperCc (see Table 6.15), respectively. *passD* is the output of shaperD (see Table 6.16). *addRateOK, addRateCongestedOK,* and *hopsToCongestion* are outputs of the fairness algorithm (see 9.5).

Row 6.7-1: Stage queue is ready, so check shapers.
Row 6.7-2: For store and forward stage queue implementations, at least one MTU of stage-queue storage is required to safely buffer client-supplied frames. For cut through stage queue implementations, the logical stage queue must be capable of immediately transmitting the client frame.

Row 6.7-3: Fairness eligible traffic may be sent past the point of congestion.
Row 6.7-4: Fairness eligible traffic may not be sent past the point of congestion.

Row 6.7-5: Fairness eligible traffic may be sent up to the point of congestion.
Row 6.7-6: Fairness eligible traffic may not be sent any distance.

6.7 MAC shapers

The shapers described in the following subclauses generate the passA0, passA1, passB, passC, and passD values used to generate the sendA (see 6.6.2), sendB (see 6.6.3), and sendC (see 6.6.4) indications that are supplied to the client, and the passM value used to indicate to the MAC control sublayer when a classA control frame may be transmitted. The shapers and indications operate on a per-ringlet basis.

Although multiple shapers are used within this standard, the behaviors of all shapers can be characterized by a common algorithm with application-specific parameters. All shapers' credits are adjusted down or up by *decSize* and *incSize*, respectively, as illustrated in Figure 6.6. The *decSize* and *incSize* values typically represent sizes of a transmitted frame and of credit increments in each update interval, respectively.

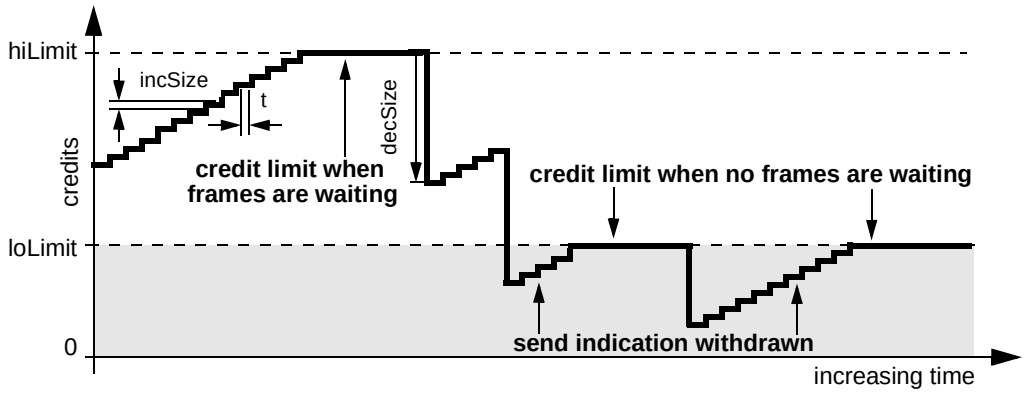


Figure 6.6—Credit adjustments over time

Crossing below the *loLimit* 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. When no frames are ready for transmission, credits can accumulate to no more than loLimit to bound the burst traffic after inactivity intervals. For all shapers described in this clause, *loLimit* is set to *sizeMTU* in order to allow the transmission of a full sized frame without reducing the credits below zero.

The *hiLimit* 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 *hiLimit*. The *hiLimit* value must be at least *sizeMTU*. If set to exactly *sizeMTU*, no bursts are allowed.

Each dynamic shaper consists of a token bucket. The token bucket has a default maximum depth of at least *sizeMTU* bytes. The credits in the token bucket are incremented by *incSize* at every time interval *t*. When a frame is waiting for access to the ring at the head of a queue, it will be granted ring access only if there are at least loLimit (*sizeMTU*) bytes of credits in the token bucket. The number of credits in the token bucket is then reduced by the size of the frame transmitted, *decSize*. It is never possible for the number of credits to become negative after subtracting the number of bytes in the transmitted frame. The maximum number of credits that can accumulate in the token bucket depends upon the shaper, and is usually an integral multiple of *sizeMTU* bytes.

6.7.1 Add queue rate shaping

Optional idle frames from the MAC rate synchronization add queue are shaped by shaperI (shaper of idles), described in 6.7.3.

Frames from the MAC control add queue are usually shaped by shaperM (shaper of MAC), a sub-shaper of shaperA, described below. shaperM is described in 6.7.4. Those control frames directed to the classB or classC add paths, are shaped by the shapers for those add paths, described below.

All classA add traffic is shaped by shaperA (shaper of classA), to avoid having the client exceed its classA allocated rates. shaperA is logically partitioned into shaperA0 (shaper of subclassA0), shaperA1 (shaper of subclassA1), and shaperM. MAC control and client traffic both flow through shaperA, although the shaper effectively only throttles the client's cumulative classA traffic since the control traffic has higher precedence. Which shaper is chosen for any particular classA frame is described in Table 6.18. shaperA0 and shaperA1 are described in 6.7.5.

All classB add traffic is shaped by shaperB (shaper of classB) and/or by shaperC (shaper of classC), to constrain the client within its classB CIR and EIR rates. Which shaper is chosen for any particular classB frame is described in Table 6.18. shaperB is described in 6.7.6.

Editors' Notes: *To be removed prior to final publication.*

The reference above to Table 6.5 is to Table 6.5 of Draft 1.1, not Table 6.5 in this proposed text.

All classC add traffic is shaped by shaperC (shaper of classC), to constrain the client within its weighted fair-share use of the residual (unallocated or unused allocated) bandwidth. shaperC is described in 6.7.7.

The client's subclassA1, classB, and classC traffic is additionally shaped by shaperD (shaper for downstream), to constrain the client to sustain the downstream allocated subclassA0 rate. (see Clause 9 for definition of unreserved_rate). Both the class specific shaper and shaperD must be passed in order to transmit a client frame. shaperD is described in 6.7.8.

6.7.2 Shaper summary

Each MAC rate shaper can be readily identified by its credit-value name, as listed in Table 6.8. Some of the transmission paths are affected by only one of these shapers; others are influenced by multiple shapers.

Editors' Notes: *To be removed prior to final publication.*

There has been some interest in adding another shaper to interleave frames from the transit and stage queues rather than service them in a strict priority manner. The need for this and means of implementing such have not achieved consensus.

Table 6.8—Rate-shaper summary

description	credits	shaper	subclause
idle transmissions	creditI	shaperI	6.7.3
MAC control allocated transmissions	creditM	shaperM	6.7.4
classA allocated transmissions	creditA0 creditA1	shaperA0 shaperA1	6.7.5
classB allocated transmissions	creditB	shaperB	6.7.6
fairness eligible transmissions	creditC creditCc	shaperC shaperCc	6.7.7
sustain downstream subclassA0 transmissions	creditD	shaperD	6.7.8

6.7.3 Idle shaper

Editors' Notes: *To be removed prior to final publication.*

The value of idleThreshold has not yet been specified, but is expected to be defined as a few multiples of the minimum (16-byte) frame size.

The idle shaper limits the MAC-supplied idle traffic to its provisioned limits, based on the parameters listed in Table 6.9. Idle frames are nominally inserted into the transmit datapath at a fixed 500PPM rate. 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 250PPM, which increases the transmit data rate, which eventually reduces the queue depth. The idle shaper is optional.

Table 6.9—Idle shaper parameters

creditI adjustment amounts		**creditI limit amounts**		**passI result**	
decSize	**incSize**	**hiLimitI value**	**loLimitI value**	creditI >= loLimitI	creditI < loLimitI
macI	rateI*time	idleSize	idleSize	TRUE	FALSE

As indicated by the decSize column, the *creditI* credits decrease by the size of the MAC-supplied idle frame transmissions. As indicated by the incSize column, the *creditI* credits increase by the current rate for idle frame transmissions (*rateI*) multiplied by the unit of time since the last increment. *rateI* is recalculated for every reception and transmission from and to the PTQ according to the formula:

rateI = (*freeSpaceInPTQ* < *idleThreshold* ? 250 : 500) / 1000000; (6.1)

6.7.4 Control shaper

The control shaper limits the MAC-supplied classA traffic to its allocated limits, based on the parameters listed in Table 6.10.

Table 6.10—shaperM shaper parameters and results

creditM adjustment amounts		**creditM limit amounts**		**passM result**	
decSize	**incSize**	**hiLimitM value**	**loLimitM value**	creditM >= loLimitM	creditM < loLimitM
macA	rateM*time	2*sizeMTU	sizeMTU	TRUE	FALSE

As indicated by the decSize column, the *creditM* credits decrease by the size of the MAC-supplied classA frame transmissions. As indicated by the incSize column, the *creditM* credits increase by the allowed rate for control frame transmissions multiplied by the unit of time since the last increment.

The default *hiLimitM* value is 2 * *sizeMTU*.

Editors' Notes: *To be removed prior to final publication.*

Is 2 MTU the correct default? Is 1 MTU better, or too restrictive? How bursty do we want to allow control traffic to be? Should the hiLimit be user configurable, or just an implementation choice?

6.7.5 classA shapers

The classA shapers limit the client-supplied classA transmissions to their allocated limits, using the parameters listed in Table 6.11 and Table 6.12.

Table 6.11—shaperA0 shaper parameters and results

creditA0 adjustment amounts		creditA0 limit amounts		passA0 result	
decSize	**incSize**	**hiLimitA0 value**	**loLimitA0 value**	creditA0 >= loLimitA0	creditA0 < loLimitA0
clientA0	rateA0*time	sizeMTU + rateA0*MAX_JITTER/2	sizeMTU	TRUE	FALSE

As indicated by the decSize column, the *creditA0* credits decrease by the size of the client-supplied subclassA0 frame transmissions. As indicated by the incSize column, the *creditA0* credits increase by the allocated rates for subclassA0 frame transmissions multiplied by the unit of time since the last increment.

The default value for *hiLimitA0* is set to preserve the delay and jitter guarantees of classA traffic. The default value is given by the following formulas:

$$hiLimitA0 = sizeMTU + rateA0 * \text{MAX\_JITTER} / 2 \quad (6.1)$$

$$\text{MAX\_JITTER} = (numStations * sizeMTU) / (1 - (rateA0 / \text{LINK\_RATE})) \quad (6.2)$$

NOTE—*sizeMTU* is chosen in the above formulas since this is the value of *loLimitA0*.

Editors' Notes: *To be removed prior to final publication.*

Should the hiLimit be user configurable, or just an implementation choice?

Table 6.12—shaperA1 shaper parameters and results

creditA1 adjustment amounts		creditA1 limit amounts		passA1 result	
decSize	**incSize**	**hiLimitA1 value**	**loLimitA1 value**	creditA1 >= loLimitA1	creditA1 < loLimitA1
clientA1	rateA1*time	sizeMTU + rateA1*MAX_JITTER/2	sizeMTU	TRUE	FALSE

As indicated by the decSize column, the *creditA1* credits decrease by the size of the client-supplied subclassA1 frame transmissions. As indicated by the incSize column, the *creditA1* credits increase by the allocated rates for subclassA1 frame transmissions multiplied by the unit of time since the last increment.

The default value for *hiLimitA1* is set to preserve the delay and jitter guarantees of classA traffic. The default value is given by the following formulas:

hiLimitA1 = *sizeMTU* + *rateA1* * MAX_JITTER / 2 (6.3)

MAX_JITTER = (*numStations* * *sizeMTU*) / (1 - (*rateA1* / LINK_RATE)) (6.4)

NOTE—*sizeMTU* is chosen in the above formulas since this is the value of *loLimitA1*.

Editors' Notes: *To be removed prior to final publication.*

Should the hiLimit be user configurable, or just an implementation choice?

6.7.6 classB shaper

The classB shaper limits the client-supplied classB transmissions to their allocated limits, using the parameters listed in Table 6.13.

Table 6.13—shaperB shaper parameters and results

creditB adjustment amounts		creditB limit amounts		passB result	
decSize	**incSize**	**hiLimitB value**	**loLimitB value**	creditB >= loLimitB	creditB < loLimitB
clientB	rateB*time	(2*N*sizeMTU+RTT)*R	sizeMTU	TRUE	FALSE

The *creditB* credits decrease by the sizes of client-supplied classB CIR frame transmissions. As indicated by the incSize column, the *creditB* credits increase by the allocated rate for classB CIR frame transmissions multiplied by the unit of time since the last increment.

The default *hiLimitB* value is (2 * N * *sizeMTU* + RTT) * R, where:

N = *numStations*
R = *rateB*/LINK_RATE
RTT = number of bytes that can be transmitted during the time it takes for one round trip time and *sizeMTU* is chosen since this is the value of *loLimitB*.

Editors' Notes: *To be removed prior to final publication.*

*Is (2 * N * MTU + RTT) * R the correct default? How bursty do we want to allow allocated classB traffic to be? Should the hiLimit be user configurable, or just an implementation choice? Should N be fixed at maxStations or dynamically reset to numStations at each topology change?*

6.7.7 Fairness eligible shapers

The fairness eligible shapers limit the client-supplied classB EIR and classC transmissions to their allowed limits. Different limits apply depending upon whether the traffic is passing the point of congestion or not, as determined by comparing the timeToLive assigned by ringlet selection to the value of hopsToCongestion.

Traffic traveling not as far as the congestion point (frame.timeToLive <= *hopsToCongestion*) uses the parameters listed in Table 6.14.

Table 6.14—shaperC shaper parameters and results

creditC adjustment amounts		creditC limit amounts		passC result	
decSize	**incSize**	**hiLimitC value**	**loLimitC value**	creditC >= loLimitC	creditC < loLimitC
clientC	allowedRate*time	2*sizeMTU	sizeMTU	TRUE	FALSE

The *creditC* credits decrease by the sizes of client-supplied fairness eligible frame transmissions travelling short of the congestion point. The *creditC* credits increase by the fairness algorithm supplied *allowedRate* multiplied by the amount of time since the last increase.

The default value for *hiLimitC* is 2 * *sizeMTU*.

Editors' Notes: *To be removed prior to final publication.*

*Is 2 * MTU the correct default? How bursty do we want to allow fairness eligible traffic to be? Should the hiLimit be user configurable, or just an implementation choice? Should N be fixed at maxStations or dynamically reset to numStations at each topology change?*

Traffic traveling beyond the congestion point (frame.timeToLive > *hopsToCongestion*) uses the parameters listed in Table 6.15.

Table 6.15—shaperCc shaper parameters and results

creditCc adjustment amounts		creditCc limit amounts		passCc result	
decSize	**incSize**	**hiLimitCc value**	**loLimitCc value**	creditCc >= loLimitCc	creditCc < loLimitCc
clientCc	allowedRateCongested*time	2*sizeMTU	sizeMTU	TRUE	FALSE

The *creditCc* credits decrease by the sizes of client-supplied fairness eligible frame transmissions travelling past the congestion point. The *creditCc* credits increase by the fairness algorithm supplied *allowedRateCongested* multiplied by the amount of time since the last increment.

The default value for *hiLimitCc* is 2 * *sizeMTU*.

Editors' Notes: *To be removed prior to final publication.*

*Is 2 * MTU the correct default? How bursty do we want to allow fairness eligible traffic to be? Should the hiLimit be user configurable, or just an implementation choice? Should N be fixed at maxStations or dynamically reset to numStations at each topology change?*

6.7.8 Downstream shaper

The downstream shaper monitors the transit traffic to ensure sufficient levels of sustainable subclassA0 traffic for downstream stations, as listed in Table 6.16.

Table 6.16—shaperD shaper parameters and results

creditD adjustment amounts		creditD limit amounts		passD result	
decSize	**incSize**	**hiLimitD value**	**loLimitD value**	creditD >= loLimitD	creditD < loLimitD
!clientA0	unreservedRate*time	sizeMTU	sizeMTU	TRUE	FALSE

As indicated by the decSize column, the *creditD* credits decrease by the sizes of client-supplied frames that are not subclassA0. As indicated by the incSize column, the *creditD* credits increase by the *unreservedRate* multiplied by the amount of time since the last increase.

Editors' Notes: *To be removed prior to final publication.*

The calculation for passD removes the need for the following line in 9.7.3.8:
&& (nrXmitRate < unreservedRate) // space for reserved traffic

The default *hiLimitD* value is *sizeMTU*.

Editors' Notes: *To be removed prior to final publication.*

Is 1 MTU the correct default? How bursty do we want to allow unreserved traffic to be? Note that the smaller of hiLimitD and the hiLimits for shaperB and shaperC ends being the limiting amount for classB and classC, respectively.

Should the hiLimit be user configurable, or just an implementation choice?

6.8 Receive operation

If a station is operating in a passthru mode, it is no longer considered a station on the ring, and it does not participate in the receive operation. Station passthru can be thought of as being equivalent to an optical regenerator where the MAC operates as a transit station. The means by which a MAC enters passthru, if at all, is a local implementation issue. In passthru mode, *timeToLive* is not decremented, and all frames are transited without modification.

6.8.1 Receive operation in strict mode

6.8.1.1 Context containment

In order to avoid any duplication or misordering of frames with no tolerance of such, strict data frames (data frames with *strictOrder* set to 1) use a context containment mechanism. This mechanism ensures that upon detection of a protection switch event (a protection event resulting in a change to the topology and status database), that in-flight strict data frames that were transmitted by a source using an outdated context get removed from the ring prior to the transmission of strict data frames using an updated context. In-flight strict data frames launched using an outdated context are effectively purged from the ring.

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. Refer to Clause 11 for a description of when protection control frames are dispatched and their impact on a station's topology and status database.

6.8.1.2 timeToLive validation

In addition to the *timeToLive* scoping rules used for all frames, strict data frames are also discarded if the number of hops they have traveled is not consistent with the receiving station's topology and status database. Strict data frames are discarded if they are on their primary ringlet and SRC[*numberOfHops*]!=frame.SA, where SRC is the localMac column of the topology and status database, and *numberOfHops* = frame.*ttlBase* - frame.*timeToLive*. This extra discard check ensures that no frame duplication occurs due to an intermediate station going into passthru. Both wrapping and steering systems perform this check, at all ring stations, on frames traveling on the primary ringlet. Figure 6.7 provides an illustration.

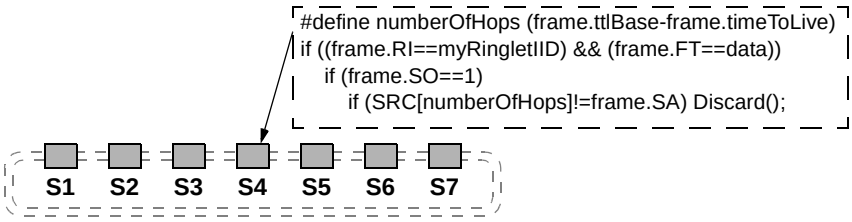


Figure 6.7—Source consistency check

6.8.1.3 Steering systems

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, the receiving steering station purges all strict data frames currently within its transit queue(s), and continues to purge all received strict data frames for a specified duration[2]. After the duration has expired, the station returns to normal operations, and transits all frames as requested.

Editors' Notes: *To be removed prior to final publication.*

The duration of the purge is under investigation.

For example, a 15ms time allocation allows an in-flight frame to travel from one station to an adjacent station and provides a margin to allow the station to remove arriving in-flight frames. A 2000km ring span results in frame delivery from one station to another in 10ms. The additional 5ms is allocated for marginal station processing to remove such frames from the ring and clear the transit buffer(s) of data frames.

Should this be described in a data path state machine or a protection state machine?

Implementation considerations (for single-queue systems):

A 15 ms timer can be implemented in SW or HW. For the duration of the timer, the checker entity purges all strict data frames prior to transfer to PTQ. All strict data frames leaving the PTQ are also purged.

Implementation considerations (for dual-queue systems):

This involves storing the current STQ write pointer and deleting any strict data frames during read operations while the read pointer has not passed the stored WP. Additionally, any strict data frames arriving are purged. A 15ms timer can be implemented in SW or HW.

Consider the illustration in Figure 6.8 below. 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

[2]For example, the duration could be specified by a discovery provided purgeNow=1 indication. Alternatively, a duration of 15ms could be generally specified.

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).

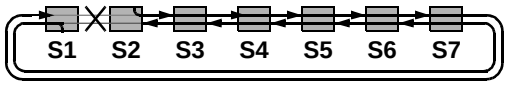


Figure 6.8—Protection event example

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.

The context containment mechanism ensures that there is no reorder of frames. Reorder can occur during a protection switch. Specifically, when a source station transmits frames using one context followed by a different context. Since the flooding scope (FS) of these frames could be different, it is possible for these frames to be delivered to a particular destination station out of order. Context containment removes frames dispatched using an old context before frames are dispatched using a new context. Frame reorder prevention is guaranteed.

6.8.1.4 Wrapping systems

It is possible that a series of failures can leave the topology significantly different from when the frame was originally launched. A *pastSource* bit in the *rprHeader* and some eligibility rules for wrapping frames can prevent frame duplication.

The *pastSource* bit is initially cleared when a frame is transmitted by a station. In order to wrap a frame from its primary ringlet to the secondary ringlet, the *wrapEligibile* bit (found in the *baseRingControl*) must be set, the *pastSource* bit must be cleared, and the *ringletId* of the frame and ringlet traveled must be the same. When the frame passes the source station on the secondary ringlet, the *pastSource* bit is set. In order to wrap a frame from the secondary ringlet to the primary ringlet, the *wrapEligible* bit must be set, the *pastSource* bit must be set, and the *ringletId* of the frame and ringlet traveled must be different.

The behavior of a wrapping system is illustrated in Figure 6.9.

The *timeToLive* is decremented when the frame enters the secondary ringlet (i.e., initially wrapped). While the frame travels on the secondary ringlet, the *timeToLive* is no longer decremented. When the frame is wrapped back to the primary ringlet, normal *timeToLive* decrement operation continues.

The act of unwrapping the ring (i.e., ring recovery) will trap frames on the wrong ringlet. These frames will be 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 will have been discarded and wrapping these frames onto their primary ringlet will cause reorder. Therefore, following the act of

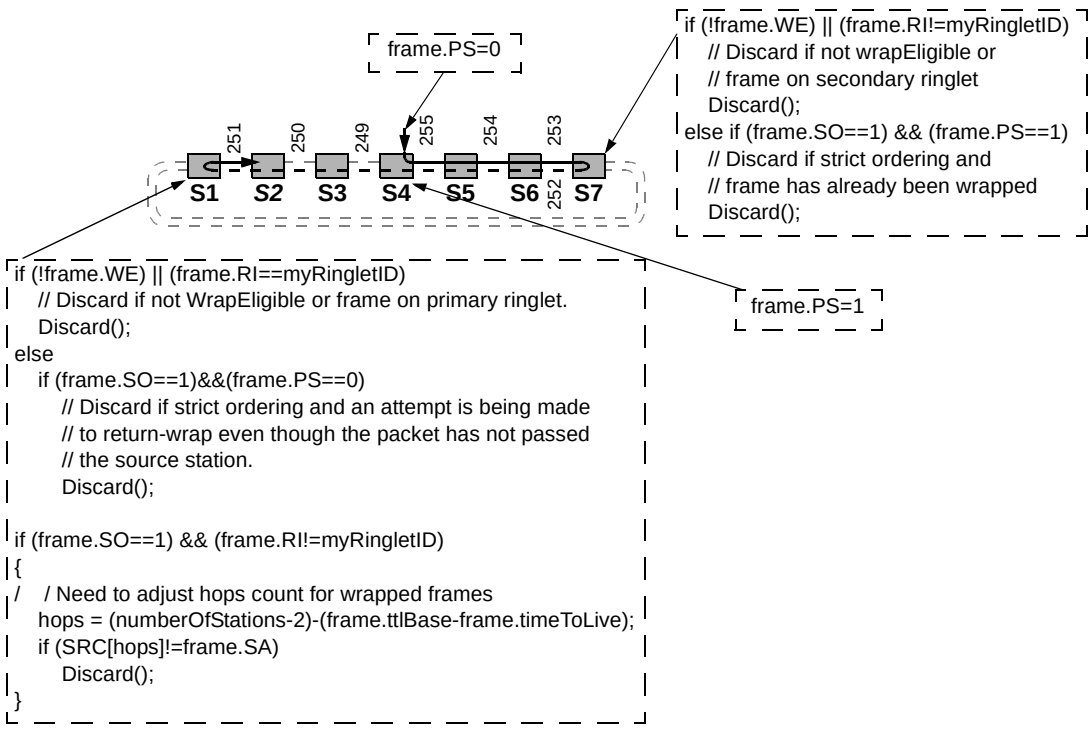


Figure 6.9—Strict mode wrapping system behavior

unwrapping a ring, all stations must delete all strict data frames in flight and all strict data frames in the transit queues whose *RI* does not match the ringlet traveled. This state must persist for a specified duration.

Editors' Notes: *To be removed prior to final publication.*

Implementation considerations for the deletion of strict data frames from transit queues on receipt of an unwrap message.

This involves storing the current STQ write pointer and deleting any strict data frames with the wrong RI during read operations while the read pointer has not passed the stored WP. Additionally, any strict data frames arriving with the wrong RI are purged. For example, a 15ms timer can be implemented in SW or HW.

The duration of the purge is under investigation.

Should this be described in a data path state machine or a protection state machine?

Note that if a new wrap occurs within this period, some additional frame loss will occur (of the newly wrapped frames) but no reorder will occur.

6.8.2 Receive operation state machine

This subclause specifies the state machine for receiving frames from the physical layer.

6.8.2.1 Receive operation flow chart

The receive operation is shown in Figure 6.10.

The single-queue and dual-queue operations are both defined, with the queue-design dependent portions shaded in grey.

In case of conflict, the state tables in this clause shall take precedence over flow charts or text. Flow charts are intended to give overviews, whereas the details are described in state tables.

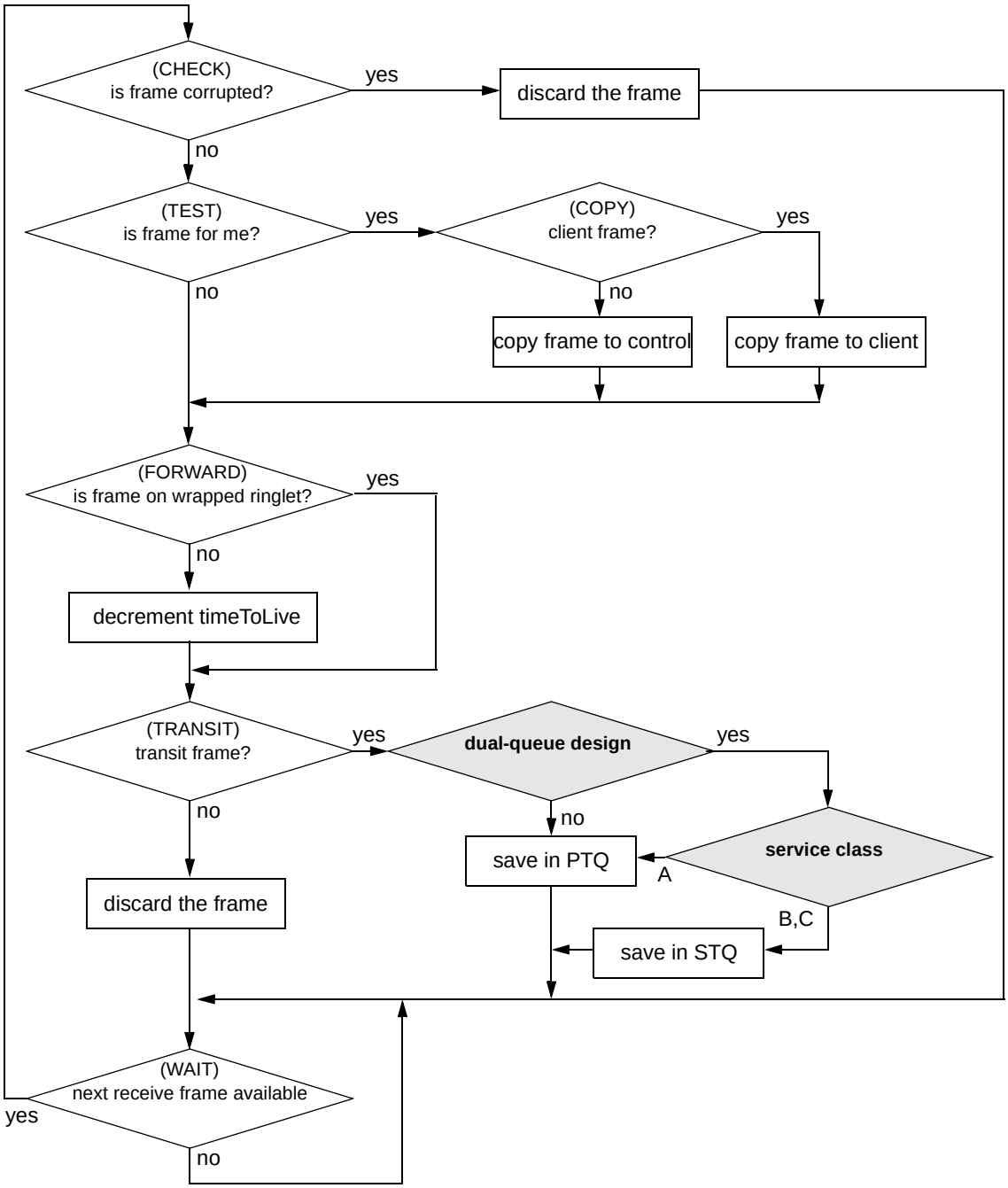


Figure 6.10—Frame reception

6.8.2.2 Inputs

frame
The INPUT_FRAME parameter received from the PHY_DATA.indicate() primitive, as described in 7.2.2.

myMacAddress
The MAC address of the local MAC.

myRingletID
The ringlet ID of the ringlet processing the received frame.

myState
The state of the MAC, as defined in ?.

6.8.2.3 Constants

CLASS_A0
A *serviceClass* value defined in 8.2.2.4.

CLASS_A1
A *serviceClass* value defined in 8.2.2.4.

CONTROL_FRAME
A *frameType* value defined in 8.2.2.3.

DATA_FRAME
A *frameType* value defined in 8.2.2.3.

FAIRNESS_FRAME
A *frameType* value defined in 8.2.2.3.

IDLE_FRAME
A *frameType* value defined in 8.2.2.3.

IDLE
A protection state value defined in Clause 11.

SC_FCM
A *fcmType* value defined in 9.7.4.1.

STEERABLE
A *wrapEligible* value defined in 8.2.2.5.

TOPOLOGY
A *controlType* value defined in 8.3.7.

WRAPABLE
A *wrapEligible* value defined in 8.2.2.5.

6.8.2.4 Variables

frame0
A copy of frame, to send on ringlet0.

frame1
A copy of frame, to send on ringlet1.

6.8.2.5 Functions

crc16
Returns the CRC-16 of the referenced bytes, as described in 8.2.7.

crc32
Returns the CRC-32 of the referenced bytes, as described in 8.2.10.
copyToClient
Copies the frame to the client.
copyToControl
Copies the frame to control sublayer.
discard
Discards the frame.
frameLength
Returns the length of a frame.
Value: The number of bytes in the frame.
modifyCrc16
Modifies the frame HEC to adjust for the change of the timeToLive value, as described in F.2.4.
moveToPTQ
Moves the frame to the end of the primary transit queue of the ringlet on which it was received.
moveToSTQ
Moves the frame to the end of the secondary transit queue of the ringlet on which it was received.
multicast
To indicate if a MAC address is a multicast address.
Values:
TRUE: The address is a multicast address.
FALSE: The address is not a multicast address.
myMulticast
To indicate if a MAC address is a multicast address being accepted by this MAC.
Values:
TRUE: The address is an accepted multicast address.
FALSE: The address is not an accepted multicast address.
parity
Returns the parity of the referenced bits, as described in 8.2.2.6.
protectionState
The current state of the protection state machine, as defined in Clause 11.
stomp
Returns the stomped value of the CRC-32 of the referenced bytes, as described in F.2.3.
topologyCapabilities
The topology capabilities, as defined in 10.2.6.

6.8.2.6 Receive operation state table

The frame reception state machine specified in Table 6.17, where rows are evaluated in top-to-bottom order, implements the functions necessary for client frame reception, copying of frames to the client and to the control sublayer, forwarding frames along the same ringlet and the associated frame header processing, and discarding frames that should not be forwarded. The state machine is initiated upon receipt of a PHY_DATA.indicate, as described in 7.2.2. The single-queue and dual-queue operations are both defined; the queue-design dependent portions are shaded in grey. 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 machine is described in 3.4.

Table 6.17—Frame reception states

Current state		Row	Next state	
state	**condition**		**action**	**state**
CHECK	(frame.FT == FAIRNESS_FRAME && frame.parity != parity(frame.header)) || frame.HEC != crc16(frame.header)	1	discard()	WAIT
	frame.timeToLive == 0	2		
	frame.RI == myRingletID && frame.FT == DATA_FRAME && frame.SO && SRC[numberOfHops] != frame.SA	3		
	(frame.FT == FAIRNESS_FRAME || frame.FT == IDLE_FRAME) && frameLength(frame) != 16	4		
	multicast(frame.sourceMacAddress)	5		
	frame.WE == 0 && frame.RI != myRingletID	6		
	frame.RI != myRingletID && protectionState == IDLE	7		
	frame.RI != myRingletID && topologyCapabilities.WC == FALSE	8		
	frame.FCS == crc32(frame.payload)	9	—	TEST
	frame.FCS == stomp(frame.payload)	10	—	
	—	11	frame.FCS = stomp(frame.payload)	
TEST	frame.FT == IDLE_FRAME	12	discard()	WAIT
	frame.FT == FAIRNESS_FRAME	13	—	COPY
	frame.RI != myRingletID && frame.FT == CONTROL_FRAME && frame.controlType == TOPOLOGY	14	copyToControl()	WAIT
	frame.RI != myRingletID && ! myState.wrapped	15	—	FORWARD
	frame.FT == DATA_FRAME && (myState.promiscuous == 1 || frame.FF == 1 || frame.FF == 2)	16	—	COPY
	frame.destinationMacAddress == myMacAddress	17	—	COPY
	myMulticast(frame.destinationMacAddress)	18	—	COPY
	frame.sourceMacAddress == myMacAddress	19	discard()	WAIT
	—	20	—	FORWARD

Table 6.17—Frame reception states *(continued)*

Current state		Row	Next state	
state	**condition**		**action**	**state**
COPY	frame.FT == CONTROL_FRAME || frame.FT == FAIRNESS_FRAME	21	copyToControl()	FORWARD
	frame.sourceMacAddress == myMacAddress	22	discard()	WAIT
	—	23	copyToClient()	FORWARD
FORWARD	frame.RI != myRingletID	24	—	TRANSIT
	—	25	timeToLive -= 1; modifyCrc16(frame.HEC);	
TRANSIT	frame.timeToLive == 0	26	discard()	WAIT
	frame.FT == FAIRNESS_FRAME && frame.FMT == SC_FCM	27		
	frame.WE == 0 && myState.wrapped	28		
	frame.FT != FAIRNESS_FRAME && frame.destinationMacAddress == myMacAddress && ((frame.RI == myRingletID) || (mystate.wrapped && mystate.wrappingMethod == centerWrap))	29		
	frame.sourceMacAddress == myMacAddress && ((frame.RI == myRingletID) || (mystate.wrapped && mystate.wrappingMethod == centerWrap))	30		
	single transit-queue implementation	31	moveToPTQ()	WAIT
	frame.SC == CLASS_A0	32		
	frame.SC == CLASS_A1	33		
	—	34	moveToSTQ()	
WAIT	next receive frame available	35	—	CHECK1
	—	36	—	WAIT

Row 6.17-1: If the header CRC or parity does not match its expected value, discard the frame.
Row 6.17-2: Expired frames are rejected.
Row 6.17-3: Strict data frames on the primary ringlet with an inconsistent timeToLive are discarded.
Row 6.17-4: If a fairness or idle frame is an invalid length, discard the frame.
Row 6.17-5: Multicast source addresses are illegal and therefore rejected.
Row 6.17-6: Steer-only frames are rejected when apparently wrapped.
Row 6.17-7: Wrapped frames are rejected when the ring protection state is IDLE.
Row 6.17-8: Wrapped frames are rejected when the ring is a steering ring.

Row 6.17-9: A good payload CRC is processed normally.
Row 6.17-10: A stomped payload CRC is processed normally.
Row 6.17-11: A bad payload CRC is stomped (as described in F.2.3) and then processed normally.

Row 6.17-12: Idle frames are accepted unconditionally, but not processed.
Row 6.17-13: Fairness frames are accepted without checking ringletID or destination address.
Row 6.17-14: Topology frames received on the wrong ringlet are accepted by the topology control unit.
Row 6.17-15: Wrapped frames are not accepted when returning on their opposing ringlet.

Editors' Notes: *To be removed prior to final publication.*

Data frames and control frames are not accepted on the opposing ringlet. Should fairness frames be accepted?

Row 6.17-16: All data frames are accepted if the station is in promiscuous mode or if the frame is flooded.
Row 6.17-17: Unicast frames are accepted at their destination station.
Row 6.17-18: Multicast frames whose destination address is supported by this station are accepted.
Row 6.17-19: Frames are rejected after returning to their source.
Row 6.17-20: Otherwise, frames are not accepted.

Row 6.17-21: Accepted control frames are copied to the control sublayer of the MAC.
Row 6.17-22: Data frames are rejected upon returning to their source. (Control and fairness frames will be stripped **after** returning to their source, in row 6.17-29.)
Row 6.17-23: Otherwise (frame.FT == DATA_FRAME), accepted (data) frames are passed to the client.

Row 6.17-24: Wrapped frames do not get TTL decremented when returning on their opposing ringlet.
Row 6.17-25: Otherwise, the *timeToLive* field is always updated on transiting frames.

Row 6.17-26: Expired frames are discarded.
Row 6.17-27: SC-FCM frames are not transited.
Row 6.17-28: Steer-only frames are not transited when a wrapped station is reached.
Row 6.17-29: Control and data frames are not transited after reaching their destination, on the correct ringlet or at the wrapping point for center wrapping stations. (Fairness frames have no destination.)
Row 6.17-30: Control and fairness frames are not transited after returning to their source, on the correct ringlet or at the wrapping point for center wrapping stations. (Data frames returning to this station were stripped in row 6.17-22.)
Row 6.17-31: In single-queue designs, all transiting frames are placed into the primary transit queue.
Row 6.17-32: The transiting classA, subclassA0 frames are placed into the primary transit queue.
Row 6.17-33: The transiting classA, subclassA1 frames are placed into the primary transit queue.
Row 6.17-34: The transiting classB and classC frames are placed into the secondary transit queue.

Row 6.17-35: When a new frame is received, start checking it.
Row 6.17-36: Wait for a new frame to be received.

6.9 Transmit operation

If a station is operating in a passthru mode, it is no longer considered a station on the ring, and it does not participate in the transmit operation. Station passthru can be thought of as being equivalent to an optical regenerator where the MAC operates as a transit station. The means by which a MAC enters passthru, if at all, is a local implementation issue. In passthru mode, no local client or MAC frames are added, and all frames on the data path are transited without modification.

The transmit operation is initiated upon receipt of a MA_DATA.request, as described in 5.3.1. The transmit operation consists of ringlet selection (described in 6.3.5), followed by stage queue processing (described in 6.9.1), followed by transmit selection (described in 6.9.8 or 6.9.11).

6.9.1 Stage queue state machine

A logical FIFO stage queue is provided for client transmission without client access delay. The size of the stage queue is dependent upon the implementation. Implementations not implementing any physical structure for the stage queue may view the stage queue as a zero length queue.

NOTE—The standard does not define any minimum or maximum size for the stage 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 rate limiting 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 must contain at least the header.

A frame in a 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. Acceptance into the stage queue, and therefore for transmission, is governed by the per class shapers, the aggregate downstream shaper (shaperD), and the fairness algorithm (as defined in Clause 9).

6.9.1.1 Stage queue flow chart

Acceptance of frames from the MAC client 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 precedence of acceptance of frames is shown in Figure 6.11.

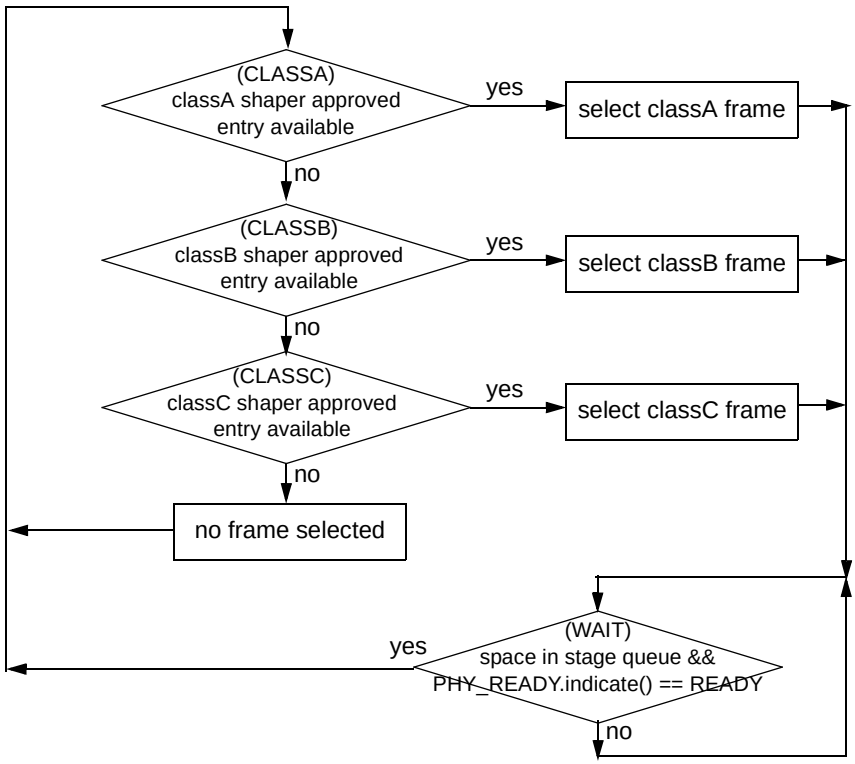


Figure 6.11—Stage queue frame selection

6.9.1.2 Inputs

creditA0
: The credits value for the subclassA0 shaper, as described in 6.7.5.

creditA1
: The credits value for the subclassA1 shaper, as described in 6.7.5.

creditB
: The credits value for the classB shaper, as described in 6.7.6.

creditC
: The credits value for the fairness eligible shaper handling traffic traveling short of the congestion point, as described in 6.7.7.

creditCc
: The credits value for the fairness eligible shaper handling traffic traveling beyond the congestion point, as described in 6.7.7.

frame
: The output from the ringlet selection state machine, as described in 6.3.5.5.

markFE
: The *markFE* parameter value passed in the MA_DATA.request() primitive, as defined in 5.3.1.

passA0
: The output from the subclassA0 shaper, as described in 6.7.5.

passA1
: The output from the subclassA1 shaper, as described in 6.7.5.

sendA
: The output from the sendA state machine, as described in 6.6.2.

sendB
The output from the sendB state machine, as described in 6.6.3.
sendC
The output from the sendC state machine, as described in 6.6.4.

6.9.1.3 Constants

CLASS_A0
A *serviceClass* value defined in 8.2.2.4.
CLASS_A1
A *serviceClass* value defined in 8.2.2.4.
CLASS_B
A *serviceClass* value defined in 8.2.2.4.
CLASS_C
A *serviceClass* value defined in 8.2.2.4.

6.9.1.4 Variables

No variables are used by this state machine.

6.9.1.5 Functions

classAEntryAvailable
To indicate if a classA entry is available from the client.
Values:
TRUE: A classA entry is available.
FALSE: A classA entry is not available.
classBEntryAvailable
To indicate if a classB entry is available from the client.
Values:
TRUE: A classB entry is available.
FALSE: A classB entry is not available.
classCEntryAvailable
To indicate if a classC entry is available from the client.
Values:
TRUE: A classC entry is available.
FALSE: A classC entry is not available.
frameLength
Returns the length of a frame.
Value: The number of bytes in the frame.
moveToStage
Moves the frame to the end of the stage queue of the ringlet which was selected.
stageQueueAvailable
To indicate if the stage queue is available to accept an entry from the client.
Values:
TRUE: The stage queue is available.
FALSE: The stage queue is not available.

6.9.1.6 Stage queue state table

The frame selection state machine specified in Table 6.18, where rows are evaluated in top-to-bottom order, implements the functions necessary for selecting a client frame for admission to the stage queue and the associated frame header processing. 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 machine is described in 3.4.

Table 6.18—Stage queue frame selection states

Current state		Row	Next state	
state	**condition**		**action**	**state**

CLASSA	classAEntryAvailable() && passA0 && sendA	1	frame.FE = 0;
			frame.SC = CLASS_A0;
			creditA0 -=
			frameLength(frame);
			moveToStage();
				WAIT

	classAEntryAvailable() && passA1 && sendA	2	frame.FE = 0;
			frame.SC = CLASS_A1;
			creditA1 -=
			frameLength(frame);
			moveToStage();	

	—	3	—	CLASSB
CLASSB	classBEntryAvailable() && sendC && markFE	4	frame.FE = 1; frame.SC = CLASS_B;	CONGESTED
	classBEntryAvailable() && markFE	5	—	CLASSC

	classBEntryAvailable() && sendB	6	frame.FE = 0;
			frame.SC = CLASS_B;
			creditB -=
			frameLength(frame);
			moveToStage();
				WAIT

	classBEntryAvailable() && sendC	7	frame.FE = 1; frame.SC = CLASS_B;	CONGESTED
	—	8	—	CLASSC
CLASSC	classCEntryAvailable() && sendC	9	frame.FE = 1; frame.SC = CLASS_C;	CONGESTED
	—	10	—	WAIT

Table 6.18—Stage queue frame selection states *(continued)*

Current state		Row	Next state	
state	**condition**		**action**	**state**
CONGESTED	frame.timeToLive <= hopsToCongestion	11	creditC -= frameLength(frame); moveToStage();	WAIT
	—	12	creditCc -= frameLength(frame); moveToStage();	
WAIT	stageQueueAvailable() && PHY_READY.inidicate() == READY	13	—	CLASSA
	—	14	—	WAIT

Row 6.18-1: ClassA frames within the bounds of shaperA0 are marked subclassA0 and added to the stage queue.
Row 6.18-2: ClassA frames within the bounds of shaperA1 are marked subclassA1 and added to the stage queue.
Row 6.18-3: No classA frames to be transmitted at this time.

Row 6.18-4: ClassB frames within the bounds of shaperC and requested to be marked *fairnessEligible* are marked *fairnessEligible*.
Row 6.18-5: ClassB frames requested to be marked *fairnessEligible* and with no shaperC credits are not accepted for transmission.
Row 6.18-6: ClassB frames within the bounds of shaperB are marked not *fairnessEligible* and added to the stage queue.
Row 6.18-7: ClassB frames within the bounds of shaperC are marked *fairnessEligible*.
Row 6.18-8: No classB frames to be transmitted at this time.

Row 6.18-9: ClassC frames within the bounds of shaperC are marked *fairnessEligible*.
Row 6.18-10: No classC frames to be transmitted at this time. No frame is added to the stage queue.

Row 6.18-11: Fairness eligible frames traveling short of the congestion point are added to the stage queue.
Row 6.18-12: Fairness eligible frames traveling beyond the congestion point are added to the stage queue.

Row 6.18-13: There is space for a frame to be added to the stage queue and the physical layer has indicated that it is ready.
Row 6.18-14: Wait for space to become available for a frame to be added to the stage queue.

6.9.2 Data frame processing

Data frames transmitted by a station shall set the following values:

a) *ttlBase* shall be set to the value of *timeToLive.*
b) *pastSource* shall be set to 0.
c) *floodingForm* shall be set to 01_2 for frames that will be unidirectionally flooded over the ring. It shall be set to 10_2 for frames that will be bidirectionally flooded over the ring. The value of 01_2 or 10_2 is typically set if:
    1) A bridging client is requesting the transmission of the frame.

2) A RPR client is requesting the transmission of a frame whose *destinationMACAddress* is not local to the ring.

Otherwise, this bit is set to 00_2.

d) *extendedFrame* shall be set to 1 when a client provided frame is contained after the HEC field. For example, if the frame is transmitted in strict mode (i.e., *strictOrder* is set to 1), then the value is set to 1 if a bridging client is requesting the transmission of the frame.
NOTE: An enhanced bridging client may want to set the *extendedFrame* to 1 independent of the setting of the *strictOrder* bit.
Otherwise, this bit is set to 0.

e) *sourceMACAddress* shall be set to the source station MAC address when transmitting a frame in strict mode (i.e., *strictOrder* is set to 1).

f) *strictOrder* shall be set to 1 when strict ordering of this frame is required, and shall be set to 0 when relaxed ordering of this frame is allowed.

6.9.3 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 described in Clause 9.

Single-queue implementations measure congestion with access delay timers. There are separate access delay timers for 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. If the MAC is unable to transmit the frame before the timer expires, the MAC treats this the same as having exceeded the *lowThreshold* condition as described below.

The values of the access delay timers should be based on the acceptable MAC end-to-end delay for classB and classC traffic. The default value should be set to 5 ms for classC and to 1 ms for classB. The range is from 1 ms to 100 mS in 1 ms increments.

Editors' Notes: *To be removed prior to final publication.*

There is no access delay timer for classA traffic. An access delay timer expiring for classA traffic would mean there is something wrong with respect to provisioning. Perhaps OAM should look into providing an alarm for this condition.

Dual-queue implementations measure congestion with STQ thresholds. There are both a low threshold, *lowThreshold*, and high threshold, *highThreshold*, 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.

6.9.4 Transmit rate synchronization

RPR stations and networks may be implemented with either synchronous or asynchronously-timed PHYs. In a synchronous network, the transmit clock at each station is locked to a clock recovered from the received data stream, 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 nominal 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.

This subclause 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 datastream. The mechanism for varying the number of idle frames based on the primary transit queue (PTQ) level is described in 6.7.3.

The transmit rate synchronization function is part of the MAC datapath. Although designed for operation with asynchronously-timed stations, this function may 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.

6.9.5 Flooding

Two flooding alternatives for data frames are provided. They are:

Unidirectional: A frame forwarding transfer involving sending a flooding frame on only one ringlet, to all stations on that ringlet[3]. The *sourceMACAddress* found in the *rprHeader* is the local source MAC address. The *floodingForm* field is set to unidirectional for this flooding alternative.

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 *timeToLive* within the *rprHeader*. The *floodingForm* field is set to bidirectional for this flooding alternative.

A variety of remote-transfer flows are illustrated in the following subclauses, as illustrated in Figure 6.12. Downward and upward arrows identify client-to-MAC and MAC-to-client transfers, respectively. Downward end-of-flow curves identify locations where frames are stripped. The *x* marker at the end of an error identifies locations where frames are discarded, due to detected inconsistency errors.

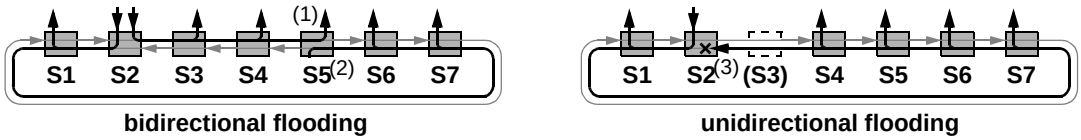


Figure 6.12—Flooded receive operations

When a ring is operating with wrapping protection, bidirectional flooding is not a requirement.

NOTE—The efficiency of bidirectional flooding in support of enhanced bridging argues that supporting bidirectional flooding when wrapping makes little sense. This is due to the inefficiency of sending some of the frames all the way around the opposite ringlet to be delivered to some of the stations. It is clearly more bandwidth efficient to avoid that path entirely and steer the frames instead.

For completeness, the following subclauses elaborate on the various possible flooding and protection combinations.

6.9.5.1 Flooding with steered protection

Steered flooding involves concurrent bidirectional transmissions with distinctive nonadjacent stations. This is illustrated in Figure 6.13 with station S1 and station S7 as the failure-point destinations.

From the client's perspective, flooding on steered or wrapped rings has the same behavior, although the paths of frames changes due to the wrapping at failure points.

[3]A potentially more efficient form of unidirectional flooding can also be supported. This is unidirectional transmission that terminates at the source station's upstream neighbor. The *sourceMACAddress* found in the *rprHeader* is the local source address. The *timeToLive* and *ttlBase* fields in the *rprHeader* are set to numberOfStations-1, upon transmission by the source station. The *floodingForm* field is set to unidirectional.

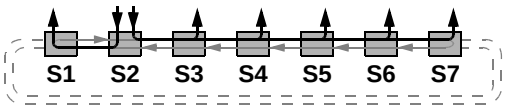


Figure 6.13—Steered flooding

6.9.5.2 Bidirectional flooding

Bidirectional flooding of a ring involves concurrent bidirectional transmissions, typically directed to a pair of mid-point stations, as illustrated in the left and right sides of Figure 6.14. A *floodingForm* of bidirectional is specified for both copies of the flooded frame, causing the flooded frame to be passed to the client at each of its removal stations.

Figure 6.14—Bidirectional flooding

Bidirectional floods are ***not*** wrap eligible.

6.9.5.3 Unidirectional flooding

Unidirectional flooding involves either a unidirectional transmission directed to the source station, illustrated in both the left and right sides of Figure 6.15. A *floodingForm* of unidirectional is specified, regardless of which direction is selected.

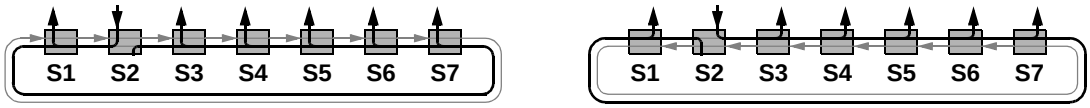


Figure 6.15—Unidirectional flooding

Unidirectional flooding with wrapped protection involves either a west side or east side unidirectional transmission directed to the source station, illustrated in both the left and right side of Figure 6.16. The wrapped flooding operation relies on the wrap capability at the endpoints. A *floodingForm* of unidirectional is specified, regardless of which direction is selected.

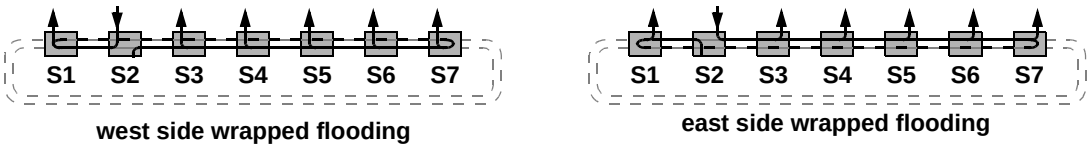


Figure 6.16—Unidirectional flooding with wrapped protection

6.9.5.4 Flood copy rules

Flooding involves selectively copying non-deleted primary-run frames to the client, if a *floodingForm* indication of unidirectional or bidirectional is encountered. Frames are stripped from the ring based upon existing frame stripping rules outlined in 6.8.

6.9.5.5 Unicast considerations

Local stations improve efficiencies by directing local-unicast traffic to the affected station, rather than flooding this traffic to all others, as illustrated in Figure 6.17. The determination of whether to use flooded or uni-

cast frames is based on a comparison of the frame's *destinationMacAddress* with the RPR topology database. A unicast transmission is used if a local station matches the same *destinationMacAddress*; a flood is used otherwise.

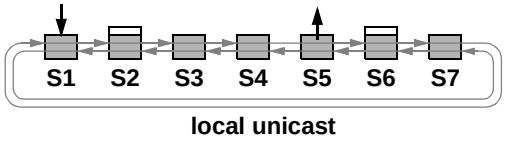


Figure 6.17—Local unicasts

6.9.5.6 Multicast/broadcast forwarding

The most basic multicast/broadcast distribution techniques involves circulating a frame through all stations on the ring.

6.9.6 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.

6.9.7 Single-queue MAC data paths

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 shown in Figure 6.18. The PTQ has the behavior of a small FIFO.

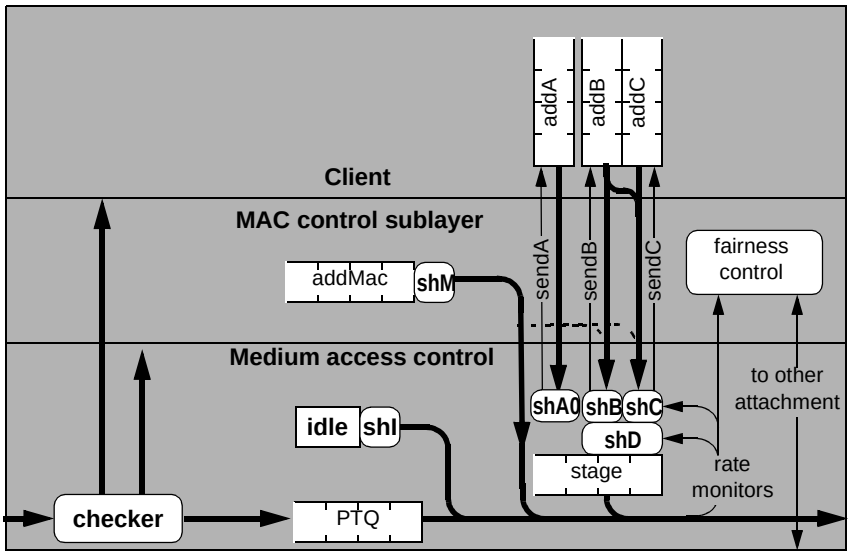


Figure 6.18—Single-queue MAC data path

Frames from the MAC control add queue are rate-limited by shaperM. All classA traffic is rate-limited by shaperA0 only since single-queue implementations can not support subclassA1 add traffic.

A FIFO ordering shall be maintained when entries pass through the PTQ.

6.9.8 Single-queue transmit state machine

The transmit behavior of a single-queue MAC is described by its transmit-selection protocols (described 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 sendA is required to ensure conformance of classA frame transmissions.

6.9.8.1 Single-queue transmit flow chart

A single-queue MAC can transmit data frames from three possible internal queues per ringlet, as illustrated in Figure 6.19.

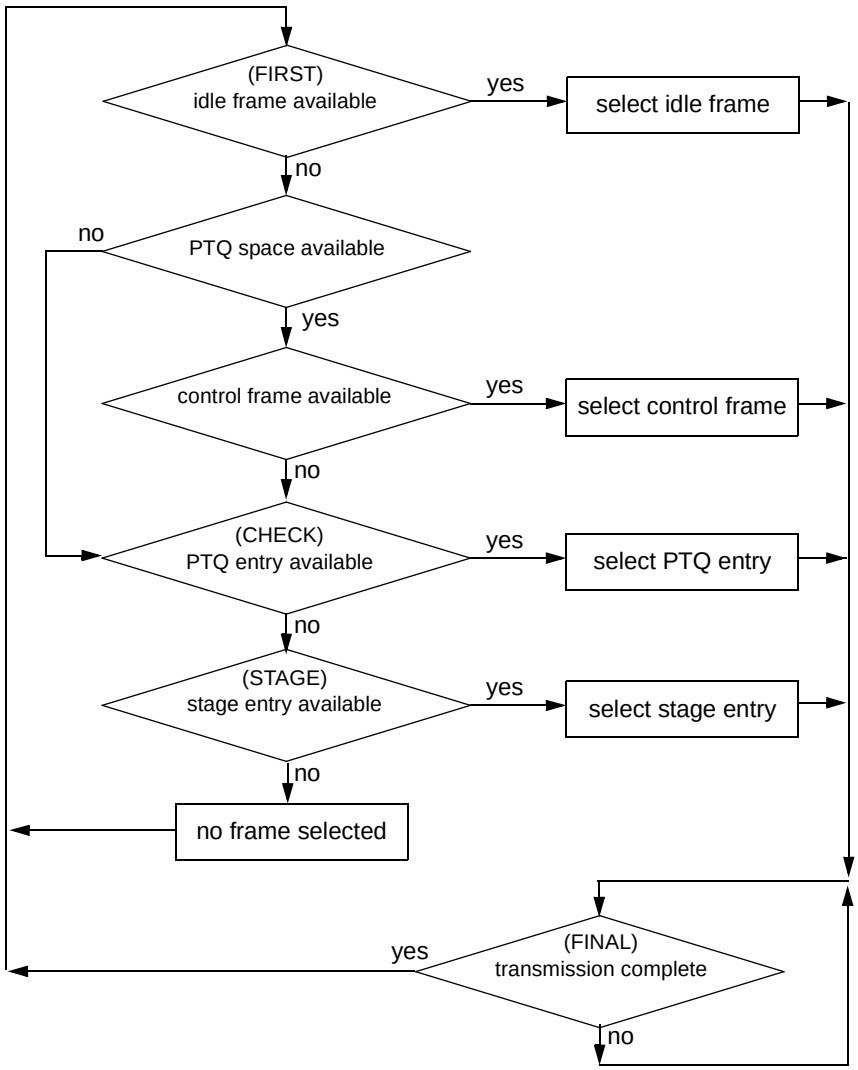


Figure 6.19—Single-queue transmit selection

6.9.8.2 Inputs

passM
: The output from the MAC control shaper, as described in 6.7.4.

sizeOfMacControl
: The size, in bytes, of the queued data within the MAC control queue.

spaceInPTQ
The size, in bytes, of the remaining space in the primary transit queue.

6.9.8.3 Constants

No constants are used by this state machine.

6.9.8.4 Variables

No variables are used by this state machine.

6.9.8.5 Functions

entryInMacControl
To indicate if there is an entry in the MAC control queue.
Values:
TRUE: There is an entry in the MAC control queue.
FALSE: There is not an entry in the MAC control queue.

entryInMacIdle
To indicate if there is an entry in the MAC idle queue.
Values:
TRUE: There is an entry in the MAC idle queue.
FALSE: There is not an entry in the MAC idle queue.

entryInPTQ
To indicate if there is an entry in the primary transit queue.
Values:
TRUE: There is an entry in the primary transit queue.
FALSE: There is not an entry in the primary transit queue.

entryInStage
To indicate if there is an entry in the stage queue.
Values:
TRUE: There is an entry in the stage queue.
FALSE: There is not an entry in the stage queue.

transmitComplete
To indicate if the transmission of the previous frame, if any, has completed, such that the MAC is ready to transmit another frame.
Values:
TRUE: The transmission has completed.
FALSE: The transmission has not completed.

txMacIdle
Transmits the frame at the head of the MAC idle queue.

txMacControl
Transmits the frame at the head of the MAC control queue.

txPTQ
Transmits the frame at the head of the primary transit queue.

txStage
Transmits the frame at the head of the stage queue.

6.9.8.6 Single-queue transmit state table

The single-queue transmit frame selection state machine specified in Table 6.19, where rows are evaluated in top-to-bottom order, 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. This is done to avoid overflow of the PTQ by enqueueing additional incoming transit frames during client 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 machine is described in 3.4.

Table 6.19—Single-queue transmit selection states

Current state		Row	Next state	
state	**condition**		**selection**	**state**
FIRST	entryInMacIdle()	1	txMacIdle();	FINAL
	!entryInMacControl() || passM == 0	2	—	CHECK
	sizeOfMacControl > spaceInPTQ	3		
	—	4	txMacControl();	FINAL
CHECK	entryInPTQ()	5	txPTQ();	FINAL
	—	6	—	STAGE
STAGE	entryInStage()	7	txStage();	FINAL
	—	8	—	
FINAL	transmitComplete()	9	—	FIRST
	—	10	—	FINAL

Row 6.19-1: MAC idle transmissions have precedence over all other transmissions.
Row 6.19-2: Nothing is in the MAC control add queue, or there are no MAC control transmission credits, so other transmission sources are checked.
Row 6.19-3: The size of the queued MAC control frame shall be less than the available PTQ space.
Row 6.19-4: MAC control transmissions have precedence over client and PTQ transmissions.

Row 6.19-5: The primary transit queue 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 transmissions.
Row 6.19-6: The MAC control queue transmissions have precedence over client transmissions.

Row 6.19-7: Nothing is in the PTQ, so choose the stage queue if it has an entry present.
Row 6.19-8: No frame is selected when no frame transmissions are possible.

Row 6.19-9: Wait until the transmission completes.
Row 6.19-10: The selected transmission/retransmission continues until completed.

6.9.9 Dual-queue MAC design

The transmit behavior of a dual-queue MAC is described by its transmit-selection protocols (described in this subclause) and shaping functions. The transmit-selection protocol and shaping functions are largely independent, but coupling (via the internal shaper provided sendM and sendA indications) is required to ensure conformance of MAC-supplied classA control frame transmissions. Additional coupling (via the internal shaper provided sendD indication) is required to properly sustain downstream classA transmissions.

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 following subclauses.

6.9.10 Dual-queue MAC data paths

A dual-queue MAC makes use of two transit queues, as shown in Figure 6.20.

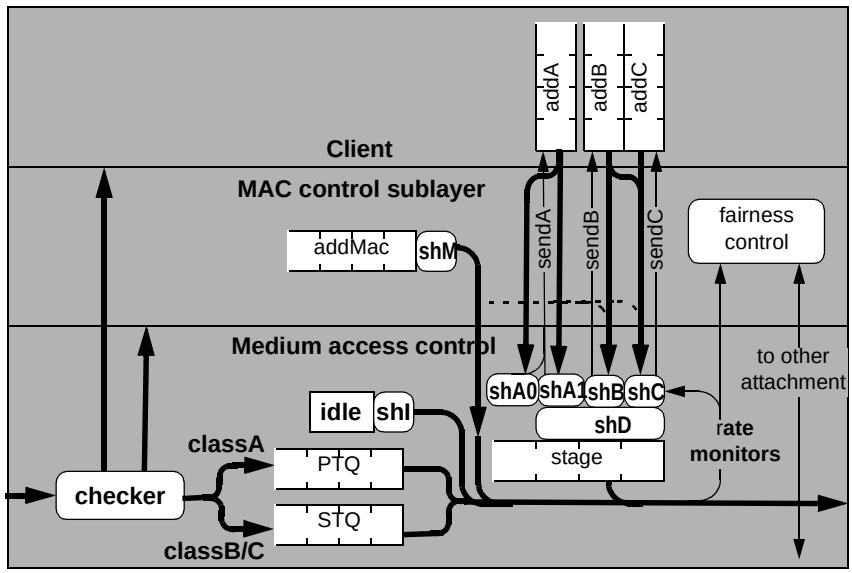


Figure 6.20—MAC data paths

Frames from the MAC control add queue are rate-limited by shaperM. All client classA traffic is rate-limited by shaperA0 and shaperA1.

The following externally visible behaviors shall be supported:

a) PTQ ordering. A FIFO ordering shall be maintained when entries pass through the PTQ.
b) STQ ordering. A FIFO ordering shall be maintained when entries pass through the STQ.
c) Cross ordering. An entry from the STQ shall not be output before a previously received PTQ entry.

6.9.11 Dual-queue transmit state machine

The transmit behavior of a dual-queue MAC is described by its transmit-selection protocols (described 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 sendA is required to ensure conformance of classA frame transmissions.

6.9.11.1 Dual-queue transmit flow chart

A dual-queue MAC can transmit data frames from four possible internal queues per ringlet, as illustrated in Figure 6.21.

.

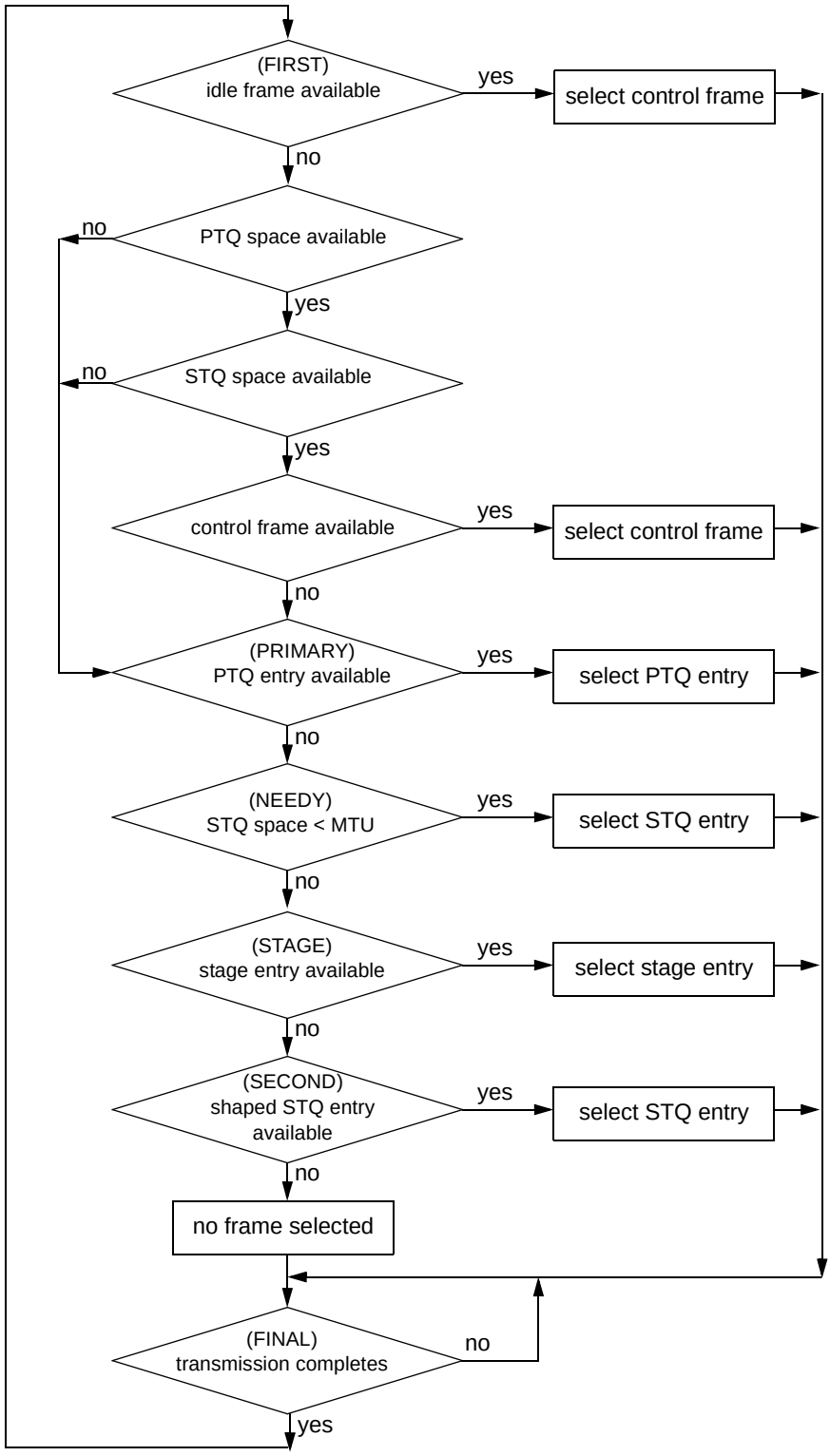


Figure 6.21—Dual-queue transmit selection

6.9.11.2 Inputs

passD
The output from the downstream shaper, as described in 6.7.8.
passM
The output from the MAC control shaper, as described in 6.7.4.
sizeOfMacControl
The size, in bytes, of the queued data within the MAC control queue.
sizeMTU
The size, in bytes, of the maximum transfer unit, as defined in 8.2.
spaceInPTQ
The size, in bytes, of the remaining space in the primary transit queue.
spaceInSTQ
The size, in bytes, of the remaining space in the secondary transit queue.

6.9.11.3 Constants

No constants are used by this state machine.

6.9.11.4 Variables

No variables are used by this state machine.

6.9.11.5 Functions

entryInMacControl
To indicate if there is an entry in the MAC control queue.
Values:
TRUE: There is an entry in the MAC control queue.
FALSE: There is not an entry in the MAC control queue.
entryInMacIdle
To indicate if there is an entry in the MAC idle queue.
Values:
TRUE: There is an entry in the MAC idle queue.
FALSE: There is not an entry in the MAC idle queue.
entryInPTQ
To indicate if there is an entry in the primary transit queue.
Values:
TRUE: There is an entry in the primary transit queue.
FALSE: There is not an entry in the primary transit queue.
entryInStage
To indicate if there is an entry in the stage queue.
Values:
TRUE: There is an entry in the stage queue.
FALSE: There is not an entry in the stage queue.
entryInSTQ
To indicate if there is an entry in the secondary transit queue.
Values:
TRUE: There is an entry in the secondary transit queue.
FALSE: There is not an entry in the secondary transit queue.
transmitComplete
To indicate if the transmission of the previous frame, if any, has completed, such that the MAC is

ready to transmit another frame.
Values:
TRUE: The transmission has completed.
FALSE: The transmission has not completed.

txMacIdle
Transmits the frame at the head of the MAC idle queue.

txMacControl
Transmits the frame at the head of the MAC control queue.

txPTQ
Transmits the frame at the head of the primary transit queue.

txStage
Transmits the frame at the head of the stage queue.

txSTQ
Transmits the frame at the head of the secondary transit queue.

6.9.11.6 Dual-queue transmit state table

The dual-queue transmit frame selection state machine specified in Table 6.20, where rows are evaluated in top-to-bottom order, implements the functions necessary for selecting which frame to transmit in a dual-queue MAC. 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 machine is described in 3.4.

Table 6.20—Dual-queue transmit selection states

Current state		Row	Next state	
state	**condition**		**selection**	**state**
FIRST	entryInMacIdle()	1	txMacIdle();	FINAL
	!entryInMacControl() || passM == 0	2	—	PRIMARY
	sizeOfMacControl > spaceInPTQ	3		
	sizeOfMacControl > spaceInSTQ	4		
	—	5	txMacControl();	FINAL
PRIMARY	entryInPTQ()	6	txPTQ();	FINAL
	—	7	—	NEEDY
NEEDY	spaceInSTQ < sizeMTU	8	txSTQ();	FINAL
	—	9	—	STAGE
STAGE	passD==0 || (entryInStage() && frame.SC == CLASS_A0)	10	—	SECOND
	entryInStage()	11	txStage();	FINAL
	—	12	—	SECOND

Table 6.20—Dual-queue transmit selection states *(continued)*

Current state		Row	Next state	
state	**condition**		**selection**	**state**
SECOND	entryInSTQ()	13	txSTQ();	FINAL
	—	14	—	
FINAL	transmitComplete()	15	—	FIRST
	—	16	—	FINAL

Row 6.20-1: MAC idle transmissions have precedence over all other transmissions.
Row 6.20-2: Nothing is in the MAC control add queue, or there are no MAC control transmission credits, so other transmission sources are checked.
Row 6.20-3: Validate that the MAC control frame is less than the available PTQ space.
Row 6.20-4: Validate that the MAC control frame is less than the available STQ space.
Row 6.20-5: MAC control transmissions have precedence over transit and client transmissions.

Row 6.20-6: 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 6.20-7: In the absence of primary-transit-queue frames, other transmission sources are checked.

Row 6.20-8: The secondary transit queue is emptied when less than one MTU remains free, to avoid overflows.
Row 6.20-9: In the absence of secondary-transit-queue overflow threats, other transmission sources are checked.

Row 6.20-10: When necessary to sustain downstream classA traffic, non-subclassA0 add traffic is ignored.
Row 6.20-11: The stage queue entry preempts STQ transmissions, since provision checks were done previously.
Row 6.20-12: In the absence of stage-queue frames, other transmission sources are checked.

Row 6.20-13: The secondary transit queue is serviced when an entry is available.
Row 6.20-14: The secondary transit queue is ignored when no frame is available.

Row 6.20-15: The next selection occurs after the transmission completes.
Row 6.20-16: The selected transmission/retransmission continues until completed.

6.10 Wrappable data paths

Wrapping of frames is controlled by several conditions. First, the ring must be a wrapping ring, as described in 11.4. Next the local station be wrapped, as determined by myState.wrapped and as described in 6.8. Finally, the frame to be wrapped must have the *WE* bit set, as described in 8.2.2.5.

Each wrapping-capable station has wrappable data paths that allow frames to loop back to the opposing ringlet after link failures are detected. There are 2 methods of implementing wrapping, each with different effects on client add traffic behavior, but with no effect on the behavior of the ringlet data paths. Wrapping systems may choose either method.

NOTE—Center wrap is usually chosen to allow for implementation using dual line card redundancy. Edge wrap is usually chosen because transit buffer contents are not discarded upon wrap, and because a simpler client implementation is made possible by allowing the client to always use the same physical signals to communicate with the MAC.

6.10.1 Center wrap

The center wrap implementation is shown in Figure 6.22.

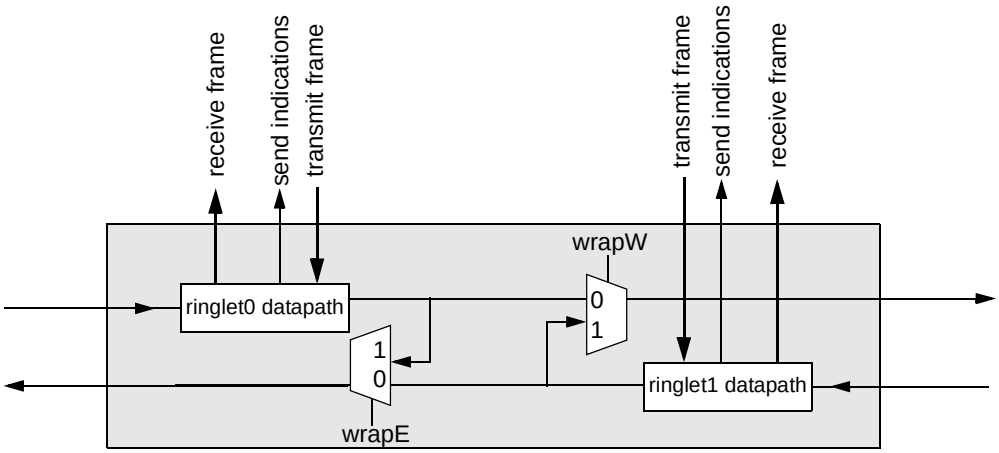


Figure 6.22—Center wrap data paths

6.10.2 Edge wrap

The edge wrap implementation is shown in Figure 6.23.

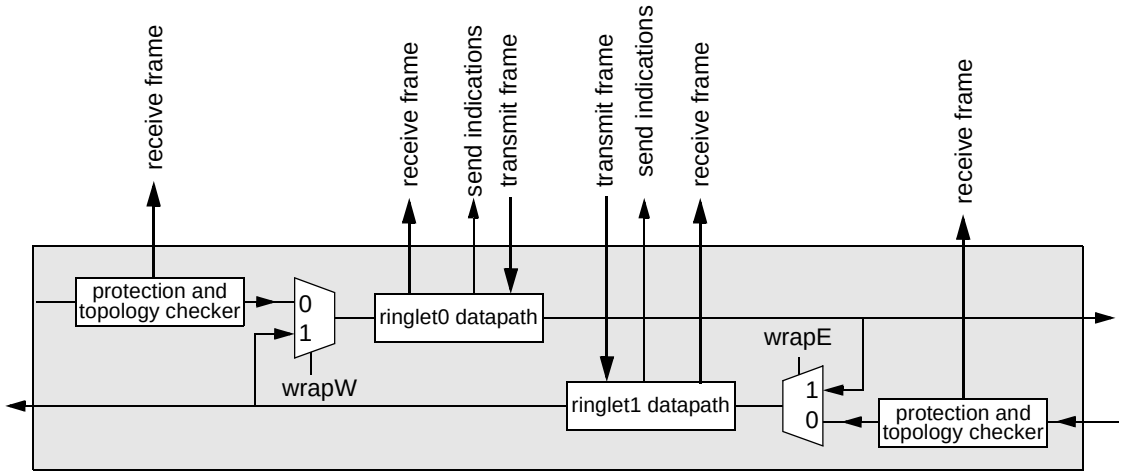


Figure 6.23—Edge wrap data paths

7. MAC physical interface

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	Revised according to WG comments for TF review.
Draft 0.3, June 2002	Revised draft for WG review according to comments on D0.2.
Draft 1.0, August 2002	Revised draft for TF review according to comments on D0.3.
Draft 1.1, October 2002	Revised draft for WG review according to comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

7.1 Overview

7.1.1 Scope

This clause specifies the resilient packet ring (RPR) medium access control (MAC) physical layer interfaces, and describes the physical layer entities (PHYs) specified for use in RPR systems.

This standard does not specify any new PHYs for RPR implementations. Rather, it defines two families of reconciliation sublayers for use with Ethernet and SONET/SDH PHYs. The reconciliation sublayers map the logical MAC physical layer service primitives to and from standard electrical interfaces used by the PHYs.

The Ethernet reconciliation sublayers provide interfaces to standard gigabit Ethernet and ten gigabit Ethernet LAN and WAN PHYs. Although this clause provides interfaces that allow RPR implementations to use standard Ethernet PHYs, these implementations do not provide Ethernet frame format compliant interfaces.

The SONET/SDH reconciliation sublayers provide interfaces to adaptation sublayers that specify either generic framing procedure (GFP) or byte-synchronous high-level data link control (HDLC)-like framing, for SONET/SDH networks and PHYs operating at 155 Mbps to 10 Gbps or higher.

A compliant RPR MAC implementation requires at least one of the reconciliation sublayers specified in this standard. Implementations may include more than one reconciliation sublayer.

A compliant RPR implementation may use either Ethernet or SONET/SDH interfaces and PHYs, and may operate at various data rates. No specific configuration is preferred or recommended over another by this standard.

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.

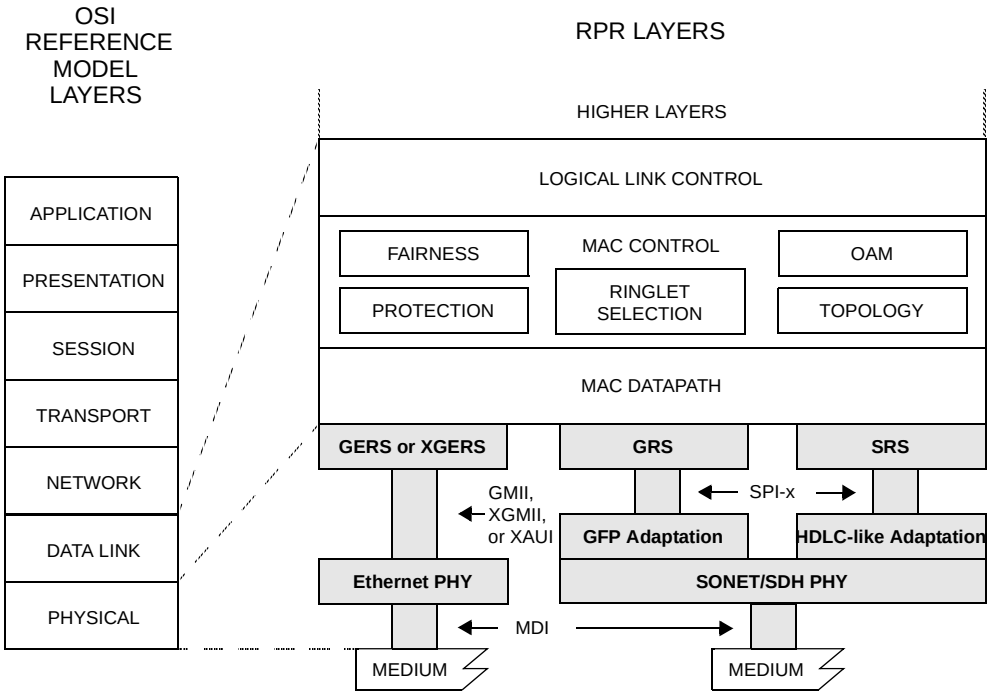
7.1.2 Objectives

The objectives of the RPR physical layer interfaces and PHYs are:

a) Support the resilient packet ring MAC;
b) Support gigabit Ethernet and ten gigabit Ethernet PHYs;
c) Support SONET/SDH PHYs using GFP or byte-synchronous HDLC-like framing and operating at speeds of 155 Mbps to 9.95 Gbps;
d) Support synchronous or asynchronous network applications;
e) Support full duplex operation only.

7.1.3 Relationship to other standards

Figure 7.1 shows the relationship between the RPR MAC, reconciliation sublayers, adaptation sublayers, and physical layers.



GMII = GIGABIT MEDIA INDEPENDENT INTERFACE
GERS = GIGABIT ETHERNET RECONCILIATION SUBLAYER
GRS = GFP RECONCILIATION SUBLAYER
MAC = MEDIUM ACCESS CONTROL
MDI = MEDIUM DEPENDENT INTERFACE
PHY = PHYSICAL LAYER ENTITY
SRS = SONET/SDH RECONCILIATION SUBLAYER
XAUI= 10 GIGABIT ATTACHMENT UNIT INTERFACE
XGMII = 10 GIGABIT MEDIA INDEPENDENT INTERFACE
XGERS = 10 GIGABIT ETHERNET RECONCILIATION SUBLAYER

Figure 7.1—RPR RS and PHY relationship to the ISO/IEC Open Systems Interconnection (OSI) reference model

7.2 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.indicate
— PHY_LINK_STATUS.indicate

— PHY_READY.indicate.

7.2.1 PHY_DATA.request

7.2.1.1 Function

This primitive defines the transfer of data from the MAC to the reconciliation sublayer.

7.2.1.2 Semantics of the service primitive

PHY_DATA.request (OUTPUT_FRAME, LENGTH)

The OUTPUT_FRAME parameter takes the value of a complete RPR frame, and represents the transfer of an RPR frame from the MAC to the reconciliation sublayer.

The LENGTH parameter is optional, and is used by some reconciliation sublayers to indicate the length of the RPR frame. The value of this parameter represents the length of the associated OUTPUT_FRAME, or may be undefined if not used by the reconciliation sublayer.

7.2.1.3 When generated

This primitive is generated by the MAC sublayer to request the transmission of a frame on the physical medium.

7.2.1.4 Effect of receipt

The effect of receipt of this primitive and a detailed mapping of this primitive to specific electrical interfaces are described in Annexes C and D.

7.2.2 PHY_DATA.indicate

7.2.2.1 Function

This primitive defines the transfer of data from the reconciliation sublayer to the MAC.

7.2.2.2 Semantics of the service primitive

PHY_DATA.indicate (INPUT_FRAME, STATUS, LENGTH)

The INPUT_FRAME parameter takes the value of a frame, and represents the transfer of an RPR frame from the reconciliation sublayer to the MAC.

The STATUS parameter takes the value of OK or ERROR and is used to identify frames that are received with error indications from the PHY, for example, due to assertion of a error signal on the PHY electrical interface. A value of OK indicates no error indication. A value of ERROR indicates that the PHY signaled an error during frame transmission to the reconciliation sublayer.

The LENGTH parameter is optional, and is used by some reconciliation sublayers to indicate the length of the RPR frame. The value of this parameter represents the length of the associated INPUT_FRAME, or is null if not used by the reconciliation sublayer.

7.2.2.3 When generated

The definition of when this primitive is generated and a detailed mapping of specific electrical interfaces to this primitive are described in Annexes C and D.

7.2.2.4 Effect of receipt

The effect of receipt of this primitive by the MAC sublayer is specified in Clause 6.

Editors' Notes: *To be removed prior to final publication.*

D0_2 comment #387 specified that the preceding paragraph will identify the subclause that defines the effect of the PHY_DATA.indicate primitive, but the subclause number was unavailable when this text was generated. This will be done in the next revision.

7.2.3 PHY_LINK_STATUS.indicate

7.2.3.1 Function

This primitive conveys the status of the physical link as detected by the PHY to the MAC.

7.2.3.2 Semantics of the service primitive

PHY_LINK_STATUS.indicate (LINK_STATUS)

The LINK_STATUS parameter takes the value of OK, FAIL or DEGRADE, and signifies the status of the link as detected by the PHY. OK indicates the absence of link degradation or fault conditions. DEGRADE indicates a link degradation condition. FAIL indicates a link fault condition. Detection and signaling of OK and FAIL are supported by all reconciliation sublayers. Detection and signaling of DEGRADE is not supported by all reconciliation sublayers.

7.2.3.3 When generated

The PHY_LINK_STATUS.indicate 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 Annexes C and D.

7.2.3.4 Effect of receipt

The effect of receipt of this primitive by the MAC sublayer is specified in Clause 11.

7.2.4 Mapping of PHY_READY.indicate

7.2.4.1 Function

This primitive indicates whether the reconciliation sublayer is ready to accept a new MAC frame.

7.2.4.2 Semantics of the service primitive

PHY_READY.indicate (READY_STATUS)

The READY_STATUS parameter takes the value of READY or NOT_READY, and signifies whether the reconciliation sublayer is able to accept a frame from the MAC for transmission on the PHY.

7.2.4.3 When generated

The definition of when this primitive is generated and a detailed mapping of specific electrical interfaces to this primitive are described in Annexes C or D.

7.2.4.4 Effect of receipt

The effect of receipt of this primitive by the MAC sublayer is specified in Clause 6.

Editors' Notes: *To be removed prior to final publication.*

D0_2 comment #390 specified that the preceding paragraph will identify the subclause that defines the effect of the PHY_DATA.indicate primitive, but the subclause number was unavailable when this text was generated. This will be done in the next revision.

7.3 Ethernet physical layer interfaces and PHYs

7.3.1 Ethernet reconciliation sublayers

Two Ethernet reconciliation sublayers are defined to provide interfaces to Ethernet PHYs. The first is the Gigabit Ethernet Reconciliation Sublayer (GERS) that provides a standard interface for use with gigabit Ethernet PHYs. The second is the Ten Gigabit Ethernet Reconciliation Sublayer (XGERS) that provides a standard interface for use with ten gigabit Ethernet PHYs.

7.3.1.1 Gigabit Ethernet Reconciliation Sublayer (GERS)

The Gigabit Ethernet Reconciliation Sublayer (GERS) maps the MAC physical layer service primitives to and from the Gigabit Media Independent Interface (GMII) specified in Clause 35 of IEEE Std 802.3-2000. The requirements for the GERS are specified in Annex C.

7.3.1.2 Ten Gigabit Ethernet Reconciliation Sublayer (XGERS)

The Ten Gigabit Ethernet Reconciliation Sublayer (XGERS) maps the MAC physical layer service primitives to and from the Ten Gigabit Media Independent Interface (XGMII) specified in Clause 46 of IEEE Std 802.3-2000. The XGMII is an optional interface. An XGMII Extender Sublayer (XGXS) may be used to provide a Ten Gigabit Attachment Unit Interface (XAUI) as defined in Clause 47 of IEEE Std 802.3-2000. The XGERS shall provide the XGMII, or implement an XGXS and provide the XAUI. The requirements for the XGERS are specified in Annex C.

7.3.2 Ethernet physical layer entities (PHYs)

7.3.2.1 Gigabit Ethernet PHYs

The GERS supports gigabit Ethernet PHYs that meet the following requirements:

a) Support the GMII interface;
b) Provide full duplex operation;
c) Support a link BER of 10^{-12} or better.

PHYs meeting these requirements include the 1000BASE-SX, 1000BASE-LX, and 1000BASE-CX specified in IEEE Std 802.3-2000.

7.3.2.2 Ten gigabit Ethernet PHYs

The XGERS supports ten gigabit Ethernet PHYs that meet the following requirements:

a) Support the XGMII or XAUI interface;
b) Provide full duplex operation;
c) Support a link BER of 10^{-12} or better;
d) Operate at the LAN or WAN data rates defined for ten gigabit Ethernet by IEEE Std 802.3-2000.

PHYs meeting these requirements include the 10GBASE-LX4, 10GBASE-SR and SW, 10GBASE-LR and LW, and the 10GBASE-ER and EW.

7.4 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 GFP or byte-synchronous HDLC-like 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 either 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 packet length information to the GFP adaptation sublayer to optionally eliminate the need to calculate this 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.

The GFP and HDLC 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 ITU-T G.707 and G.783, and is not defined in this standard.

The SONET/SDH C2 signal labels specified in ITU-T G.707 to be used for GFP and HDLC-like framing sublayers are described in Table 7.1.

Table 7.1—C2 signal labels for GFP and HDLC-like framing

Adaptation sublayer type	C2 signal label
GFP framing	0x1B
HDLC-like framing	0x16

The functionality performed in the SONET/SDH layer is beyond the scope of this standard. The implementations must be compliant with ITU-T G.707 and G.783. 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 layer may include protection mechanisms in addition to those provided by the RPR MAC. Interoperation with these protection mechanisms is described in Clause 11.

7.4.1 SONET/SDH reconciliation sublayers

7.4.1.1 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) using 8-bit transmit and receive data paths and operating at 155 Mbps to 622 Mbps;
b) SPI Level 3 (SPI-3) using 32-bit transmit and receive data paths and operating at 155 Mbps to 2.5 Gbps;
c) SPI Level 4 Phase 1 (SPI-4.1) operating at 622 Mbps to 10 Gbps;
d) SPI Level 4 Phase 2 (SPI-4.2) operating at 622 Mbps to 10 Gbps.

Four versions of the SRS using these four interfaces are specified in Annex D.

7.4.1.2 GFP Reconciliation Sublayer (GRS)

Editors' Notes: *To be removed prior to final publication.*

The text does not currently define the format in which the PLI information is conveyed on the SPI interface, and does not specify how the PLI is originally generated. The MAC physical layer service interface has provisions to transfer the PLI as an optional field but doesn't specify where it originates.

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) using 8-bit transmit and receive data paths and operating at 155 Mbps to 622 Mbps;
b) SPI Level 3 (SPI-3) using 32-bit transmit and receive data paths and operating at 155 Mbps to 2.5 Gbps;
c) SPI Level 4 Phase 1 (SPI-4.1) operating at 622 Mbps to 10 Gbps;
d) SPI Level 4 Phase 2 (SPI-4.2) operating at 622 Mbps to 10 Gbps.

The GRS is identical to the SRS, except that the GRS additionally conveys packet length information to the GFP adaptation sublayer. Four versions of the GRS using the four electrical interfaces are specified in Annex D.

7.4.2 SONET/SDH adaptation sublayers

Two 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 GFP framing. The second adaptation sublayer uses byte-synchronous HDLC-like framing. These two different adaptation layers do not interoperate with each other.

7.4.2.1 Generic Framing Procedure (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. GFP defines a frame format for protocol data units (PDUs) transferred between GFP initiation and termination points.

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, as described in Table 7.2.

Table 7.2—GFP EXI and UPI values for RPR

GFP Parameter	Value
Extension Header Identifier (EXI)	EXI = 0000_2
User Payload Identifier (UPI)	UPI = 0000 1001_2

The GFP Reconciliation Sublayer (GRS) and the MAC sublayer provide an optional capability to propagate the PDU length value to optimize the forwarding of PDUs with minimal delay.

7.4.2.2 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 PPP over SONET/SDH, and in IETF RFC 1662 for PPP in HDLC-like framing, except that:

a) Link Control Protocol (LCP) is not used to negotiate a link. Rather, a set of static link parameters are 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.

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 and Control Escape bytes are specified in Table 7.3.

Table 7.3—Flag and Control Escape sequences for HDLC-like framing

Byte sequence	Value
Flag Sequence	$7E_{16}$
Control Escape	$7D_{16}$

A diagram of the HDLC-like framing structure is shown in Figure 7.2. 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;

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 7.2—RPR in HDLC-like framing structure

7.4.3 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 7.4 illustrates some of the supported speeds.

Table 7.4—SONET/SDH supported path layers and speeds

SONET path sublayer	SDH path sublayer	Bit rate
STS-3c-SPE	VC4	150 Mb/s
STS-12c-SPE	VC4-4c	600 Mb/s
STS-48c-SPE	VC4-16c	2.4 Gb/s
STS-192c-SPE	VC4-64c	9.6 Gb/s
STS-1c-Mv SPE[a]	VC3-Mv[a]	Mx51Mb/s[a]
STS-3c-Nv SPE[b]	VC4-Nv[b]	Nx150Mb/s[b]

[a]All of the integer values of M are supported from M=3 to M=192.
[b]All of the integer values of N between 2 and 192 are supported. Virtual concatenated paths can support intermediate speeds from 300 Mbps to 9.6 Gbps.

8. Frame formats

Editors' Notes: To be removed prior to final publication.

References:
None

Definitions:
None

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for WG review.
Draft 0.2, April 2002	Draft 0.2 for TF review, modified according to comments on D0.1.
Draft 0.3, June 2002	Draft 0.3 for WG review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for WG review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

8.1 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.

Four frame formats are specified in this clause:

a) Data frame
b) Control frame (other than fairness or idle)
c) Fairness frame
d) Idle frame

This section describes the frame formats used by RPR. The frame format does not include any layer 1 frame delineation.

8.2 Ring data frame format

RPR data messages are identified by the 2-bit *frameType* field (described in 8.2.2.3) being set to 11_2. The minimum data frame size is 20 bytes (the sum of the sizes of the header and trailer and a null PDU). For rings supporting jumbo frames, the maximum transfer length (MTU) is 9216 bytes. For rings not supporting jumbo frames, the MTU is the size of the maximum Ethernet payload plus the size of the RPR header plus the size of the RPR trailer. (Negotiation of jumbo frame capability is described in .) The frame format for data frames is shown in Figure 8.1.

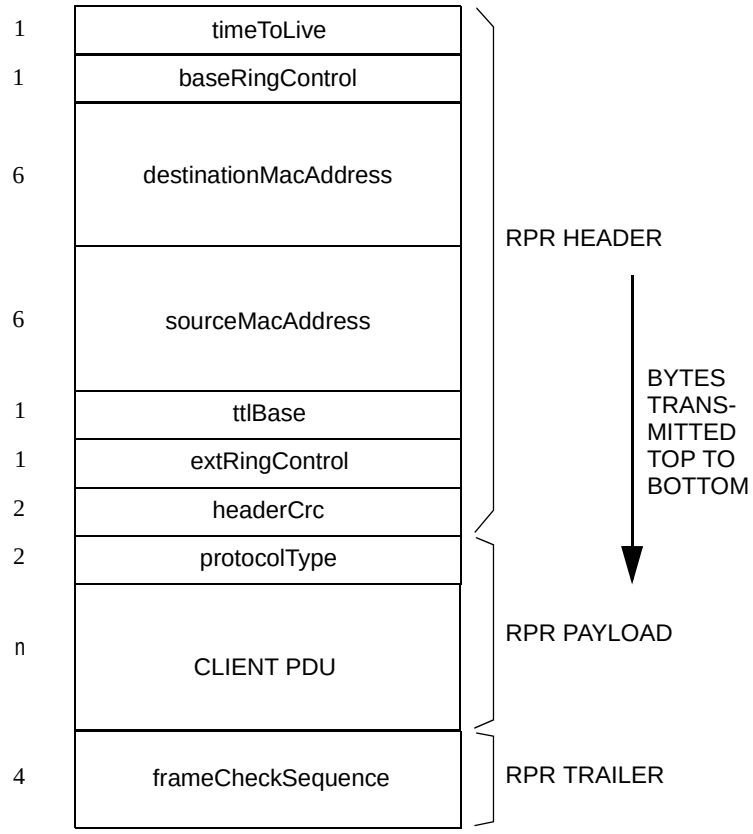


Figure 8.1—RPR data frame format

8.2.1 timeToLive (TTL)

This 8-bit field is a hop count that specifies the maximum number of hops the frame is expected to cover before reaching the destination. It is used to provide a mechanism to ensure that frames do not circulate forever on the ring. The initial value may be exactly the number of known hops to the destination, or a larger number, as described in 6.3.4.7 and 6.3.4.8.

8.2.2 baseRingControl

Each frame has options controlled by the 1-byte *baseRingControl* field. The frame options are shown in Figure 8.2

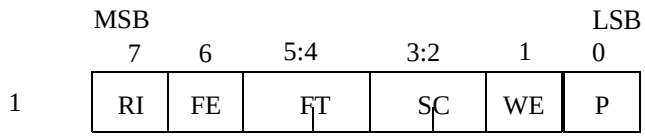


Figure 8.2—baseRingControl field format

The subfields of the *baseRingControl* field are described in the subclauses below.

8.2.2.1 ringletID (RI)

This bit indicates the ringlet onto which the frame was originally transmitted. The following values are defined in Table 8.1

Table 8.1—ringletID values

Value	Name	Description
0	RINGLET_0	ringlet0
1	RINGLET_1	ringlet1

The usage of the RI field is described in 6.3.

8.2.2.2 fairnessEligible (FE)

This bit is used to mark whether the frame is subject to the RPR fairness algorithm. The receiver shall ignore the *FE* field for frames with a value of CLASSA in the *SC* field and for frames with a value of IDLE_FRAME or FAIRNESS_FRAME in the *FT* field. The following values are defined in Table 8.2

Table 8.2—fairnessEligible values

Value	Description
0	Not subject to fairness
1	Eligible for fairness

The usage of the FE field is described in 6.4.

8.2.2.3 frameType (FT)

This 2-bit field identifies the type of the frame. The frame types are defined in Table 8.3:.

Table 8.3—frameType values

Value	Name	Description
00_2	IDLE	Idle frame
01_2	CONTROL	Control frame
10_2	FAIRNESS	Fairness frame
11_2	DATA	Data frame

The values of other fields within the *ringControl* field that have no meaning when combined with a given *frameType* value are ignored.

8.2.2.4 serviceClass (SC)

This 2-bit field is used to identify the service class of the frame. The following classes are defined in Table 8.4

Table 8.4—serviceClass 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 values of other fields within the *ringControl* field that have no meaning when combined with a given *serviceClass* value are ignored. The uses of each of these service classes is defined in 6.4.

8.2.2.5 wrapEligible (WE)

This bit indicates that the frame is eligible to be wrapped during a wrap condition. The following values are defined in Table 8.5

Table 8.5—wrapEligible values

Value	Name	Description
0	STEERABLE	Steerable only
1	WRAPABLE	Wrap eligible

The usage of the WE field is described in Clause 6

8.2.2.6 parity (P)

The *parity* bit provides a parity check of the *ringControl* field. For fairness and idle frames, the *parity* bit shall be set such that the total number of 1 bits in the *ringControl* field, including the parity bit, is odd. For data frames and control frames, it is reserved for future use and shall be set to 0 by implementations meeting this standard, and ignored by any receivers.

8.2.3 destinationMacAddress

The destination address field specifies the station(s) for which the frame is intended. It may be an individual or group address. The destination address field contains a 48-bit address as defined in 5.2 of IEEE Std 802-1990.

8.2.4 sourceMacAddress

The source address field specifies the station sending the frame. The source address contains an individual 48-bit address as defined in 5.2 of IEEE Std 802-1990.

8.2.5 ttlBase

The *ttlBase* is a 1 byte field. It is set to the initial value of the *timeToLive* upon transmission of a data frame.

8.2.6 extRingControl

All data frames have additional options controlled by the 1-byte *extRingControl* field. The frame options are shown in Figure 8.2

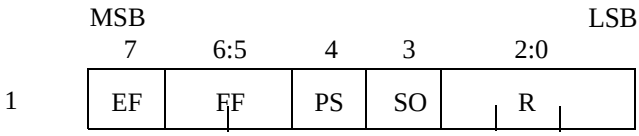


Figure 8.3—extRingControl field format

The subfields of the *extRingControl* field are described in the subclauses below.

8.2.6.1 extendedFrame (EF)

The *extendedFrame* (EF) is a 1-bit field. It is set to 1 to indicate that a client provided frame is contained after the HEC field. Bridging clients will typically set this bit to 1 when relaying frames with a remote *sourceMACAddress* and/or *destinationMACAddress*. See Figure 8.7 for an illustration.

The usage of the EF field is described in 6.?.

8.2.6.2 floodingForm (FF)

The *floodingForm* (FF) is a 2-bit field. It indicates whether the frame is flooded, and if so, by which means. The values of this field are shown in Table 8.6

Table 8.6—floodingForm values

Value	Name	Description
00_2	NO_FLOOD	no flood
01_2	UNI_FLOOD	unidirectional flood
10_2	BI_FLOOD	bidirectional flood
11_2	—	reserved

The usage of the FF field is described in 6.?.

8.2.6.3 pastSource (PS)

The *pastSource* (PS) is a 1-bit field. It is used by wrapping systems (along with other fields) to prevent frame misorder and duplication. It is set to 0 when a frame is first transmitted by a station and set to 1 when a wrapped frame (i.e., frame traveling on the secondary ringlet) passes the source station.

The usage of the PS field is described in 6.?.

8.2.6.4 strictOrder (SO)

The *strictOrder* (SO) is a 1-bit field. It indicates whether the frame should conform to relaxed or strict ordering requirements. A value of 0 indicates relaxed, while a value of 1 indicates strict.

The usage of the SO field is described in 6.?.

8.2.6.5 reserved (R)

The *reserved* field is a 3-bit field. It is reserved for future use and shall be set to all zeros by implementations meeting this standard, and ignored by any receivers.

8.2.7 headerCrc (HEC)

The *headerCrc* (HEC) is a 16 bit checksum. The generator polynomial is:

$$\text{CRC-16} = x^{16} + x^{12} + x^5 + 1$$

The checksum is computed over the *timeToLive*, *baseRingControl*, *destinationMacAddress*, *sourceMacAddress*, *ttlBase*, and *extRingControl*, with the bits of the frame presented to the CRC generator in the same order as is done for Ethernet. The initial value for the HEC CRC calculation is an all-zeros value. Single-bit error correction by the receiver is optional. Annex F provides more information on the *headerCrc* calculation.

8.2.8 protocolType

Contained within the first two bytes of the RPR payload is the *protocolType* field. When the value of this field is greater than or equal to 1536 decimal (equal to 0600 hexadecimal) the *protocolType* field indicates the nature of the MAC client protocol (Type interpretation). When less than 1536, the field is interpreted as the length of the frame. The length and type interpretations of this field are mutually exclusive. For data frames, the *protocolType* field, when used with the type interpretation, shall indicate the type of data frame, selecting from values designated by the IEEE Type Field Register.

8.2.9 serviceDataUnit (SDU)

The primary contents of the RPR payload is a variable length field containing the service data unit (SDU) provided by the client.

8.2.10 frameCheckSequence (FCS)

The *frameCheckSequence* (FCS) is a 32-bit cyclic redundancy check (CRC) as used in IEEE Std 802.3-2000 CSMA/CD. The generator polynomial is:

$$\text{CRC-32} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

The FCS CRC is calculated starting from the byte following the headerCrc to the end of frame, with the bits of the frame presented to the CRC generator in the same order as is done for Ethernet. The initial value for the FCS CRC calculation is an all-ones value. Annex F provides more information on the FCS CRC calculation.

8.2.11 Data frame format usage

8.2.11.1 Local unicast forwarding

Local unicast forwarding refers to frame transmissions on the ring where the local source station is dispatching a local unicast packet. Local transfers involve insertion of *timeToLive*, *baseRingControl*, *ttlBase*, *extRingControl*, and *headerCRC* information, as illustrated in Figure 8.4, to ensure reliable RPR local delivery.

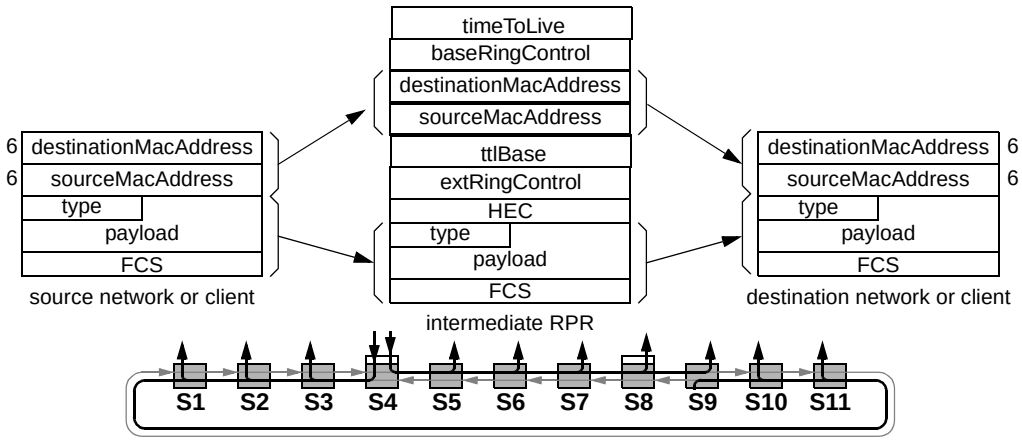


Figure 8.4—Local frame forwarding

8.2.11.2 Local multicast, broadcast, and unknown unicast forwarding

Local multicast forwarding refers to frame transmissions on the ring originated by a local station (e.g. host or router) but dispatched to a multicast group. Local broadcast forwarding refers to frame transmission on the ring originated by a local station (e.g. host or router) but dispatched to a broadcast group. Local unknown unicast forwarding refers to frame transmission on the ring originated by a local station (e.g. host or router), where the client provided *destinationMACAddress* is not local to the ring. Local unknown unicast forwarding transmissions are flooded on the ring.

All of the above transmission types are supported by a single frame structure as illustrated in Figure 8.5. This is the same frame structure that is used for local unicast forwarding, as illustrated in Figure 8.4. Transfers involve insertion of *timeToLive*, *baseRingControl*, *ttlBase*, *extRingControl*, and *headerCRC* information to ensure reliable RPR local delivery.

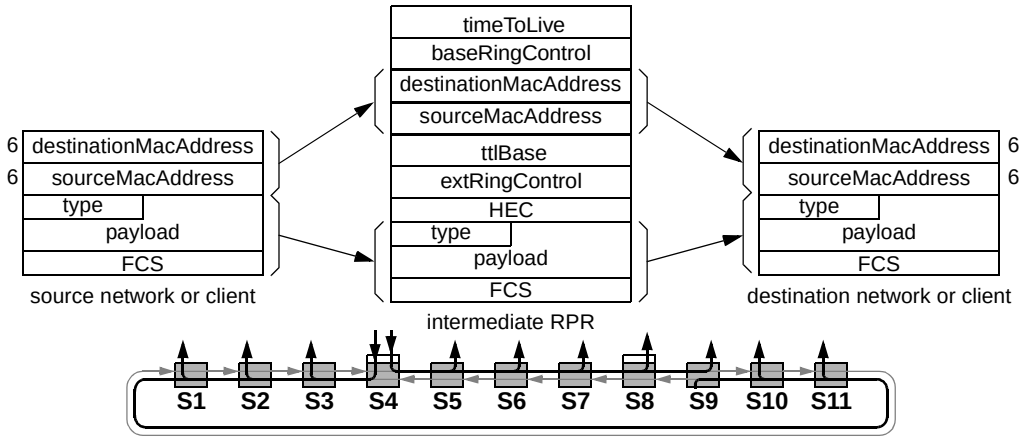


Figure 8.5—Local multicast, broadcast, unknown unicast forwarding

8.2.11.3 Remote forwarding

Remote transfers refer to traffic that is originated by a remote device (i.e., an off ring device) and/or terminate to a remote device. Either the *sourceMACAddress* and/or *destinationMACAddress* of the client frame is a remote MAC address. Remotely-sourced transfers involve insertion of *timeToLive*, *baseRingControl*, *ttlBase*, *extRingControl*, and *headerCRC* information, along with setting the 48-bit *sourceStationID* components to ensure reliable RPR local delivery. The *extendedFrame* bit is set to 1 for these transmissions.

Figure 8.6 illustrates a remote transfer that is broadcast over the ring.

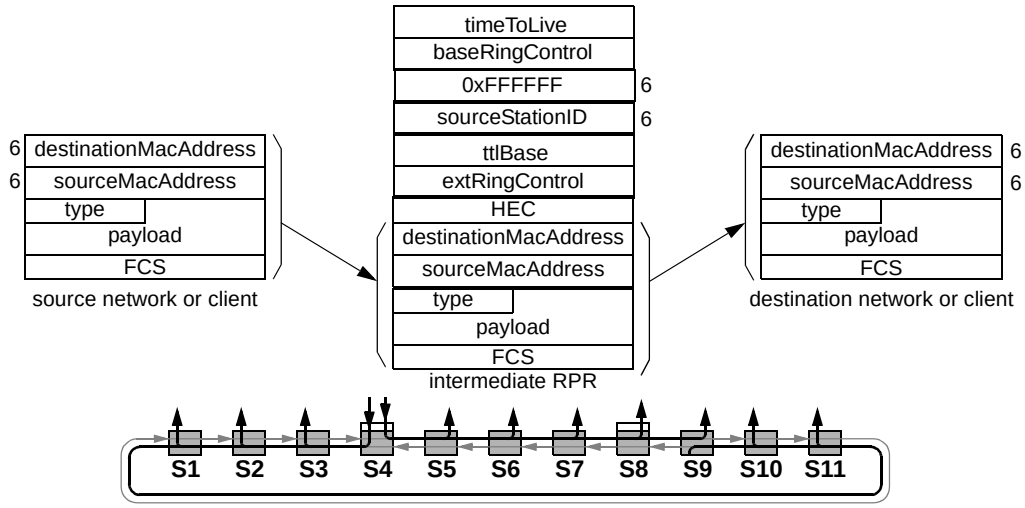


Figure 8.6—Example 1: Remote frame forwarding

Figure 8.7 illustrates a remote transfer that will terminate on the ring. This is required to support clients such as enhanced bridging.

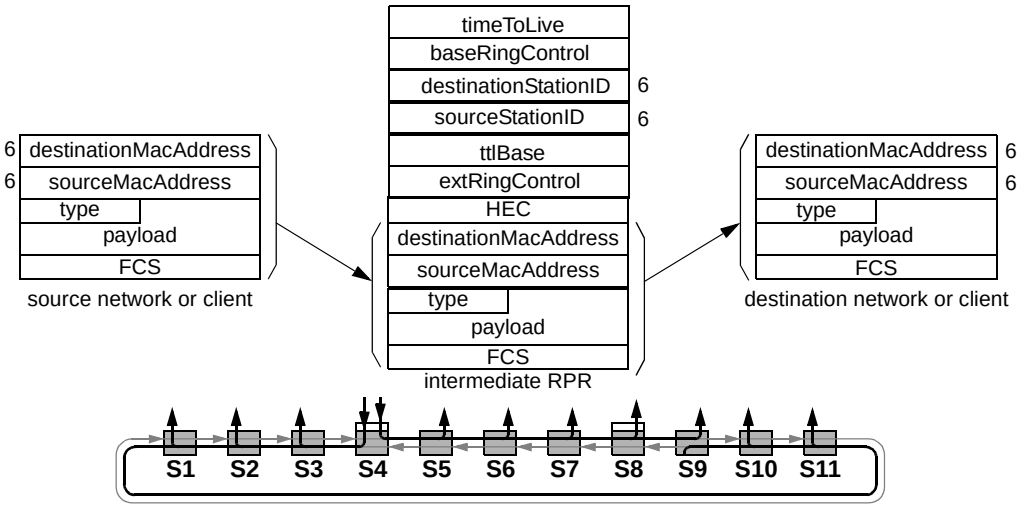


Figure 8.7—Example 2: Remote frame forwarding

8.3 RPR control frame format

RPR control messages are identified by the 2-bit *frameType* field (described in 8.2.2.3) being set to 01_2. The minimum control frame size is 24 bytes (the sum of the sizes of the header and trailer and a 2-byte PDU). For rings supporting jumbo frames, the maximum transfer length (MTU) is 9216 bytes. For rings not supporting jumbo frames, the MTU is the size of the maximum Ethernet payload plus the size of the RPR header plus the size of the RPR trailer. (Negotiation of jumbo frame capability is described in .) An RPR control frame could be a broadcast or unicast message. The serviceClass (SC) value for control frames shall be set to indicate classA, subclassA0, except in the case of some OAM control frames which may be set to

any SC value as indicated by the control request primitive. Figure 8.8 shows the RPR frame format for control frames.

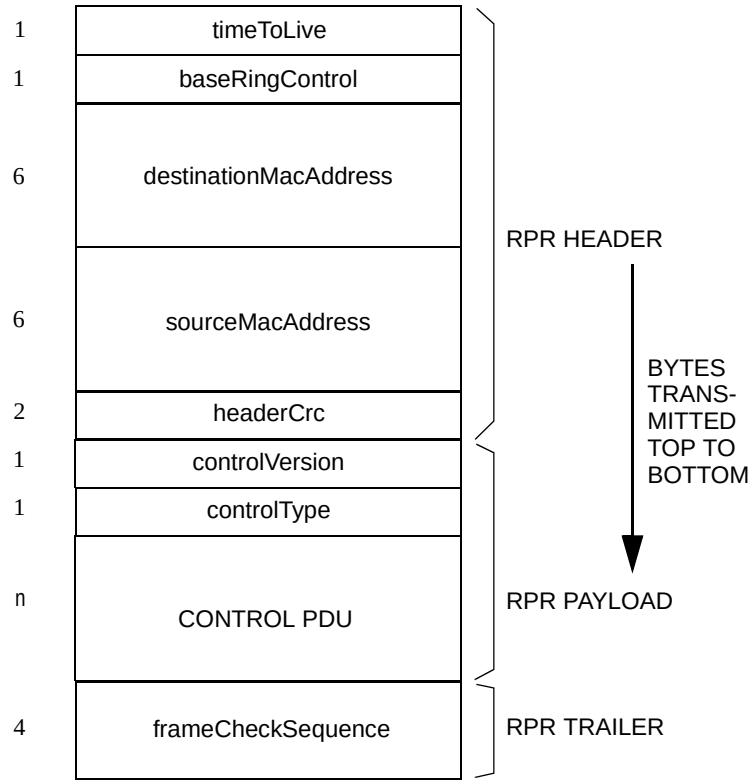


Figure 8.8—Control frame format

8.3.1 timeToLive (TTL)

The *timeToLive* (TTL) field in a control frame is interpreted the same as described in 8.2.1. The initial value is set as specified in each of the clauses describing control frames.

8.3.2 baseRingControl

The *baseRingControl* field in a control frame is interpreted the same as described in 8.2.2.

8.3.3 destinationMacAddress

The *destinationMacAddress* field in a control frame is interpreted the same as described in 8.2.3.

8.3.4 sourceMacAddress

The *sourceMacAddress* field in a control frame is interpreted the same as described in 8.2.4.

8.3.5 headerCrc (HEC)

The *headerCrc* (HEC) in a control frame is interpreted the same as described in 8.2.7, with the exception that the checksum is computed over the *timeToLive*, *baseRingControl*, *destinationMacAddress*, and *sourceMacAddress* fields.

8.3.6 controlVersion

The 1-byte *controlVersion* field is the version number associated with the *controlType* field. This field is used to provide a means of identifying future versions of the control frames. Initially, all control types will be version 0. The handling of frames with other values is not specified by this standard.

8.3.7 controlType

The 1-byte *controlType* field indicates the type of control frame, selecting from values designated by this standard. Table 8.7 contains the currently defined control types.

Table 8.7—controlType values

controlType	Name	Description
1	TOPOLOGY	Topology Discovery
2	PROTECTION	Protection message
3	OAM	OAM control frame
all others	—	Reserved

8.3.8 Control PDU

The control PDU is a variable length field dependent on the value of the controlType field. This field is specified in each of the clauses describing control frames.

8.3.9 frameCheckSequence (FCS)

The *frameCheckSequence* (FCS) in a control frame is interpreted the same as described in 8.2.10.

8.4 RPR fairness frame format

RPR fairness messages are identified by the 2-bit *frameType* field (described in 8.2.2.3) being set to 10_2. The size of a fairness frame size is 16 bytes. An RPR Fairness Frame, as shown in Figure 8.9, is sent to MAC neighbors to convey data for the fairness algorithm specified in Clause 9.

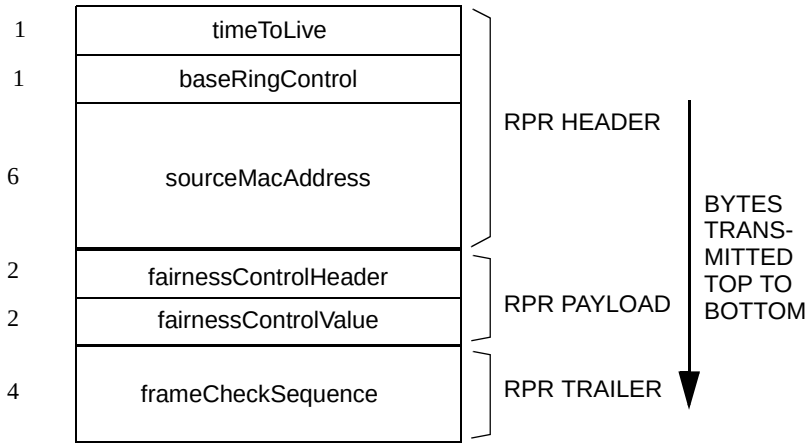


Figure 8.9—RPR fairness frame format

NOTE—The frame format for fairness frames is different from the frame format for data frames and other control frames. Fairness messages are not sent to specific destination nodes, but 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 messages are kept as small as possible to minimize bandwidth consumed by sending fairness packets and to minimize memory requirements for storing multiple fairness messages (especially when multi-choke fairness algorithms are used).

8.4.1 timeToLive (TTL)

The *timeToLive* (TTL) field in a fairness frame is interpreted the same as described in 8.2.1. The initial value is set as specified in Clause 9.

8.4.2 baseRingControl

The *baseRingControl* field in a fairness frame is interpreted the same as described in 8.2.2, with the noted difference of using the parity (P) bit. Since fairness frames do not contain a *headerCrc* field for header protection, the *parity* bit is used to protect the *baseRingControl* field, instead of being reserved.

8.4.3 sourceMacAddress

The *sourceMacAddress* field in a control frame is interpreted the same as described in 8.2.4.

8.4.4 fairnessControlHeader

This two byte field is specified in 9.7.4.

8.4.5 fairnessControlValue

This two byte field is specified in 9.7.4.

8.4.6 frameCheckSequence (FCS)

The *frameCheckSequence* (FCS) is a 32-bit cyclic redundancy check (CRC) as used in IEEE Std 802.3-2000 CSMA/CD. The generator polynomial is:

$$CRC\text{-}32 = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

The FCS CRC is calculated starting from the byte following the ringControl to the end of frame, with the bits of the frame presented to the CRC generator in the same order as is done for Ethernet. The initial value for the FCS CRC calculation is an all-ones value. Annex F provides more information on the FCS CRC calculation.

8.5 RPR idle frame format

RPR idle frames are identified by the 2-bit *frameType* field (described in 8.2.2.3) being set to 00_2. The size of an idle frame size is 16 bytes. An RPR idle frame, as shown in Figure 8.10, is sent to MAC neighbors to adjust the rate synchronization between the station and its neighbors as specified in 6.9.4.

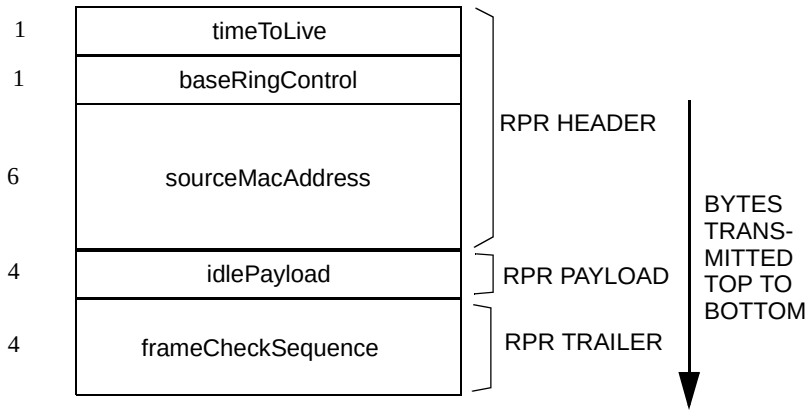


Figure 8.10—RPR idle frame format

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 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 minimize jitter on other frames.

8.5.1 timeToLive (TTL)

The *timeToLive* (TTL) field in an idle frame is interpreted the same as described in 8.2.1. The initial value is set as specified in Clause 6.

8.5.2 baseRingControl

The *baseRingControl* field in an idle frame is interpreted the same as described in 8.2.2, with the noted difference of using the parity (P) bit. Since idle frames do not contain a *headerCrc* field for header protection, the *parity* bit is used to protect the *baseRingControl* field, instead of being reserved.

8.5.3 sourceMacAddress

The *sourceMacAddress* field in a control frame is interpreted the same as described in 8.2.4.

8.5.4 idlePayload

This 4-byte field is reserved for future use and shall be set to all zeros by implementations meeting this standard and ignored on receipt.

8.5.5 frameCheckSequence (FCS)

The *frameCheckSequence* (FCS) is a 32-bit cyclic redundancy check (CRC) as used in IEEE Std 802.3-2000 CSMA/CD. The generator polynomial is:

$$\text{CRC-32} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

The FCS CRC is calculated starting from the byte following the ringControl to the end of frame, with the bits of the frame presented to the CRC generator in the same order as is done for Ethernet. The initial value for the FCS CRC calculation is an all-ones value. Annex F provides more information on the FCS CRC calculation.

8.6 Invalid RPR frame

An invalid RPR frame is defined as one that meets at least one of the following conditions

a) Data and control frames where the *headerCrc* does not match with the frame as received.
b) Fairness or idle frames where the *parity* (P) bit gives the incorrect parity for the *baseRingControl* field.
c) Any frame where the FCS does not match with the frame as received. (This is optional for data frames).

The MAC shall discard invalid RPR frames except as noted below.

The contents of RPR frames with FCS mismatches may, optionally, be passed to certain MAC clients when the receiving station is configured to accept such frames. The presence or lack of an FCS error is indicated in the receptionStatus parameter of the MA_DATA.indication primitive, as shown in Table 5.1.

The occurrence of invalid MAC frames shall be communicated to network management.

9. Fairness

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	Revised according to comments on D0.1 for TF review.
Draft 0.3, June 2002	Revised according to comments on D0.2 for WG review.
Draft 1.0, August 2002	Revised according to comments on D0.3 for TF review.
Draft 1.1, October 2002	Revised according to comments on D1.0 for WG review.
Draft 1.2, December 2002	Revised according to comments on D1.1 for TF review.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

Editors' Notes: *To be removed prior to final publication.*

The resolution to comment #174 11/02 required that material associated with MC-FCM to be moved to a separate section. This change has been deferred to the next version.

9.1 Overview

9.1.1 Scope

This clause defines fairness control functions. Key among these functions is the fairness algorithm (FA). The FA is a distributed computation of the maximum rate at which fairness-eligible (FE) traffic can be added to the ringlet by a station. FE traffic includes classC traffic and that portion of classB traffic in excess of the committed rate. The fairness control function is performed by the fairness control unit (FCU), which resides in the MAC control sublayer, as illustrated by the shaded region of Figure 9.1.

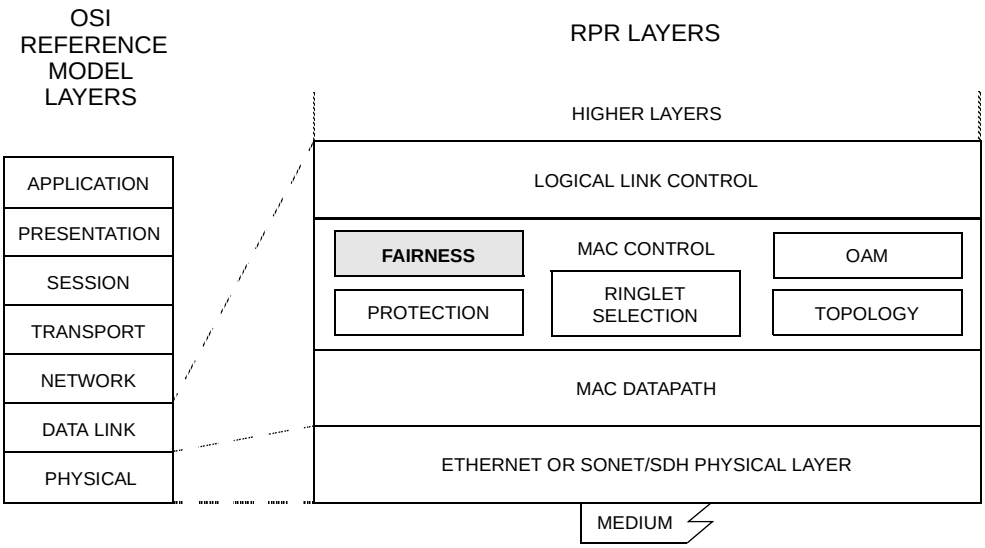


Figure 9.1—Position of the fairness control unit in the layer model

9.1.2 Goals and objectives

The fairness protocol has the following objectives:

Editors' Notes: *To be removed prior to final publication.*

Comments #123 - 128 11/02 will be addressed by edit of this section. It is also proposed that this information be provided as 'services and features' in the 'scope' section, as is the case with Clauses 10, 11, and 12.

The qualifier 'source-based' requires the explanation that this is in contrast to 'per-flow' or 'per source-destination pair' fairness.

a) Source-based weighted fairness—On any given link on the ringlet, the available bandwidth is allocated to each station in proportion to its relative weight. For example, if every station has an equal weight, then the available bandwidth on the link should be shared equally by all stations. On the other hand, if one station has a higher weight, the bandwidth allocated to that station should be in proportion to the station's weight divided by the sum of the weights of all the stations.

b) **Reclamation of unused committed bandwidth**—The fairness protocol should be able to reclaim unused bandwidth, including that which is allocated but reclaimable (i.e. subclassA1 and classB CIR bandwidth).

c) **Support for single-choke and multi-choke capable clients**—The fairness protocol should be able to support clients whether they utilize a single choke-point or multiple choke-points (such as those clients performing virtual destination queueing (VDQ)).

d) **Fast response time**—Because data traffic tends to be bursty, in order to ensure maximum ring bandwidth utilization and to ensure that the protocol is responsive to instantaneous changes in traffic load, it must have a fast response time.

e) **High bandwidth utilization on the ring**—The protocol should be able to achieve very high levels of bandwidth utilization even under heavy loads approaching 100% of the ring capacity.

f) **Scalability**—The protocol should be scalable and should be able to function predictably for all ringlet speeds and ring diameters allowed by this standard.

g) **Stability**—The fairness protocol should enter a steady state within a finite time when presented with a steady input traffic pattern.

9.1.3 Relationship to other clauses

Fairness control shares the following global variables with Clause 6: *addRate*, *addRateCongested*, *fwRate*, *fwRateCongested*, *nrXmitRate*, *allowedRate*, *allowedRateCongested*, and *hopsToCongestion*.

Fairness control uses the ScFcmInd and McFcmInd opcodes of the MA_control.ind primitive described in Table 5.3 of Clause 5 to optionally communicate rate and congestion information from fairness control to the MAC client.

Fairness control uses the MLME-GET.request primitive, described in Clause 13, to obtain the value of a MIB attribute that has been set by other MAC control functions.

9.2 Variables and terminology

Editors' Notes: *To be removed prior to final publication.*

These definitions, and possibly their order of introduction, will be modified consistent with comment #412 11/02 in order to provide greater clarity.

Variables and constants described in this section will be moved to the state machine description, consistent with the new (11/02) state machine template.

This clause contains the following definitions and formula variables:

9.2.1 *activeStations*: (informative) Calculated. The number of stations detected as having sourced fairness-eligible-frames on the ringlet during the last *agingInterval* as detected on the outgoing link of the local station. The local station is included in the count of *activeStations* if (a) if has sourced such a frame or (b) it is congested.

NOTE—Active-stations is used as an initial value in the '*conservative*' mode fairness computation. The value affects convergence time, but not the steady-state computed fair-rates. Detection of active stations is optional. If the number of *activeStations* is not measured, a value between (a) *unreservedRate* / (numStations * WEIGHT) and (b) *unreservedRate*/WEIGHT can be used as an initial estimate of the *localFairRate*.

9.2.2 *activeWeights*: (informative) Calculated. The sum of the WEIGHT values associated with *activeStations*.

9.2.3 *addRate:* Global variable (see Clause 3).

9.2.4 *addRateCongested*: Global variable (see Clause 3).

9.2.5 *advertisedFairRate*: Calculated. The value placed in the controlValue field of a fairness control message sent by the local station. The value is either *normLocalFairRate*, *rcvdFairRate*, or FULL_RATE indicating the absence of congestion on the ringlet.

9.2.6 *advertisementInterval*: Calculated. The interval at which fairness control messages are sent. The *advertisementInterval* is computed as (sizeof(FCM)) / (LINK_RATE**advertisementRatio*).

Editors' Notes: *To be removed prior to final publication.*

The note below will be changed to reflect that it is the advertisementRatio, not the advertisementInterval, that has a default value.

NOTE— Default values of *advertisementInterval* and *agingInterval* were selected to be (a) large enough so that (1) the bandwidth overhead required for control is small (<<1%), (2) processing requirements are not excessive with respect to current technology, and (3) queuing requirements are not excessive with respect to current technology and (b) small enough to allow the fairness algorithm to converge. The *agingInterval* must be the same across all stations on the ring. The *advertisementInterval* may vary from station to station, consistent with the definition provided.

9.2.7 *advertisementRatio*: Configured. The ratio of (a) link capacity reserved for fairness control messages to (b) total link capacity. Allowed values are in the range [0.01, 0.00025]. Default is 0.00125.

9.2.8 AGECOEF: Configured. A coefficient used by the aging function specify the relative weights assigned to (a) the change in the value of a variable during the most recent *agingInterval* and (b) the value of the variable at the end of the previous aging interval. The default value is 4. Allowed values are {1, 2, 4, 8, 16}.

9.2.9 *aggressive*: Configured. Mode of operation of the fairness algorithm in which the *localFairRate* is computed based on the *addRate* of the congested station. This allows relatively high utilization of capacity at the expense of wider fluctuation in *localFairRate* values. See section 9.4.2 for further description of operational modes.

9.2.10 aging: A method of adjusting a counter so as to (a) smooth successive values of the counter and (b) bound the maximum value of the counter in order to prevent overflow. See section 9.4.3 for further description of aging.

9.2.11 *agingInterval*: Configured. The interval at which aging and low-pass filtering functions are performed. The value of the *agingInterval* is specified as a function of link data rate by Table 9.1.

Table 9.1—agingInterval as a function of link rate

Link Speed	agingInterval
OC-12 and higher rates	100 usec
Rates less than OC-12	400 usec

9.2.12 *allowedRate*: Global variable (see Clause 3).

9.2.13 *allowedRateCongested*: Global variable (see Clause 3).

9.2.14 *congestionPoint*: Calculated. A station identified by a local station as the most congested station from which the local station receives SC-FCMs. The degree of congestion is inferred from the value of *rcvdAdvertisedRate* extracted from the *controlValue* field of the FCM.

9.2.15 *conservative*: Configured. Mode of operation of the fairness algorithm in which the *localFairRate* is computed based on the previous *fairRate* value and the effect of previous *fairRate* advertised on the conges-

tion state. This allows relatively narrow fluctuation in *fairRate* values at the expense of capacity utilization. See section 9.4.2 for further description of operational modes.

9.2.16 *controlValue*: Calculated: Field in the fairness control message that carries the *advertisedRate* sent by a station and from which the *rcvdAdvertisedRate* is extracted.

9.2.17 *downstreamCongested*: Calculated: A boolean value indicating whether the local station has information indicating that a station downstream on the ringlet is congested.

9.2.18 *dualQueue*: Configured: Boolean: Indicates whether or not station deploys a dual-queue MAC. If false, the station deploys a single-queue MAC.

9.2.19 FULL_RATE: Constant. A special value of *controlValue*, indicating the absence of congestion. The value is $FFFF_{16}$

9.2.20 *fwRate*: Global variable (see Clause 3).

9.2.21 *fwRateCongested*: Global variable (see Clause 3).

9.2.22 *hopsToCongestion*: Global variable (see Clause 3).

9.2.23 LINK_RATE: Constant. The number of bytes that can be sent during AGECOEF *agingIntervals*. A value of LINK_RATE is associated with each PHY type and is specified in the annex describing the reconciliation sublayer associated with that PHY.

9.2.24 *localCongested*: Calculated: A boolean value indicating whether or not the local station considers itself to be congested.

9.2.25 *localFairRate*: Calculated. The rate at which the station is allowed to add client-supplied FE marked traffic to the ringlet in the absence of downstream congestion, specified as the number of bytes that may be added per AGECOEF *agingIntervals*.

9.2.26 low-pass filtering: A method of smoothing that removes rapid oscillations from a sequence of observed values. See 9.4.4 for a description of the low-pass filter function.

9.2.27 LPCOEF: Configured. A constant used by the low-pass filter function to specify the relative weights applied to (a) the observed value during the most recent filterInterval and (b) the previous low-pass filtered value. The former is assigned a weight of 1 and the latter a weight of (LPCOEF-1). The default value for this constant is 64. Allowed values are {16, 32, 64, 128, 256, 512}.

9.2.28 *lpAddRate*: Calculated. A smoothed version of *addRate* obtained by applying the low-pass filter function to that variable.

9.2.29 *lpAddRateCongested*: Calculated. A smoothed version of *addRateCongested* obtained by applying the low-pass filter function to that variable.

9.2.30 *lpFwRate*: Calculated. A smoothed version of *fwRate* obtained by applying the low-pass filter function to that variable.

9.2.31 *lpFwRateCongested*: Calculated. A smoothed version of *fwRateCongested* obtained by applying the low-pass filter function to that variable.

9.2.32 *lpNrXmitRate*: Calculated. A smoothed version of *nrXmitRate* obtained by applying the low-pass filter function to that variable.

9.2.33 MAX_ALLOWED_RATE: Configured. The maximum value for *allowedRate*. The default value for is LINK_RATE.

> ***Editors' Notes:*** *To be removed prior to final publication.*
>
> *'Multi-choke' describes anything associated with the existence of multiple congestionPoints on a ringlet. The only activity performed by the MAC, involving multiple congestionPoints is the broadcast of fairRate values on the ringlet and the transfer of that information via an indication (McFcmInd) to the client. The MAC has no awareness of a multi-choke fairness algorithm. Suggest something like this:*
> ***multi-choke:*** *Definition. Associated with multiple congestionPoints on a ringlet.*

9.2.34 multi-choke: Definition. The ability to work with multi-choke points (of different values) at the same time, e.g. as in a VDQ implementation. The primary intended use of the multi-choke fairness control message (MC-FCM).

9.2.35 *normalize*: To adjust a local value to a value that allows a standard interpretation when communicated from one station to another. The normalized value is equal to the local value divided by the constant NORMCOEF.

9.2.36 NORMCOEF: Calculated. The a value used to adjust a local rate value to a normalized rate value. The normalized rate is equal to the local rate divided by the NORMCOEF. The value of NORMCOEF is the product of AGECOEF, RATECOEF, and WEIGHT.

9.2.37 *normLocalFairRate*: Calculated. The normalized value of *localFairRate*.

9.2.38 *normLpFwRateCongested*: Calculated. The normalized value of *lpFwRateCongested*.

9.2.39 nrXmitRate: Global variable (see Clause 3).

9.2.40 *numStations*: Calculated. The number of stations known by the local station to be on the ringlet. Obtained via rprIfNodesOnRing.

9.2.41 RAMPCOEF: Configured. The coefficient used for ramping-up the *allowedRateCongested* or *localFairRate*. The default value for this constant is 64. Allowed values are {16, 32, 64, 128, 256, 512}.

> ***Editors' Notes:*** *To be removed prior to final publication.*
>
> *Definition of ramping-up to be added.*

9.2.42 RATECOEF: Configured. The coefficient used for normalizing advertised fair rates to a common link rate. The RATECOEF is used to ensure that the control value does not overflow 16-bits. The value of RATECOEF is equal to 1 for link data rates less than or equal to 2.5Gbps. and is equal to the link data rate divided by 2.5Gbps., rounded to an integer power of 2, for link data rates greater than 2.5Gbps.

9.2.43 *rateFullThreshold*: Configured. A level of data rate in a dual-queue deployment, indicating that the station is transiting traffic at a rate that is close to the maximum available rate. The value of *rateFullThreshold* is set to the *unreservedRate*.

9.2.44 *rateHighThreshold*: Constant. Rate at or above which congestion on the outbound link is declared. Default value is 0.95**rateFullThreshold*. Range is [0.4**rateFullThreshold*, 0.99**rateFullThreshold*].

9.2.45 rateLowThreshold: Constant. Rate at or above which congestion on the outbound link is imminent. Default value is 0.9**rateHighThreshold* and range[0.5**rateHighThreshold*, 0.99**rateHighThreshold*].

9.2.46 *rcvdFairRate*: Calculated. The normalized rate (in bytes) received in an FCM, giving the fair number of bytes that can be sent during the next *agingInterval*.

9.2.47 *reservedRate*: Calculated. The total classA traffic that is reserved (i.e. the total of all subclassA0 traffic) on the outgoing link for all stations, including the local station, specified as the number of bytes per AGECOEF *agingIntervals*.

9.2.48 single-choke: Definition. The ability to work with only one choke-point at a time. The primary intended use of the single-choke fairness control message (SC-FCM).

9.2.49 *STQFullThreshold*: Configured. A level of buffer occupancy in a single-queue deployment, indicating that the STQ is almost full. The default value for STQF*ullThreshold* is sizeof(STQ) - sizeof(MTU). The range is [*STQHighThreshold* - 1MTU, sizeOf(STQ) - 1MTU].

9.2.50 *STQHighThreshold:* Constant. Secondary transit queue occupancy at or above which congestion on the outbound link is declared. Defined only for dual transit-queue implementations. Default value is 0.25**STQSize*. Range is [2MTU, *STQFullThreshold* - MTU].

9.2.51 *STQLowThreshold:* Constant. Secondary transit queue occupancy at or above which congestion on the outbound link is imminent. Defined only for dual transit-queue implementations. Default value is 0.125**STQSize*. Range is [1MTU, *STQHighThreshold* - 1MTU].

9.2.52 *unreservedRate*: Calculated. The rate available for the outbound link that is not reserved, specified as the number of bytes per AGECOEF *agingIntervals*.

9.2.53 WEIGHT: Configured. The local weight by which *allowedRate*, *allowedRateCongested*, and *localFairRate* are calculated. The default value for this constant is 1. The allowed range is [1, 255].

9.3 Fairness operation

Editors' Notes: *To be removed prior to final publication.*

Provide explanation (or review) of fairness at the start of this section. The functions listed are not self-explanatory (comment #157). Section should be organized with subsections corresponding the listed functions (comment #160). Add distinct sections to describe aging, low-pass filtering, normalizing, etc. Add distinct section describing methods for establishing initial value of localFairRate.

The fairness algorithm implemented within the FCU consists of the following functions:

Editors' Notes: *To be removed prior to final publication.*

It is apparently not clear whether the 'fairness algorithm' includes all fairness control functions or just a subset, and which functions are included.

a) Determining when the congestion threshold is crossed and when the congestion has subsided;
b) Determining the fair rate for advertisement;
c) Determining the station's allowed rate;
d) Sourcing and consuming fairness control messages (FCMs);
e) Communicating the allowed rate to the data path shapers for controlling access to the medium;
f) Providing the information contained in MC-FCMs to the client.

A station is configured with a weight that allows adjustment of the portion of available ring bandwidth assigned to the station by the fairness algorithm. The ability to consider such weights when computing fair

rates, is known as the weighted fairness property of the fairness algorithm. A station learns the weights of other stations on the ring via the weight TLV of the extended status message.

Editors' Notes: *To be removed prior to final publication.*

Provide figure illustrating the direction of flow of fairness messages and flow chart or other diagram illustrating the sequence of actions described in this paragraph (comments #158, 159). The value 2.5 Gbps either requires explanation, should be moved to a later subsection where explanation would be more appropriate, or should be explained by a 'note'.

Clarify statement that normalization of localFairRate yields both normLocalFairRate and the *advertisedFairRate.*

A distinct instance of the FA is associated with each of the two ringlets. Each station advertises a fair rate to upstream stations via the ringlet opposite that associated with the FA instance. The fair rate is run through a low pass filter function, with the result known as the *localFairRate*. The *localFairRate* is normalized by the local station weight (WEIGHT), the aging coefficient (AGECOEF), and the rate coefficient (RATECOEF) to yield the *normLocalFairRate* and the *advertisedFairRate*. The low-pass filter stabilizes the feedback, the division by WEIGHT normalizes the transmitted value to a weight of 1.0, the division by AGECOEF normalizes the transmitted value to one *agingInterval*, and the division by RATECOEF normalizes the transmitted value to a link speed of 2.5 Gbps. When the upstream stations receive an advertised fair rate, they will adjust their transmit rates for fairness-eligible traffic so as not to exceed the advertised value (adjusted by their respective weights).

Propagation of the advertised value to other stations on the ring is done using fairness control messages. The format of the fairness control messages is described in 9.7. There are two types of fairness control messages—single-choke and multi-choke.

Single-choke messages are propagated hop-by-hop around the opposite ringlet and are processed by the FCU. They are sent every *advertisementInterval*.

Editors' Notes: *To be removed prior to final publication.*

The meaning of 'folded ring' is not obvious. The paragraph will be rewritten in accordance with comment #169. Provide figure for illustration (comment #171)

A wrapped ring shall be treated as a folded ring from the point of view of SC-FCMs. A station with a wrapped attachment point receiving a SC-FCM shall wrap the SC-FCM.

Editors' Notes: *To be removed prior to final publication.*

Add figures and/or flow charts to clarify the paragraphs that follow. (Comments #172, 173)

A single-choke fairness control messages contains the SA of the most congested station through the message passes. If a station experiences congestion, it will include a non-FULL_RATE value for the fair rate in its single-choke fairness control messages. A station that receives a single-choke message with a SA of its MAC address shall treat the message as having a *controlValue* of FULL_RATE regardless of the actual *controlValue*.

A station that is in the congested state shall advertise the minimum of its *localFairRate* and the last received fair rate (known as the *rcvdFairRate*) in its single-choke message.

A station that is not congested and that receives a single-choke message containing a non-FULL_RATE *rcvdFairRate* shall either propagate the *rcvdFairRate* to its upstream neighbor (leaving the SA the same), or

it will send a value of FULL_RATE and set the SA to its own. Which of these is sent is determined by the following. If the low pass filtered *fwRateCongested* is less than the *allowedRateCongested* divided by the local station weight, then a FULL_RATE value is propagated to the upstream neighbor instead of the *rcvdFairRate*; in other words, an upstream station is not the cause of congestion and there is no need to propagate the single-choke FCM indicating congestion upstream of this station.

The value of the rate that is allowed for the fairness-eligible traffic (*allowedRateCongested*) that can be added by the station is derived from the received advertised rate in the single-choke messages and the local station normalization factor (NORMCOEF). If the number of hops to the destination of a data packet from the MAC client is less than the number of hops to the station that generated the single-choke message, then the MAC client can take advantage of the available bandwidth on the ringlet; otherwise, if the destination of a data packet would allow the packet to go beyond the station that generated the FCM, the client will at least receive its fair share of the bandwidth from the most congested span that it contends for.

Editors' Notes: *To be removed prior to final publication.*

Cleanly separate multi-choke explanations into a distinct subclause, making it clear that the MC-FCM does not participate in the fairness algorithm. (comment #174)

Multi-choke messages are broadcast on the ringlet opposite to the ringlet upon which the fairness algorithm is running and contain the SA of the station that originated the message. The messages are required and are sent every 10 *advertisementInterval*s. If a station experiences congestion, it will include a non-FULL_RATE value for the fair rate in its multi-choke FCMs; otherwise it will include a FULL_RATE value for the fair rate. Multi-choke messages received by the MAC are not processed by the FCU, but the information is passed to the MAC client by the FCU and is used by the MAC client.

NOTE (informative) —The client can also take advantage of Virtual Destination Queueing (VDQ) by utilizing the multi-choke concept of FA. VDQ combined with FA can increase ring utilization. The multi-choke concept deals with the case where a station wants to send traffic to a destination that is closer than a congested link. As an example, consider the case where station 1 wants to send traffic to station 2, and the link between stations 2 and 3 is congested. FA will allow station 1 to send as much traffic as it wants to station 2, and will only limit traffic to stations beyond the congested link to the fair rate. In a multi-choke implementation of the FA, each client will track advertised fair rates for congested stations. A station is allowed to send unlimited traffic to any station between itself and the first congested station (choke-point). It can send traffic to stations between the first and second choke-point based on the first choke-point's advertised fair rate. In general, a station can send traffic to a particular destination if it has satisfied the fair rate conditions for all choke-points between itself and the destination. The maximum possible number of choke-points is equal to the number of stations on the ring.

9.3.1 Fairness control message receive

For every *agingInterval*, all variables used to determine congestion condition, congestion location, and rate control are recomputed. Those values that depend upon values received in SC-FCMs use the most recently received SC-FCM.

Editors' Notes: *To be removed prior to final publication.*

The flow charts that follow contain state names such as WHO and ME, that do not match later state tables. The tables will be updated to remedy. (Comment # 193)

FCM receive processing is specified in the state machine of Table 9.1. An informational flow chart describing this processing is provided in Figure 9.2.

Editors' Notes: *To be removed prior to final publication.*

The flow chart below will be removed in accordance with comment #190

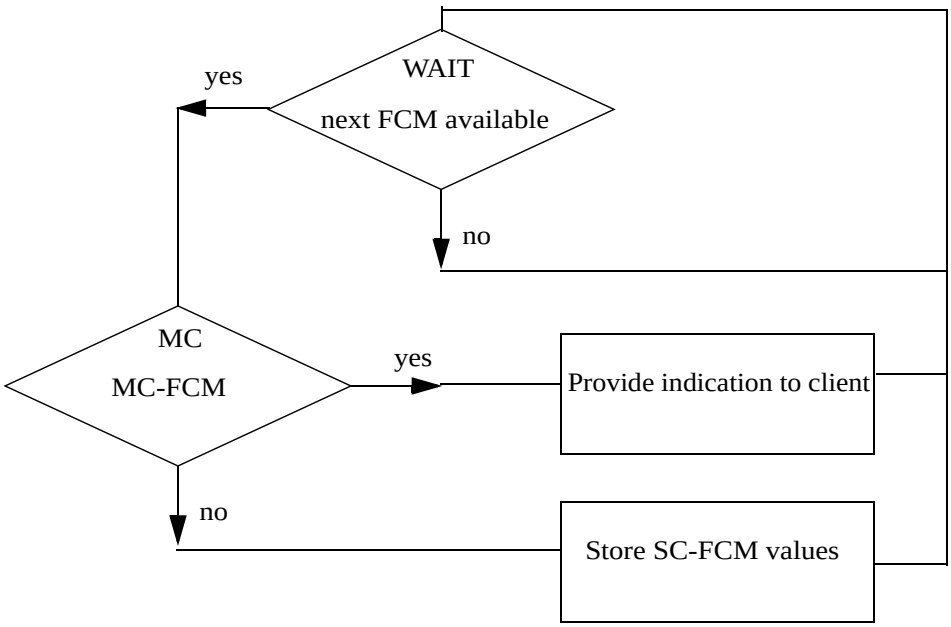


Figure 9.2— Fairness control message reception

9.3.2 *localFairRate* calculation

Using the most recently received SC-FCM, the *localFairRate* is recomputed every *agingInterval.* The localFairRate calculation is specified by Table 9.2 and Table 9.3. An informational flow chart describing this processing is provided in Figure 9.3.

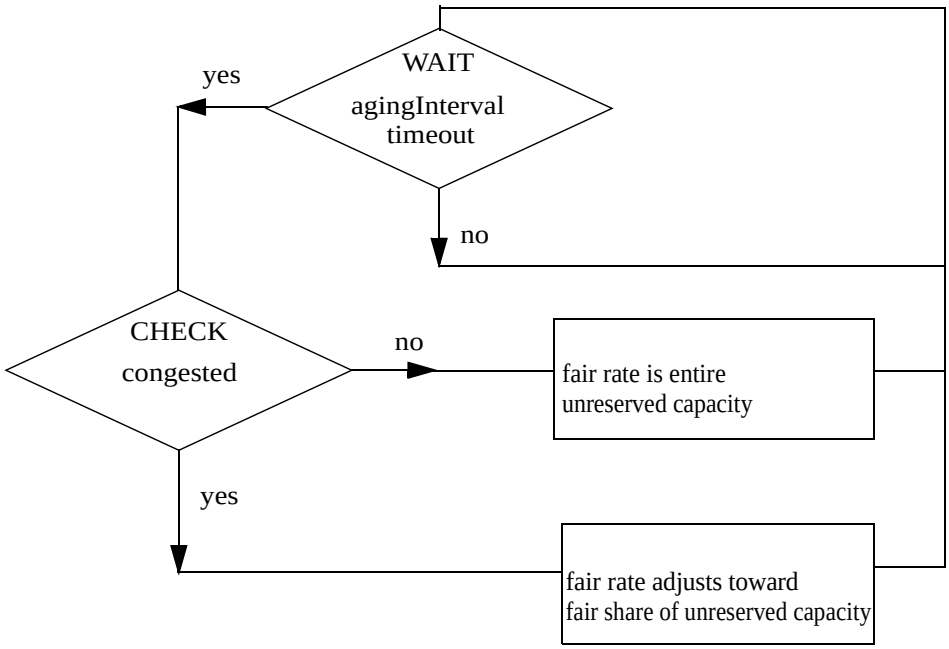


Figure 9.3—*localFairRate* calculation

9.3.3 Fairness control message transmit

9.3.3.1 Single-choke fairness control message transmit

For every *advertisementIntervals*, the *advertisedFairRate* and congested SA are recomputed (as described in 9.7.4) and advertised in a SC-FCM. FCM transmit processing is specified in the state machine of Table 9.9. An informational flow chart describing this processing is provided in Figure 9.4:.

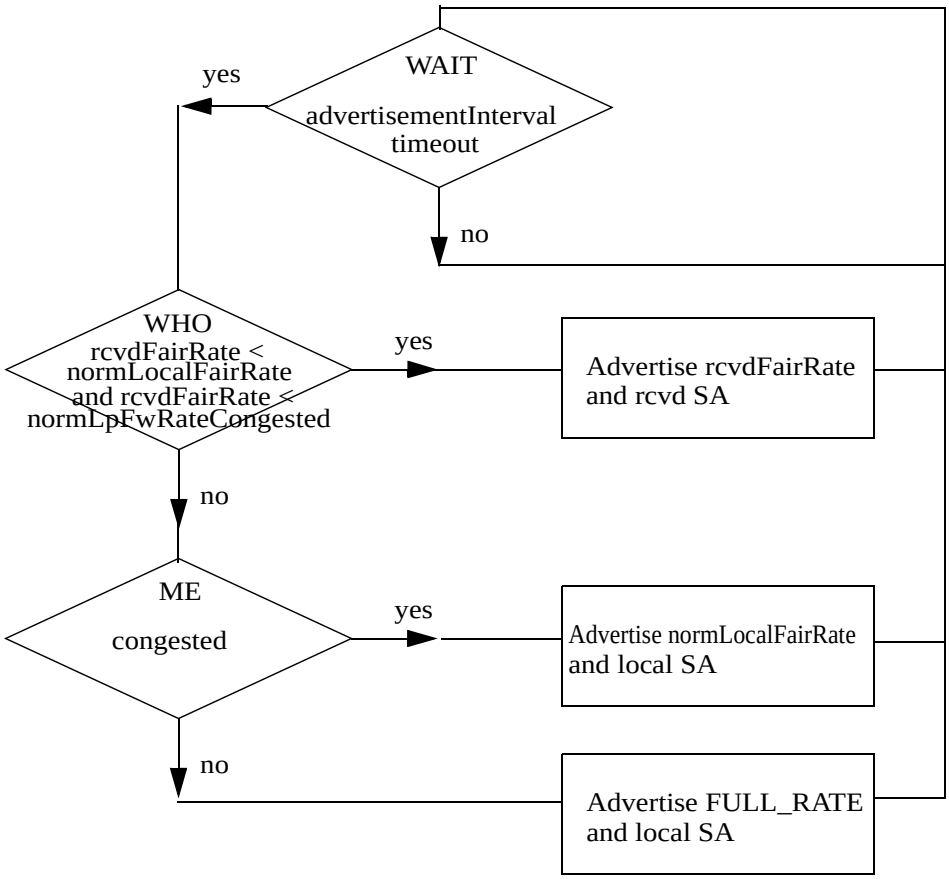


Figure 9.4—Single-choke fairness control message advertisement

9.3.3.2 Multi-choke fairness control message transmit

For every 10 *advertisementIntervals*, the *localFairRate* is advertised in a MC-FCM. MC-FCM transmit processing is specified in the state machine of Table 9.5. An informational flow chart describing this processing is provided in Figure 9.5.

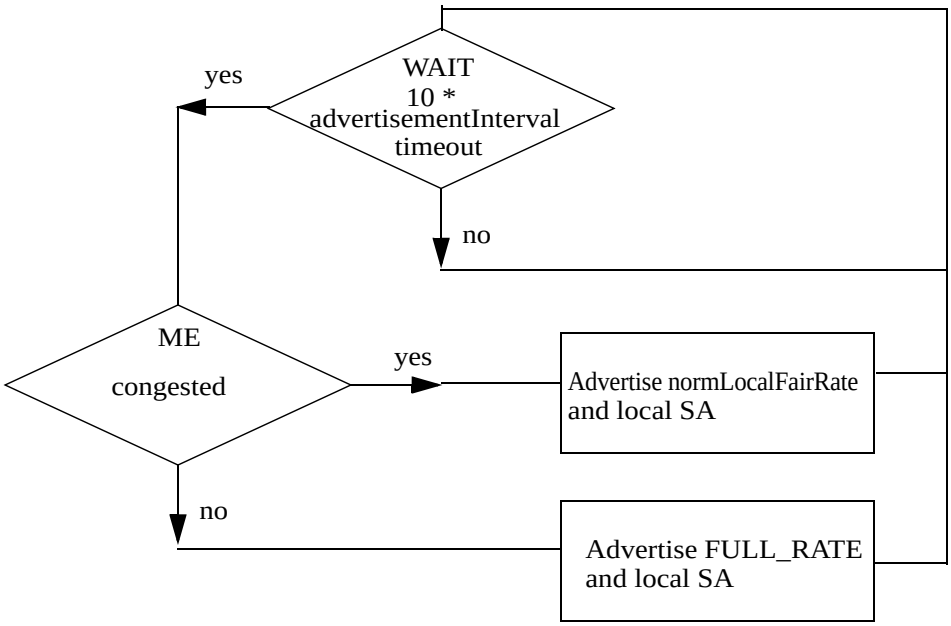


Figure 9.5—Multi-choke fairness control message advertisement

9.4 Congestion detection

Congestion is declared by using the following criteria. The formula for determining congestion is specified in 9.5.6.

Congestion is detected when any of the following conditions is true:

a) The rate of outgoing non-reserved traffic (*lpNrXmitRate*) is more than the unreservedRate (i.e. LINK_RATE - *reservedRate*).
b) The access delay timer for classB add packets expires.
c) The access delay timer for classC add packets expires.
d) The depth of the STQ exceeds the STQLowThreshold congestion depth threshold.
e) The rate of FE traffic exceeds the rateLowThreshold congestion rate threshold.

Conditions (b), (c), and (e) are optional for dual-queue MACs. Condition (d) is not applicable for single-queue MACs.

Editors' Notes: *To be removed prior to final publication.*

Adding (optional) hysteresis to each of a - e need to be added.

The above metrics are maintained independently for each ringlet. Any of them could indicate the onset of congestion for the ringlet. The absence of all that are maintained indicates the lack of congestion.

When a station determines that it is a congestion point, it will send a FCM containing a non-FULL_RATE fair rate, based on a normalized value of it low-pass filtered add rate.

9.4.1 Threshold settings

Editors' Notes: *To be removed prior to final publication.*

This text below will be updated to conform to STQxxxThreshold and ratexxxThreshold terminology.

Three thresholds, *fullThreshold*, *highThreshold*, and *lowThreshold*, are used to specify congestion thresholds. For dual-queue MACs, they are based on occupancy of the STQ. For single-queue MACs, they are based on the rate at which traffic is flowing through the MAC. The formulas for calculating these thresholds are provided in 9.7.1. When the *lowThreshold* is crossed, the station is considered to be approaching a congested state, and advertises a reduced fair rate. When the *highThreshold* is crossed, the station is considered to be in a congested state, and no more fairness-eligible traffic is accepted from the MAC client for transmission on to the ringlet. For dual-queue MACs, if the occupancy of the STQ crosses *fullThreshold*, it is almost full, and the traffic from the STQ will be prioritized above all other traffic until the occupancy of the STQ drops below *fullThreshold* (by any amount).

Editors' Notes: *To be removed prior to final publication.*

The remainder of this section deals with rationale, and should probably be moved to Annex I: Implementation Guidelines.

The setting of *lowThreshold* depends on the following factor:

a) If set too low, it will trigger congestion reports too frequently which in turn will adversely affect the link utilization.

The setting of *highThreshold* depends on the following factor:

a) If set higher, the difference between the *highThreshold* and *fullThreshold* is reduced. Because this reduces the time between congestion onset and the STQ filling up, it has the following effect:
1) For a given level of reclaimable classA bandwidth (i.e. subclassA1 bandwidth), the link distances must be shorter.
2) For a given link distance, a smaller level of reclaimable classA traffic can be supported. (See Clause 6, where this calculation is provided.)

The more severe consequences of a) vs. b) are the reason that a threshold of less than 50% is chosen. This recommendation applies only when a STQ of many MTUs is used to support high amounts of reclaimable classA bandwidth.

NOTE—*highThreshold* should be set to about 25% of the total buffer available. *fullThreshold* should be set to at least 1 MTU less than the total buffer size. The recommendation for the setting of the threshold values is done based on delivering the best possible end-to-end delay for the low priority traffic without penalizing the classA traffic. Lower values will result in higher end-to-end delays for low priority data packets. If either classA, classB, or classC traffic is extremely bursty, then a lower threshold value should be considered. A *fullThreshold* of at least 1 MTU less than the total buffer size is needed to ensure that the station has sufficient buffer space for a packet that may arrive on the transit path. If the classA traffic has a bursty nature, a more *conservative* (i.e. lower) value of *highThreshold* is recommended in order to sustain a higher level of subclassA1 and classB CIR traffic. If class A or classB traffic is extremely bursty, a congestion report will not have effect until the burst is over. By setting the lowThreshold to a higher value a congestion report might not be created and the wanted effect is achieved. This must however be measured against the fact that a higher value of the lowThreshold delays congestion reports when the increase in traffic is sustained. A high value for

lowThreshold also leaves less room in the STQ for class B and class C traffic, with the possible effect of penalizing class A traffic.

Editors' Notes: *To be removed prior to final publication.*

As requested at the May meeting, Necdet Uzun and David James will have a justification for this threshold.

9.4.2 Conservative and aggressive operation

Editors' Notes: *To be removed prior to final publication.*

An explanation of conservative vs. aggressive operation will be placed in this section.

9.4.3 Aging

Editors' Notes: *To be removed prior to final publication.*

Aging is accomplished by multiplying the count by the factor (AGECOEF-1)/AGECOEF at the completion of each agingInterval. This function bounds the count by the product of (a) AGECOEF and (b) the maximum value by which the count can increase during an agingInterval. If the count increases by a constant amount in each agingInterval, the value of the count asymptotically approaches the product of (a) AGECOEF and (b) the increase in the count during an aging interval. Hence, application of the aging function to a count allows the count to be interpreted as a rate.

A explanation of aging, including the above information will be placed in this section.

9.4.4 Low-pass filtering

Editors' Notes: *To be removed prior to final publication.*

Following description of low-pass filter function to be included in a subclause.

A new low-pass filtered value is calculated at the end of every filter interval as a weighted sum of (a) the observed value during the most recent filter interval and (b) the previous low-pass filtered value. The computation is: *lpFilteredValue* = (*observedValue* + (*lpFilteredValue* * (WEIGHTING_CONSTANT - 1))) / WEIGHTING_CONSTANT.

A explanation of low-pass filtering, including the above information will be placed in this section.

9.4.5 Ramping-up

Editors' Notes: *To be removed prior to final publication.*

A explanation of ramping-up will be placed in this section.

9.4.6 Normalizing

Editors' Notes: *To be removed prior to final publication.*

A explanation of normalizing will be placed in this section.

9.4.7 Choosing value of localFairRate on entering congested state for first time

The first time that the local station enters the congested state in the conservative mode of operation, localFairRate must be assigned a value lying between (a) unreservedRate / (numStations * WEIGHT) and (b) unreservedRate/WEIGHT.

An implementation is not required to use a particular method to select this value. The fairness algorithm will converge if the value of localFairRate is assigned within this range, but the choice of method may affect the time required for convergence. One method of assigning the value is:

localFairRate = unreservedRate / activeWeights;

```
where activeWeights can be computed as follows:

for (activeStations = 0, activeWeights = 0, station = 0; station < numStations; station++)
    if (sentFEDuringAgingInterval)
    {   ++ activeStations;
        activeWeights += Weights[station];
    }
```

Editors' Notes: *To be removed prior to final publication.*

Add examples of additional methods of computing activeWeights.

9.5 Fairness control state machine

Editors' Notes: *To be removed prior to final publication.*

This subclause should provide a brief description of the state machine; also, it should provide the normative "shall" statements that will be later referenced by the PICS, and also provides a pointer to the notational reference in Clause 3. Note that this template makes provision for a single state machine to be represented by one or more state tables or diagrams that are related to each other (e.g., if it becomes necessary for a large state table to be broken into several smaller state tables in the interest of clarity and maintainability).

The fairness state machine specifies the fairness algorithm described in section 9.x.

The fairness control state machine description is divided into the following state tables:

a) fairness control message processing
b) congestion detection and localFairRate calculation (aggressive)
c) congestion detection and localFairRate calculation (conservative)
d) generate fairness message (single-choke)
e) generate fairness message (multi-choke)
f) per-byte statistics

The fairness control state machine shall implement the state tables specified in this section and meet the corresponding state table interface requirements, also specified here. 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 machine is described in 3.4.

NOTE—Implementations may choose any conforming means of realizing the state tables described herein, provided that the external behavior of the fairness control state machine is unchanged.

9.5.1 Inputs

Editors' Notes: *To be removed prior to final publication.*
<specifies the inputs to the state machine, referencing service interface primitives or other state machine variables as required. Cross-references to the sources of the inputs are to be provided. NOTE: the format uses the IEEE standard VariableList paragraph tag; use shift-return to insert a newline without starting a new paragraph.>
The following is an example of the format that will be used in this subsection to describe inputs.

powerOn

A condition that is true until such time as the power supply to the device containing the state machine has reached the operating region.

Values:

FALSE; The device is completely powered and ready to operate.

TRUE; The device has not been completely powered.

Inputs currently listed in section 9.2 will be moved to this section. The editor is aware that this needs to be done.

9.5.2 Constants

Editors' Notes: *To be removed prior to final publication.*
<descriptions of any constants that are used within the state tables or diagrams. It is not necessary to represent simple integer numbers (e.g., 0, 1) as named constants unless they have special significance. >
The following is an example of the format that will be used in this subsection to describe constants.

HUNTPAT

A sequence of 6 consecutive A1 bytes.

Constants currently listed in section 9.2 will be moved to this section. The editor is aware that this needs to be done.

9.5.3 Variables

Editors' Notes: *To be removed prior to final publication.*
<descriptions of variables that are used within the state diagrams or tables. For example, a variable may be set by one portion of a state table, and later used by another portion. Variables may also be set by one state table and used by another state table, if both state tables belong to the same state machine. >
The following is an example of the format that will be used in this subsection to describe variables.

byteCnt

Count of bytes input to the state machine; always increments for every byte input. This variable is forced to zero in specific states of the YYYY state machine, but for all other states it increments by 1 for each byte received. It is used to force the state machine to wait in a given state until a required number of bytes have been received.

Variables currently listed in section 9.2 will be moved to this section. The editor is aware that this needs to be done.

9.5.4 Functions

Editors' Notes: *To be removed prior to final publication.*
<descriptions of functions that are required by the state machine. Functions are typically used to simplify the depiction of state diagrams, or the descriptions within cells of state tables. In particular, functions may be used to reduce long conditional expressions governing state transitions to manageable and understandable forms. Each function should be given a descriptive name, after which a brief textual description of the function should be provided. The normative definition of the function (in words, or C-code fragments) should follow the textual description. Some examples are provided below.>The following is an example of the format that will be used in this subsection to describe functions.

foundHunt

For each bit input to the Synchronization process, this function indicates whether the HUNTPAT pattern has been detected in the bit string formed by concatenating the current bit with the 47 consecutive previously input bits. If the number of bits previously input is less than 47, this function outputs a FALSE value. Note that this function inspects its input on a bit-by-bit basis.

Values:

TRUE; The 48 bits examined so far, consisting of the current bit plus previously input bits, matches HUNTPAT.

FALSE; The pattern is not matched, or less than 48 bits have been input so far.

Code snippets in this section should use the increment and decrement operators when appropriate (i.e. A += B rather than A = A+B). (comment #220)
Functions currently listed in section 9.3 will be moved to this section. The editor is aware that this needs to be done.

9.5.5 Fairness control message processing state table

Editors' Notes: *To be removed prior to final publication.*
<A brief description of the state table, followed by the state table, should now be provided. The description does not have to be very long, but should give the reader a sense of what the state diagram/table does, and also relate it to other state diagrams/tables with which it may interact.>
The description has not yet been provided but will be added.

Table 9.2—Receive fairness message

Current state		Row	Next state	
state	**condition**		**action**	**state**

WAIT	TxMuxN:MA_DATA .indicate(PT == 0x02) && SC_FCM	1	rcvdFairRate = msg.controlValue;
			rcvdSA = msg.SA;
			rcvdTTL = msg.TTL;
			rcvdRI = msg.RI;
			if (rcvdFairRate != FULL_RATE) {
			downstreamCongested = TRUE;
			if (wrappedRing && (rcvdRI != ringId)) {
			hopsToCongestion = hopsToWrapped-DownstreamStation;
			}
			else hopsToCongestion = 256 - rcvdTTL;
			} else {
			downstreamCongested = FALSE;
			}
				WAIT

		2		

Row 9.2-1: FCM-Single choke received. TxMuxN:MA_DATA.indicate is the signal generated by the MAC data path sublayer to the MAC control sublayer indicating that an FCM has arrived.
Row 9.2-2: FCM-Multi choke received. MAC_control.indicate is the signal generated by the MAC to the client indicating that control information is available.

9.5.6 Congestion detection and local fair rate calculation (aggressive)

Editors' Notes: *To be removed prior to final publication.*
<A brief description of the state table, followed by the state table, should now be provided. The description does not have to be very long, but should give the reader a sense of what the state diagram/table does, and also relate it to other state diagrams/tables with which it may interact.>
The description has not yet been provided but will be added.

Table 9.3—Congestion detection and local fair rate calculation (aggressive) state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
INIT		1	initializeVariables();	UNCG

UNCG	agingIntervalExpired && isCongested()	2	localCongested = TRUE;
			localFairRate = lpAddRate;
			normLocalFairRate = localFairRate / NORMCOEF;
			setAllowedRateCongested (rcvdFairRate);
			agingIntervalUpdate();
				CGST

	agingIntervalExpired	3	setAllowedRateCongested (rcvdFairRate); agingIntervalUpdate();	UNCG

CGST	agingIntervalExpired && (!isCongested())	4	localCongested = FALSE;
			localFairRate = unreservedRate;
			normLocalFairRrate = localFairRate / NORMCOEF;
			setAllowedRateCongested (rcvdFairRate);
			agingIntervalUpdate();
				UNCG

	agingIntervalExpired	5	localCongested = TRUE;
			localFairRate = lpAddRate;
			normLocalFairRate = localFairRate / NORMCOEF;
			setAllowedRateCongested (rcvdFairRate);
			agingIntervalUpdate();
				CGST

Row 9.3-1: Initialize threshold variables. Transition to Uncongested state. The initializeVariables() function is shown below:

```
void initializeVariables()
{
    unreservedRate = lineRate - reservedRate;
    STQHighThreshold = STQSize / 4;
    STQLowThreshold = STQSize / 8;
    rateHighThreshold = 0.95 * unreservedRate;
    rateLowThreshold = 0.8 * unreservedRate;
    localCongested = FALSE;
```

```
    localFairRate = unreservedRate;
    normLocalFairRate = localFairRate / NORMCOEF;
    allowedRate = MAX_ALLOWED_RATE;
}
```

Row 9.3-2: Congestion detected. Transition to Congested state. The isCongested() function is given below:

```
Boolean isCongested()
{
    if ((dualQueueMAC && (
        (lpNrXmitRate > unreservedRate) || (STQDepth > STQLowThreshold)))
        return TRUE;
    else if ((singleQueueMAC && ((lpNrXmitRate > unreservedRate)
            || (lpNrXmitRate > rateLowThreshold) || classBAccessDelayTimerExpired
            || classCAccessDelayTimerExpired)))
        return TRUE;
    else
        return FALSE;
}

void setAllowedRateCongested (rcvdFairRate)
{
    if (rcvdFairRate != FULL_RATE)
    {
        allowedRateCongested = rcvdFairRate * NORMCOEF;
    } else {
        allowedRateCongested += (MAX_ALLOWED_RATE - allowedRateCongested) / RAMPCOEF;
    }
}

void agingIntervalUpdate ()
{
    lpAddRate = ((LPCOEF-1) * lpAddRate + addRate) / LPOEF;
    lpAddRateCongested = ((LPCOEF-1) * lpAddRateCongested + addRateCongested) / LPOEF;
    lpFwRate = ((LPCOEF-1) * lpFwRate + fwRate) / LPOEF;
    lpFwRateCongested = ((LPCOEF-1) * lpFwRateCongested + fwRateCongested) / LPOEF;
    lpNrXmitRate ((LPCOEF-1) * lpNrXmitRate + nrXmitRate) / LPOEF;
    normLpFwRateCongested = lpFwRateCongested / NORMCOEF;

    addRate = (addRate * (AGECOEF - 1)) / AGECOEF;
    addRateCongested = (addRateCongested * (AGECOEF - 1)) / AGECOEF;
    fwRate = (fwRate * (AGECOEF - 1)) / AGECOEF;
    fwRateCongested = (fwRateCongested * (AGECOEF - 1)) / AGECOEF;
    nrXmitRate = (nrXmitRate * (AGECOEF - 1)) / AGECOEF;
}
```

Row 9.3-3: Else there is no Congestion. Remain in Uncongested state.
Row 9.3-4: Comes out of congestion. Transition to Uncongested state.
Row 9.3-5: Else congestion exists. Remain in Congested state.

9.5.7 Congestion detection and localFairRate calculation (conservative)

Editors' Notes: *To be removed prior to final publication.*
A brief description of the state table, followed by the state table, should be provided. The description should give the reader a sense of what the state diagram/table does, and also relate it to other state diagrams/tables with which it may interact. The description has not yet been provided but will be added.

Table 9.4—Congestion detection and local fair rate calculation (conservative) state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
INIT		1	initializeVariables()	UNCG

UNCG	agingIntervalExpired && isCongested()	2	localFairRate = (unreservedRate / activeStations) * WEIGHT;
			normLocalFairRate = localFairRate / NORMCOEF;
			localCongested = TRUE;
			allowedRate = localFairRate;
			resetRTTCounter();
			setAllowedRateCongested (rcvdFairRate);
			agingIntervalUpdate();
				CGST

	agingIntervalExpired	3	allowedRate += (MAX_ALLOWED_RATE - allowedRate) / RAMPCOEF;
			setAllowedRateCongested (rcvdFairRate);
			agingIntervalUpdate();
				UNCG

CGST	agingIntervalExpired && (localFairRate >= unreservedRate)	4	localFairRate = unreservedRate;
			localCongested = FALSE;
			normLocalFairRate = localFairRate / NORMCOEF;
			allowedRate += (MAX_ALLOWED_RATE - allowedRate) / RAMPCOEF;
			setAllowedRateCongested (rcvdFairRate);
			agingIntervalUpdate();
				UNCG

	agingIntervalExpired && ((addRate + fwdRate > rateHighThreshold) || (STQDepth > STQHighThreshold)) && RTTWorthOfIntervalsHavePassed	5	localFairRate = localFairRate - localFairRate / RAMPCOEF;
			normLocalFairRate = localFairRate / NORMCOEF;
			allowedRate = localFairRate;
			resetRTTCounter();
			setAllowedRateCongested (rcvdFairRate);
			agingIntervalUpdate();
				CGST

	agingIntervalExpired && ((addRate + fwdRate < rateLowThreshold) || (STQDepth < STQLowThreshold)) && RTTWorthOfIntervalsHavePassed	6	localFairRate = min (unreservedRate, localFairRate + (unreservedRate - localFairRate) / RAMPCOEF);
			normLocalFairRate = localFairRate / NORMCOEF;
			allowedRate = localFairRate;
			resetRTTCounter();
			setAllowedRateCongested (rcvdFairRate);
			agingIntervalUpdate();
				CGST

	agingIntervalExpired	7	allowedRate = localFairRate;
			setAllowedRateCongested (rcvdFairRate);
			agingIntervalUpdate();
				CGST

Row 9.4-1: Initialize threshold variables. Transition to Uncongested state.
Row 9.4-2: Congestion detected. Calculate localFairRate based on the number of active sources and reset the RTT interval counter. Transition to Congested state.
Row 9.4-3: Else there is no Congestion. Remain in Uncongested state.

Row 9.4-4: Comes out of congestion. Transition to Uncongested state.
Row 9.4-5: Else if needed, ramp down the local_fair_rate. Remain in Congested state.
Row 9.4-6: Else if needed, ramp up the local_fair_rate. Remain in Congested state.
Row 9.4-7: Else congestion exists. Remain in Congested state.

9.5.8 Generate fairness message (single-choke) state table

Table 9.5—Generate fairness message - single choke

Current state		Row	Next state	
state	**condition**		**action**	**state**

WAIT	advertisementIntervalExpires && localCongested && (!downstreamCongested || (normLocalFairRate < rcvdFairRate))	1	msg.fairnessMsgType = 0x00;
			msg.controlValue = normLocalFairRate;
			msg.SA= localSA;
			msg.TTL = 255;
			msg.RI = ringId;
				WAIT

	advertisementIntervalExpires && downstreamCongested && (!localCongested || (normLocalFairRate > rcvdFairRate))	2	msg.fairnessMsgType = 0x00;
			if (rcvdFairRate < normLpFwRateCongested)
			{
			msg.controlValue = rcvdFairRate;
			msg.SA = rcvdSA;
			msg.TTL = rcvdTTL;
			msg.RI = rcvdRI;
			}
			else
			{
			msg.controlValue = FULL_RATE;
			msg.SA = localSA;
			msg.TTL = 255;
			msg.RI = ringId;
			}	

	advertisementIntervalExpires && (!downstreamCongested) && (!localCongested)	3	msg.fairnessMsgType = 0x00;
			msg.TTL = 255;
			msg.RI = ringId;
			msg.controlValue = FULL_RATE;
			msg.SA = localSA;	

Row 9.5-1: Local station is more congested than downstream station. Advertise normLocalFairRate.
Row 9.5-2: Downstream station is more congested than the local station. Advertise rcvdFairRate if congestion is caused by upstream station else advertise FULL_RATE.
Row 9.5-3: No congestion. Advertise FULL_RATE.Congestion detection and localFairRate calculation (aggressive) state table

9.5.9 Generate fairness message (multi-choke) state table

Editors' Notes: *To be removed prior to final publication.*
A brief description of the state table, should be provided. The description should give the reader a sense of what the state diagram/table does. The description has not yet been provided but will be added.

Table 9.6—Generate fairness message - multi choke

Current state		Row	Next state	
state	**condition**		**action**	**state**

WAIT	(10 * advertisementIntervalExpires) && localCongested	1	msg.fairnessMsgType = 0x01;
			msg.controlValue = normLocalFairRate;
			msg.SA= localSA;
			msg.TTL = 255;
			msg.RI = ringID;
				WAIT

	(10 * advertisementIntervalExpires)	2	msg.fairnessMsgType = 0x01;
			msg.controlValue = FULL_RATE;
			msg.SA= localSA;
			msg.TTL = 255;
			msg.RI = ringID;	

Row 9.6-1: Local station is congested. Advertise normLocalFairRate.
Row 9.6-2: Local station is not congested. Advertise FULL_RATE.

9.5.10 Per-byte statistics state table

Table 9.7—Per-byte statistics state table

Current state		Row	Next state	
state	**condition**		**action**	**state**

WAIT	FEbyteAddedByLocalStation	1	addRate++;
			if (sentBeyondCongestionPoint) {
			addRateCongested++;
			}
			if (nonA0ByteTransmitted) {
			nrXmitRate++;
			}
				WAIT

	FEbyteForwardedByLocalStation	2	fwRate++;
			if (sentBeyondCongestionPoint) {
			fwRateCongested++;
			}
			if (nonA0ByteTransmitted) {
			nrXmitRate++;
			}	

Row 9.7-1: Local station is adding a byte on the outgoing link. Increment addRate. Increment addRateCongested if the byte is destined beyond the congestion point. If the byte belongs to class A1, B or C traffic, then increment nrXmitRate.
Row 9.7-2: Local station is forwarding a byte sent by an upstream station. Increment fwRate. Increment fwRateCongestion if the byte is destined beyond the congestion point. If the byte belong to class A1, B or C traffic, then increment nrXmitRate.

Editors' Notes: *To be removed prior to final publication.*
A brief description of the state table should be provided. The description should give the reader a sense of what the state diagram/table does.The description has not yet been provided but will be added.

9.6 Interaction with data path

Editors' Notes: *To be removed prior to final publication.*

This material is covered in other section of the clause. It will be removed and appropriate references to the material will be placed in this editors' note.

FA uses the *addRate, addRateCongested, fwRate, fwRateCongested, and nrXmitRate* values provided by the data path, as specified in Clause 6.

FA provides the *allowedRate*, *allowedRateCongested*, and *hopsToCongestion* values to the data path rate control function to police the fairness-eligible traffic that is added by the station to the ringlet. The *hopsToCongestion* variable stores the value of the distance to the present or more recent choke-point. The *allowedRateCongested* variable stores the value of the fair rate for fairness-eligible add traffic passing through the *congestionPoint*, as given by the most recently received single-choke FCM. The *allowedRate* variable stores the value of the rate for all fairness-eligible add traffic. The data path maintains rate shapers to limit the *addRate* (shD) and *addRateCongested* (shC) values below the provided *allowedRate* and *allowedRateCon-*

gested values, respectively, and to shape their output at or below the provided rates to smooth bursty transmissions.

9.7 Fairness control messages

9.7.1 Generation of fairness control messages

Single-choke and multi-choke fairness control messages shall be sent periodically to propagate fair rate information to upstream stations. Single-choke messages are sent every *advertisementInterval*. The recommended *advertisementInterval* is between the transmission delay for a single MTU and the *agingInterval*. Multi-choke messages are sent every 10 *advertisementInterval*s. The values ringControl subfields specified in a fairness control message are specified in Table 9.8.

Table 9.8—Fairness control message ringControl values

Field	Value (bin)
frameType	fairness frame (10)
serviceClass (SC)	subclass A0 (11)
fairnessEligibile (FE)	not subject to fairness (0)
wrapEligible (WE)	eligible (1)
ringletID (RI)	RI of originating ringlet
parity (P)	set such that total number 1's in ringControl field is odd

9.7.2 Impact of lost fairness control messages

The *allowedRateCongested* maintains its current value until the next single-choke FCM is received. Therefore, the loss of a single-choke FCM would result in one of the following. If the lost message contained a new value of FULL_RATE, it would prevent the station from ramping up its allowed rate for the ringlet. If the value contained in the lost message was a new non-FULL_RATE value, the MAC will not be able to react by setting its allowed rate to the received advertised rate, which could have been higher or lower than the current allowed rate. If the lost message did not contain a new value from the previous advertisement, there will be no effect on the fairness algorithm.

Editors' Notes: *To be removed prior to final publication.*

The following statement has been modified (comment #202 11/02) and will be updated when the text described below (from Jason Fan) is available in Clause 11.

"Clause 11 talks about the criteria for declaring a loss of keepalive, which is a signal fail (SF) condition. The criteria tie to failing to receive several consecutive single-choke FCM's (the actual text for this will be modified from D1.1 to D2.0). So I believe the text in Clause 9 should refer to Clause 11."

Refer to Clause 11 for the protection implications of not receiving consecutive single-choke FCMs.

The loss of an MC-FCM may result in the failure of one or more clients to learn the current value of the *localFairRate* associated with the station originating the message.

9.7.3 Validation of fairness control messages

Editors' Notes: *To be removed prior to final publication.*

This following paragraph probably belongs in Clause 6 and should be generalized for all control messages.

If the source of the *rcvdFairRate* is the same station that received it then the *rcvdFairRate* will be disregarded. When comparing the *rcvdFairRate* source address, the ring identifier of the FCM must match the receiver's ring identifier in order to qualify as a valid compare. The exception is if the receive station is in the center wrap state, in which case the FCM's ring identifier is ignored.

9.7.4 Fairness control message format

The FCM format is as shown in Figure 9.6. The fairness protocol determines the number of hops to the station that originated the FCM by subtracting the TTL in the received message from 256. Therefore all FCMs shall be sourced with a TTL of 255. And all single-choke FCMs shall reset the TTL to 255 upon changing the value of the SA to the local SA.

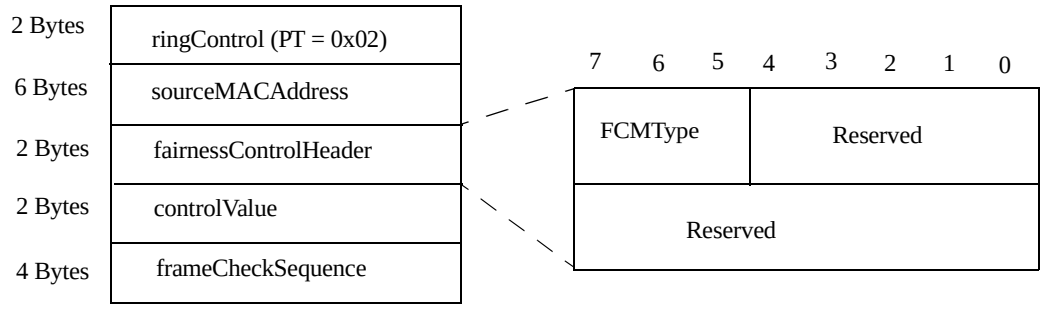


Figure 9.6—Fairness control message format

9.7.4.1 *FCMType* (3 bits)

This field specifies the type of FCM. Table 9.9 shows the values of the FCMType. Single-choke FCMs are used to implement the FA and provide information about the most congested span on the ringlet. Multi-choke FCMs are needed to support a multi-choke implementation.

9.7.4.2 Reserved field (13 bits)

These bits are ignored on receipt and set to zero on transmit for both single-choke and multi-choke FCMs.

Table 9.9—Fairness control message type values

Value (binary)	Type of fairness control message	How is it used
000	single-choke	Generated by the FCU every *advertisementInterval*. SA is the MAC address of the most congested station in the fairness domain
001	multi-choke	Generated by the FCU every 10 *advertisementIntervals*. The SA is the SA of the MAC, and the messages are broadcast. The rcvdFairRate is passed to the client.
010 to 111	Reserved	For future use.

9.7.4.3 *controlValue* (16 bits)

Editors' Notes: *To be removed prior to final publication.*

The paragraph below has been rewritten to clarify that the controlValue carries the advertised fair rate. Text describing the method of normalization has been removed and will be placed earlier in the text in order to avoid repetition of this description. ***(comment #204, 205)***

This field carries a normalized fair rate encoded as a 16-bit quantity. A value of FULL-RATE (all ones) indicates the full line rate. The *controlValue* carries the *advertisedFairRate* to an upstream neighbor in the case of an SC-FCM, or to all stations on the ringlet in the case of an MC-FCM.

9.8 Informative C-code

Editors' Notes: *To be removed prior to final publication.*

This section contains informative C-code describing the fairness control functions. The code will be moved to an informative annex after it has been reconciled with the fairness control state machine. Section numbers appearing in comments within this code should be ignored as their values will change when the code is moved into the informative annex. This code has not been divided into framemaker sections or modified in any way from the compiled text provided.

```
/* FCU2 */
#include <stdio.h>
#include <math.h>
enum boolean {FALSE, TRUE};

boolean A0() // dummy call without variable
{
 return TRUE;
}
boolean A1(int n) // dummy call with one input
{
return TRUE;
}
rcvdMsg (int rcvd_SA, int rcvdFairRate, int rcvd_TTL)
```

```
{
    // received msg processor returns received source address
    // using dummy values
    int const sourceAddress = 0x01;
    rcvd_SA = sourceAddress;

    // received msg processor returns received fair rate
    int const rcvdMsgRate = 0x622;
    rcvdFairRate = rcvdMsgRate;

    // received TTL
    int const TTL = 64;   // dummy value
    rcvd_TTL = TTL;
    return 0;
}


/* ********************************************************************** */
// c-code updated with input from draft 1.2 in the editor's note
// December 5, 2002 17:11
// activeWeights: method 4 proposed
// sendC and sendCTTL deleted

main (int LINK_RATE, int Weights[], int rprIfNodesOnRing,//9.2:32,62,_
        int  NORMCOEF,  int  LPCOEF,  int  RAMPCOEF,  int  STQ_SIZE,  int  MTU_SIZE,    //
9.2:47,41,51,_,_
      int advertismentRatio, int AGECOEF, int agingInterval,//9.2:9,10,11
      int reservedRate, int STQ_depth, int local_SA,//9.2.56,_,_

      boolean aggressive, boolean conservative //9.2:14,20

       )
{


int activeStations, activeWeights, addRate, addRateCongested,//9.2:1,2,5,6
    advertisedFairRate,//9.2.7
    allowedRate, allowedRateCongested, congestionPoint,//9.2:17,18,19
    FULL_RATE,//9.2.24
    fwRate, fwRateCongested,//9.2:29,30
    localFairRate,//9.2.34
    lpAddRate, lpAddRateCongested,//9.2:36,37
    lpFwRate, lpFwRateCongested, lpNrXmitRate,//9.2:38,39,40
    normLocalFairRate, normLpFwRateCongested,normLpFwRate,//9.2:45,46,_
    nrXmitRate;//9.2.49
// The following variables are defined but not used in this version c-code
//  advertisementInterval, controlValue;//9.2:8,21
//  downstreamCongested;//9.2.22
//  rateFullThreshold;//9.2.25
//  congestionPoint;//9.2.19
//  RATECOEF//9.2.52
const int MAX_TTL =255;
int STQHighThreshold, STQLowThreshold;//9.2:58,59
int STQfullThreshold;//9.2.26
float rateHighThreshold, rateLowThreshold;//9.2:53,54
int unreservedRate;//9.2.61

// variable assignments
// addRateCongestedOK
boolean localCongested, dualQueueMAC;// 9.2:33,23
boolean just_entered_congested;
boolean rcvdRInotMatch, rcvdRIMatch;//

// new variables:
int ttlToWrap;// used in 9.7.3.6
```

```
int rcvd_SA, rcvdFairRate, rcvd_TTL;//9.2:_,55,_
int hopsToCongestion;// 9.2.31
rcvdMsg (rcvd_SA, rcvdFairRate, rcvd_TTL);// call rcvdMsg to get info


FULL_RATE = 0xffff;// 9.2.24
if (MTU_SIZE = 0) // default MTU size is 9216 bytes
    MTU_SIZE = 9216;


int MAX_ALLOWED_RATE = LINK_RATE;//9.2.24
int  numStations = rprIfNodesOnRing;//9.2.50
// checks if LPCOEF and NORMCOEF are powers of 2
// otherwise assign 64

if (  fmod(log10(LPCOEF)/log10(2), 1) != 0)
    LPCOEF = 64; // powers of 2's
if (  fmod(log10(NORMCOEF)/log10(2), 1) != 0)
    NORMCOEF = 64;

if ( STQ_SIZE == 0 ) // dualQueueMAC can be determined from STQ_SIZE
    dualQueueMAC = boolean (FALSE) ;
else
    dualQueueMAC = boolean (TRUE);

//9.7.1 Initialization

// INIT
allowedRate = MAX_ALLOWED_RATE;
unreservedRate = LINK_RATE - reservedRate;
if (dualQueueMAC)
   STQfullThreshold = STQ_SIZE - MTU_SIZE;

if (dualQueueMAC)
   STQHighThreshold = (STQ_SIZE / 4);
else
   rateHighThreshold = float (.95 * unreservedRate);

if (dualQueueMAC)
   STQLowThreshold = (STQ_SIZE / 8);
else
    rateLowThreshold = float(.8 * unreservedRate);
localCongested = FALSE;


// 9.7.2 Per byte
boolean FE_byte_added_by_local_station, sent_beyond_congestionPoint;
boolean FE_byte_forwarded_by_local_station;
boolean non_A0_byte_transmitted_by_local_station;
boolean classA;

// dummy function calls
FE_byte_added_by_local_station = A0 ();     // function that determines FE
sent_beyond_congestionPoint = A0 ();    // function that determines packet sent beyond
           // congestion point
FE_byte_forwarded_by_local_station = A0 ();
non_A0_byte_transmitted_by_local_station = A0();
classA = A0 ();

if ( FE_byte_added_by_local_station )
   ++ addRate;

if ( FE_byte_added_by_local_station &&  sent_beyond_congestionPoint  )
   ++ addRateCongested ;
```

```
if ( !classA && FE_byte_forwarded_by_local_station )
   ++ fwRate ;

if ( !classA && FE_byte_forwarded_by_local_station  && sent_beyond_congestionPoint )
   ++ fwRateCongested ;

if ( non_A0_byte_transmitted_by_local_station )
   ++ nrXmitRate;

//9.7.2.1 addRateOK, addRateCongestedOK
boolean addRateOK;// _._._ [Clause 6]

addRateOK = boolean (
    ( addRate < allowedRate)// current allowance is not exceeded
    && (nrXmitRate < unreservedRate)// space for reserved traffic
    && ( (STQ_depth == 0)// upstream stations are not in need
    || ( (fwRate > addRate)// upstream stations are not starved
    && ( STQ_depth < STQHighThreshold )) )  // station is not fully congested
 );


//9.7.3 Per agingInterval

//9.7.3.1 activeStation, activeWeights
/*
 method 1

  activeWeight = numStations /Weights(localStation);

  method 2:


for (activeWeights = 0,station =0; station < numStations, station ++)
{
    if ( send_FE_duringaging_interval (station ))
        activeWeights = activeWeights +  (activeStations/ Weights(localStation));
}

   method 3:

for (activeWeights = 0,station =0; station < numStations, station ++)
{
    if ( send_FE_duringaging_interval (station ))
        activeWeights = activeWeights +  (activeStations/ Weights(station));

*/
int station;
boolean sentFEDuringAgingInterval = A1(station);

for ( activeStations = 0, activeWeights = 0, station = 0;
    station < numStations; station++ )   // local station included
if ( sentFEDuringAgingInterval )
{   ++ activeStations;
    activeWeights += Weights[station];
}

// 9.7.3.2lpAddRate,
//         lpFwRate, lpFwRateCongested, lpNrXmitRate,
//         normLpFwRateCongested

lpAddRate = ((LPCOEF-1) * lpAddRate + addRate) / LPCOEF;
lpAddRateCongested = ((LPCOEF-1) * lpAddRateCongested + addRateCongested) / LPCOEF;
lpFwRate = ((LPCOEF-1) * lpFwRate + fwRate) / LPCOEF;
lpFwRateCongested = ((LPCOEF-1) * lpFwRateCongested + fwRateCongested) / LPCOEF;
```

```
lpNrXmitRate = ((LPCOEF-1) * lpNrXmitRate + nrXmitRate) / LPCOEF;
normLpFwRateCongested = lpFwRateCongested / NORMCOEF;

// alternatively, each can be simplified as shown for lpAddRate
// lpAddRate = lpAddRate -
// lpAddRate>>(log2(LPCOEF)) + addRate>>(log2(LPCOEF));


//9.7.3.3 addRate, addRateCongested, fwRate, fwRateCongested, nrXmitRate


addRate = (addRate * (AGECOEF - 1)) / AGECOEF;
addRateCongested = (addRateCongested * (AGECOEF - 1)) / AGECOEF;
fwRate  = (fwRate * (AGECOEF - 1)) / AGECOEF;
fwRateCongested = (fwRateCongested * (AGECOEF-1)) / AGECOEF;
nrXmitRate = (nrXmitRate * (AGECOEF - 1)) / AGECOEF;

//9.7.3.4 congested
boolean                                          access_delay_timer_for_add_classB_expired,
access_delay_timer_for_add_classC_expired;
access_delay_timer_for_add_classB_expired = A0();
access_delay_timer_for_add_classC_expired =  A0();
// this does not have hysteresis
// for hysteresis first time congest crosses high threshold
// not congest crosses rateHighThreshold - rateHysteresis
// still congest crosses rateHighThreshold
//
if ( dualQueueMAC )
{
   if ( (lpNrXmitRate > unreservedRate) || ( STQ_depth > STQLowThreshold)) // !!!!!!
       localCongested = TRUE;
    else if ( aggressive ) // clearing of congested for conservative mode
       localCongested = FALSE; // is handled in calculation for localFairRate
}


else // mono_queue_MAC
    {
    if (( lpNrXmitRate > unreservedRate)
        || (lpNrXmitRate > rateLowThreshold)
        || (access_delay_timer_for_add_classB_expired)
    || (access_delay_timer_for_add_classC_expired))  // is rate High??????
        localCongested = TRUE;
else if ( aggressive )      // clearing of congested for conservative mode ?????
    localCongested = FALSE; // is handled in calculation for localFairRate
    }

//9.7.3.5 allowedRate
// replace aggressive with

if ( aggressive )
    allowedRate = MAX_ALLOWED_RATE;
else // conservative
{
    if ( localCongested ) // local station is congested
        allowedRate = localFairRate;
    else // local station is not congested
        allowedRate = allowedRate+(MAX_ALLOWED_RATE-allowedRate)/RAMPCOEF;
}


// 9.7.3.6 allowedRateCongested, congestionPoint, hopsToCongestion

// call from FCU for received message information
```

```
if (rcvdFairRate != FULL_RATE)
{
    allowedRateCongested = rcvdFairRate * NORMCOEF;
    congestionPoint = rcvd_SA ;
    if ( rcvdRIMatch )
        hopsToCongestion = MAX_TTL - rcvd_TTL;
    else
        hopsToCongestion = ttlToWrap;
}

else // no congestion downstream, ramp up
{
    allowedRateCongested = allowedRateCongested +
    (MAX_ALLOWED_RATE - allowedRateCongested) / RAMPCOEF;
    congestionPoint = local_SA;
    hopsToCongestion = hopsToCongestion;
}
if ( allowedRateCongested == MAX_ALLOWED_RATE )
    hopsToCongestion = MAX_TTL;

// hopsToCongestion should be left at the last congestion point so that
// the allowedRate is not all of a sudden presented to the former
// congestion point. One could add the following statement:
// if (allowedRateCongested == allowedRate)
// hopsToCongestion = 255;
// But it is unnecessary since it has no effect upon the rate at which
// the station may transmit.


//9.7.3.7 localFairRate, normLocalFairRate
// dummy function call
boolean RTT_worth_of_agingIntervals_passed_since_last_update;
RTT_worth_of_agingIntervals_passed_since_last_update = A0();

//active weight

if (aggressive)
{
    if (localCongested)
        localFairRate = lpAddRate;
    else // not congested
        localFairRate = unreservedRate;
}
else // conservative
    {
    if (localCongested)
        if (just_entered_congested) // first time when congested
            localFairRate = unreservedRate / activeWeights;
        else // was congested last time
        {
            if ((addRate + fwRate < rateLowThreshold)
            && ( RTT_worth_of_agingIntervals_passed_since_last_update ))
            { // min (unreservedRate,(localFairRate +
              // (unreservedRate - localFairRate) / RAMPCOEF))
            if ( unreservedRate > (localFairRate + (unreservedRate - localFairRate) / RAMP-
COEF))
            localFairRate = (localFairRate + (unreservedRate - localFairRate) / RAMPCOEF);
            else
            localFairRate = unreservedRate;
            if (localFairRate >= unreservedRate)
            localCongested = FALSE;
            }
            else if ( (addRate + fwRate > rateHighThreshold)
            && ( RTT_worth_of_agingIntervals_passed_since_last_update ))
```

```
            localFairRate = localFairRate - localFairRate / RAMPCOEF;
            else
            localFairRate = localFairRate; // no change
        }
    else // not congested
        localFairRate = unreservedRate;// INIT
    }
normLocalFairRate = localFairRate / NORMCOEF;// INIT


// active weight calculation

// 9.7.4 Per advertisementInterval
// D1.2 December 2002

if (rcvdFairRate < normLocalFairRate) // downstream is more congested
{
    // upstream contributing to downstream congestion
    if ( ((rcvdRInotMatch) && (rcvdFairRate < normLpFwRate )) ||
        ((rcvdRIMatch) && (rcvdFairRate < normLpFwRateCongested )) )
        advertisedFairRate = rcvdFairRate;
}
else if (localCongested) // local station is congested (and more than downstream)
    advertisedFairRate = normLocalFairRate;
else // no downstream or local congestion
    advertisedFairRate = FULL_RATE;


return 0;
}
```

10. Topology discovery

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None

Abbreviations:
None.

Revision History:

Draft 0.3, June 2002	Initial draft document for RPR WG review. Editorial notes added for clarification.
Draft 1.0, August 2002	Addressed most comments from draft 0.3. Those not yet addressed are specifically mentioned in editor's notes. The protection ad-hoc (PAH) is meeting to address the list of issues described in the editorial note immediately following this one. Major modifications made in D1.0 include: Addition of layer diagram Clarifications in topology database and isolated station subclauses Reorganization and additions to topology message subclause
Draft 1.1, October 2002	Addressed most comments from draft 1.0. Those not yet addressed are specifically mentioned in editor's notes.The protection ad-hoc (PAH) is meeting to address the list of issues described in the editorial note immediately following this one.
Draft 2.0, December 2002	Addressed most comments from draft 1.1. Those not yet addressed are specifically mentioned in editor's notes.The protection ad-hoc (PAH) is meeting to address the list of issues described in the editorial note immediately following this one.

Editors' Notes: *To be removed prior to final publication.*

The protection ad-hoc (PAH) is meeting to address the following list of issues:

1. Comment #254, November: Determine if topology can include a mechanism for dynamically determining bandwidth reservations.
Status: The PAH has not yet considered this.

2. Determine text for receive buffer size statement as described in editorial note at the end of this clause.

3. Comments #252 and #239, November: Generate and review flow chart and/or state machine description for modification and validation of the topology database, including duplicate MAC address checks.

4. Determine how to handle passthrough mode, including how to achieve fast topology discovery for passthrough when there are topology messages left on the ring from stations that have disappeared from the ring is an issue that must be resolved.

5. Finalize the approach for how topology messages can be forwarded through spans containing one failed link for steering rings. An approach has been discussed to have a bit used to mark frames that have gone through a half-alive span to facilitate determination at the receiver of stations that are data reachable on a given ringlet.

The protection ad-hoc (PAH) has resolved the following list of issues:

1. Comment #528, July: Provide topology state machine.
Status: Completed. Kshitij Kumar provided the state machine.

2. Comment #529, July: Provide scenarios illustrating whether there are topology discovery related requirements for the initiation of wrapping and/or steering protection.
Status: Completed. The initiation of wrapping and/or steering protection does not require that topology discovery has completed, in the event, for example, that a protection event overlaps a period during which new stations have just been added to the ring. The determination of a preferred direction on the ring for steering protection at each station is done only for stations contained in the topology database. The determination of whether to wrap at a station is not dependent on the contents of the topology database.

3. Comment #535, July: Determine whether the topology module needs to convey a topology change indication along with the corresponding old and new neighbor information so that protection switching can determine whether a link coming up should be subject to wait to restore (WTR), or whether this can be handled by the protection protocol.
Status: Completed. The topology module will store former neighbor MAC information so that protection switching can determine whether a link coming up should be subject to WTR.

4. Comment #538, July: Determine the criteria for topology consistency checking, the algorithm used for the checking, and the intent for the checking (to inform the operator via events).
Status: Completed. Jim Kao provided the criteria for topology consistency checking. Algorithms for performing this checking may be added for informative purposes to either the clause or to an annex.

5. Comments #541, #557, and #558, July: Define the usage of the downstream link reserved subclassA0 bandwidth, how it is computed, and its units. The PAH, in conjunction with the RAH, also needs to define how each station gets the necessary information to compute this bandwidth.
Status: Completed. The units for the bandwidth and how it is reported (via the topology message) have been defined.

6. Comment #560, July: Clarify the interworking of topology and protection. It will become clear through this exercise what the actual list of triggers is for generation of topology messages.
Status: Completed. Additional modifications will come about through commenting.

Editors' Notes: *To be removed prior to final publication.*

7. Comment #562, July: Modify 10.2.2 to illustrate the key scenarios for topology discovery. These include (for example) station initialization, addition of a station, removal of a station, isolation of a station, handling of bandwidth allocation, and handling of misconfigurations. Description will be included as to how topology interacts with protection and fairness. The topology database description in 10.2.6 will be reorganized to apply to the different scenarios. The topology discovery message description in 10.2.4 will also be reorganized to apply to the different scenarios.
Status: Completed. Additional modifications will come about through commenting.

8. Comment #463, September: Combine fast topology messages with protection messages.
Status: Completed. The messages have been combined.

10.1 Scope

This clause describes the RPR topology discovery protocol, which implements a reliable and accurate means for all RPR stations on a ring to discover the initial topology of the stations on the ring and any changes to that topology. The protocol is intended to be very scalable, to cause insignificant overhead for ring traffic, and to cause insignificant overhead on software and ASICs. The protocol resides in the MAC control sublayer, as shown in the shaded region of Figure 10.1.

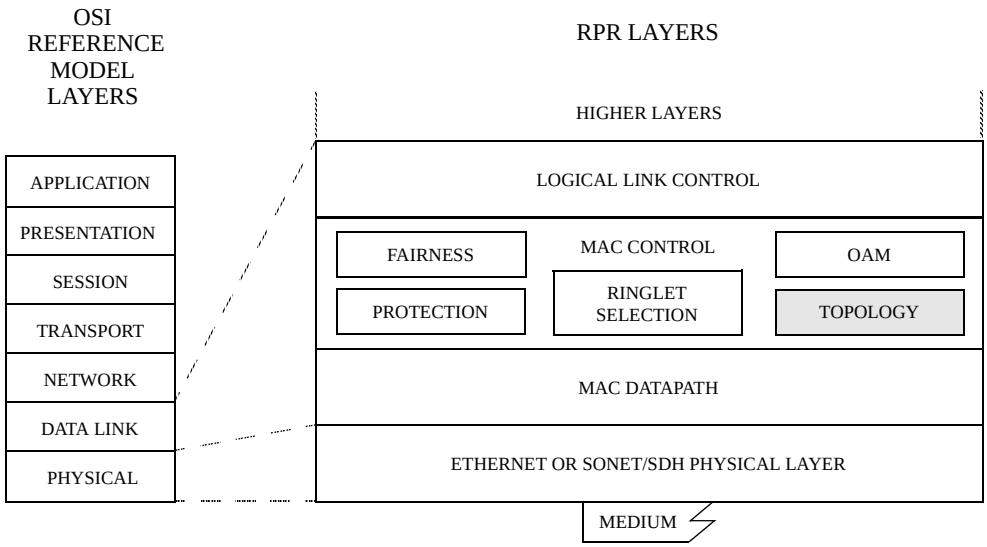


Figure 10.1—RPR layer diagram

The services and features provided are:

a) Determine/validate connectivity and ordering of stations on the ring.
b) Ensure all stations on the ring will converge to a uniform and current image of the topology normally within 1 ringlet circulation time.
c) Tolerant of message loss.
d) Operate without any master station on the ring.
e) Operate independently of and in the absence of any management systems.
f) Usable with all supported topologies: ring, bus (broken ring), and isolated station.

g) Support dynamic addition and removal of stations to/from the ring.
h) Detect mis-cabling between stations.
i) Provide means of sharing additional information between stations.
j) Cause insignificant overhead.

Editors' Notes: *To be removed prior to final publication.*

Determination of connectivity and ordering of stations on the ring includes the detection of physical connectivity on "half-alive" spans. It is an objective of topology discovery, for example, that station F in the partial string A=B=C->D=E=F be able to provide a user performing diagnostics at F with the complete view of the physical connectivity of the string.

Detecting mis-cabling (above) assumes static local assignment of ring IDs (as opposed to dynamic discovery). How the assignments are made is a local, non-standardized decision. This might need to be discussed in the OAM clause.

The RPR topology discovery protocol is used to discover the physical link configuration between stations. It is not within the scope of the RPR topology discovery protocol to determine the dynamic link status information, i.e. which ringlet links are up or down, ring segment failures, etc. The discovered topology is used by other protocols such as the RPR protection protocol and the RPR congestion avoidance protocol.

10.2 Algorithm overview

The RPR topology discovery protocol provides each station on the ring with knowledge of the number and arrangement of other stations on the ring. This collection of information is referred to as the topology image. Each station maintains its own local copy of the topology image for the entire ring in its topology database. Initially, the station's topology image contains information only about itself.

Ring topology discovery is initiated as needed and periodically. No station acts as a master for the topology image or for the protocol. All ringlet segments that can be discovered are included. A fully connected ring is not needed for the protocol.

The information required to create the basic topology image (including, for example, hop counts per ringlet from the local station to all other stations on the ring) is derivable from the protection message. The protection message is used for this purpose because it is sent at a fast (substantially sub-50 millisecond) rate, as well on triggers. A specification of the triggers for the initiation of topology discovery is provided in 11.7.1, along with a specification of the protection message. 11.3 also contains description of the complete processing flow for received protection messages.

The complete topology image contains more information than just station identifiers and physical connectivity relationships. It also contains station capabilities information for use in various parts of this standard, and may contain optional vendor-specific information. This information, contained in the topology message, is not as time-critical as that contained in the protection message.

This clause contains specification of the modification and verification of the topology image upon receipt of topology or protection messages. It also contains specification of the topology message.

The messages sent as part of the RPR topology discovery protocol are indicated in the RPR frame header as control frames.

10.2.1 At initialization

At station initialization, the local topology image is initialized to contain only the local station and no links, and the addresses of the neighboring stations are initialized to all 0's. The station starts the topology algo-

rithm by broadcasting a protection message on all ringlets. Then it continually listens for protection messages broadcast on the ring, and broadcasts protection messages periodically and upon other triggers defined in Clause 11. On initialization the station is also triggered to broadcast a topology message on all ringlets. After this initial trigger, the station broadcasts the topology message periodically, as defined in 10.5.1.

10.2.2 At addition of a station

When a station is inserted into an existing ring, it initializes itself as described above. Its neighbors will detect the station as a new neighbor by receiving its protection message with a TTL set to MAX_STATIONS (indicating the message has traveled exactly one hop) and with the SA set to a value other than that previously stored (if any).

The neighbors will respond by sending an immediate protection message to the new station. All other stations will detect the new station by receiving its protection message, and each will send an immediate protection message to the new station. The information available from the protection message includes SA, the number of hops away the station is on the ringlet on which the message is received (derived from TTL), and protection-related information described in 11.6.

All stations on the ring will send a topology message to the new station based on the periodic timer at each local station. The topology message is partitioned into Type-Length-Value (TLV) entries, as defined in 10.4.1.

10.2.3 At span failure

When a span fails, the stations detecting the failure store the knowledge of their neighbor station address on the failed interface and clear their current knowledge of their neighbor station address on that interface. The stations send protection messages that report exactly which links failed to the other stations on the ring.

10.2.4 At removal of a station

When a station is removed from a ring, the stations detecting the removal of the station take the same actions as described in 10.2.3 in the event of detection of signal fail on incoming interfaces.

Editors' Notes: *To be removed prior to final publication.*

Passthrough mode can be handled by topology discovery through the inclusion of a trigger bit in protection messages. The trigger bit would be set based on the detection of a passthrough trigger such as a change in neighbor identity for TTL==MAX_STATIONS indicating that the immediate neighbor of a station has changed. All other stations on the ring would respond to the trigger bit by sending copies of their existing protection messages without the trigger bit set. This approach enables topology to be quickly rediscovered during passthrough events.

How to achieve fast topology discovery for passthrough when there are topology messages left on the ring from stations that have disappeared from the ring is an issue that must be resolved.

10.2.5 Topology discovery

Upon triggers defined in 11.7.1, a station broadcasts a protection message to all stations on both ringlets. The protection message contains information about the local station relevant to connectivity, including its links to its neighbors. When a station receives a protection message, it updates its local topology image.

The protection message contains the ringlet ID on which it is sent. When a station receives a protection message with a TTL set to MAX_STATIONS (indicating the message has traveled exactly one hop), it verifies that the ringlets of both stations are connected with the same ringlet ID value (as indicated in the received message and by local configuration).

The protection message, like all RPR frames, contains the source MAC address of the station from which it is sent as part of the RPR header. When a station receives a protection message with a TTL set to MAX_STATIONS (indicating the message has traveled exactly one hop), it verifies that it knows who its neighbor is.

Upon triggers defined in 10.5.1, a station broadcasts a topology message to all stations on both ringlets. When a station receives a topology message, it also updates its local topology image.

10.2.6 Topology database and hop count determination

Under certain conditions such as a failed span in a bidirectional ring, a station will only be able to receive topology and protection messages from the other stations on the ring on one ringlet. It is a requirement to be able to construct a complete topology database reflecting connectivity information on both ringlets. Hence, a single topology database showing information for both ringlets can meet this requirement as well as facilitate maintenance of a consistent database.

The hop count to another station can be calculated through examination of the topology image. For each ringlet, the topology image must be completed up to the station whose hop count is desired.

The values of each entry are: the downstream hop count on ringlet0, the downstream hop count on ringlet1, the local MAC address, the MAC address of the neighboring station connected to the west span of the station, the MAC address of the neighboring station connected to the east span of the station, the protection status of the receive link from the west station, the protection status of the receive link from the east station, the wrap status of the station's west span, the wrap status of the station's east span, the reserved subclassA0 bandwidth on the transmit link to the west station, the reserved subclassA0 bandwidth on the transmit link to the east station, data reachability downstream on ringlet0, and data reachability downstream on ringlet1. The receive link's availability can be any of the protection message request values given in Table 11.4. The reserved subclassA0 bandwidth is described in .

The topology database also contains the MAC address information of the neighbor previously connected to each interface of the local station. This information is only relevant to the local station, and can be stored separately from the information shown in Table 10.1. This information is stored in the topology database because it is required by the protection protocol. The topology module will notify the protection module when this information has changed. The former neighbor entries will be rewritten upon a change in the value of the current neighbor entry for each respective interface.

Editors' Notes: *To be removed prior to final publication.*

Topology database organization will likely be shown based on information directly received, e.g. upstream hop counts per ringlet. Other clauses utilizing the information will derive information from this, e.g. downstream hop count on ringlet0 can be easily derived from upstream hop count on ringlet1.

An example table for one ringlet is shown in Table 10.1.

Table 10.1—Topology and status database example

hop count for ringlet0	hop count for ringlet1	local MAC	west neighbor's MAC	east neighbor's MAC	west receive link availability	east receive link availability	west span wrap status	east span wrap status	station capabilities	west transmit link reserved subclass A0 bandwidth	east transmit link reserved subclass A0 bandwidth	data reachability, ringlet0	data reachability, ringlet1
0	0	00-10-A4-97-A8-DE	00-10-A4-97-A8-EF	00-10-A4-97-A8-BD	IDLE	IDLE	I	I	JC=0 WC=1	50	40	Yes	Yes
1	3	00-10-A4-97-A8-EF	00-10-A4-97-A8-AC	00-10-A4-97-A8-DE	IDLE	IDLE	I	I	JC=1 WC=1	100	80	Yes	Yes
2	2	00-10-A4-97-A8-AC	00-10-A4-97-A8-BD	00-10-A4-97-A8-EF	IDLE	IDLE	I	I	JC=1 WC=1	150	120	Yes	Yes
3	1	00-10-A4-97-A8-BD	00-10-A4-97-A8-DE	00-10-A4-97-A8-AC	IDLE	IDLE	I	I	JC=1 WC=0	200	160	Yes	Yes

10.2.6.1 Modification of the topology database

Editors' Notes: *To be removed prior to final publication.*

A flow chart or other more detailed description of the rules for this section is needed.

The topology database is modified upon receipt of topology or protection messages that result in a change in information contained in the database. Protection messages may report a change in the address of a station a given number of hops away on a given ringlet, a change in link status, a change in wrap status, and wrap capability information. Topology messages may report a change in the address of a neighboring station of a station, a change in reserved subclassA0 bandwidth reported by that station, a change in station capabilities (such as whether it is wrap or jumbo frame capable), or a change in other optional vendor-specific information. Changes in reserved subclassA0 bandwidth or in station capabilities by themselves result only in modification of single entries within the database.

Changes in the address of a station a given number of hops away or in the address of a neighboring station of a station a given number of hops away force, for the sake of consistency, the deletion of all entries in the database corresponding to stations on the ringlet beyond the point of the change. For example, if a station 2 hops away reports a signal fail on receive of ringlet0, this implies that stations further upstream from that station on ringlet0 are also no longer connected to the ringlet. Therefore, the hop count entries for ringlet1 downstream of values 3 or greater are cleared. If multiple failures occur on a ring such that both the hop count values for ringlet0 and ringlet1 are cleared for a given station, then that station is removed from the topology database.

Editors' Notes: *To be removed prior to final publication.*

Passthrough mode can be handled by topology discovery in an a similar fashion to what is described above. If a station 2 hops away reports a new neighbor, this implies that stations further away than that station on the given ringlet currently stored in the topology database must be removed. Stations upstream must then reconfirm their connection to the ringlet so that they can be added back to the topology database.

Upon detection of signal fail on a given interface (west for example), the west neighbor MAC address is cleared. Upon detection of a new neighbor on that interface, the MAC address of the new neighbor is first compared to the former west neighbor MAC address. The protection module is notified if the new neighbor and former neighbor MAC addresses are different. The new neighbor MAC address is then copied into both the west neighbor MAC address and the former west neighbor MAC address.

10.2.7 Data/control reachability

Editors' Notes: *To be removed prior to final publication.*

The PAH needs to discuss whether topology messages should be forwarded through spans containing one failed link. An approach has been discussed to have a bit used to mark frames that have gone through a half-alive span to facilitate determination at the receiver of stations that are data reachable on a given ringlet.
Determination that a station is data reachable needs to be tied into rules on admitting data or not into the ringlet selection module.

A station S2 is considered data reachable from a station S1 if a path on the ring can be found from S1 to S2 (and thus from S2 to S1) such that no span on the path has a protection status of forced switch (FS) or signal fail (SF). A span has a protection status of FS or SF if either link on the span has a link protection status of FS or SF. A station S2 is considered data reachable from a station S1 on a given ringlet when the criteria above is met for the path from S1 to S2 on that ringlet. If S2 is data reachable from S1 on ringlet0, for example, then it is ensured that S1 is data reachable from S2 on ringlet1 in a correctly connected ring.

A station S1 is considered control reachable from another station S2 when topology frames can be received at station S1 from station S2. Therefore, S1 is control reachable from any station that appears in its topology database. Also, any data reachable station is implicitly control reachable, but not vice versa. It is possible that a station can be control reachable from another station, but not vice versa. If a station S2 appears in the topology database of station S1, then S1 is control reachable from S2.

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 at the same time enabling the complete physical connectivity of the ring topology to be shown. It is beneficial for the complete physical connectivity of the ring to be locally available at each station for field diagnostic purposes.

When a station is not data reachable or control reachable from any other station, it determines itself to be isolated. This will occur if the receive links on both the east and west interfaces of a station have a protection state of signal fail (SF). An isolated station shall remove all entries from its local topology image, but shall

transfer its own east neighbor and west neighbor information to the former east neighbor and former west neighbor fields in the database before clearing the east neighbor and west neighbor fields. The information in the former east neighbor and former west neighbor fields is needed to determine whether a link should be brought up immediately, as described in 11.3.3.

10.2.8 Topology consistency check

It can be easily determined when an image is complete and consistent by examining the image contents. When the contents of the local topology image show station information for each station described in the link information of another station, then the image is complete. For each ringlet, when the contents of the local topology image show that all stations on that ringlet are connected to each other in a logical ring, bus (broken ring), or isolated station, then the topology image is consistent.

A canonical form for the topology image allows all the stations to eventually arrive at the same image for the topology.

Editors' Notes: *To be removed prior to final publication.*

Algorithms for performing this checking may be added for informational purposes to either the clause or to an annex.

10.2.8.1 Determination and validation of ringlet ID

Each station determines which interface is associated with which ringlet and assigns the corresponding ringlet_id either through fixed mapping between hardware locations or through configuration. Each protection message is sent separately on each ringlet, identifying the ringlet on which it is being sent. Any protection message with a TTL set to MAX_STATIONS received on a ringlet different from the ringlet on which it is identified as being sent shall cause the link to be declared non-operational and trigger a mis-configuration alarm. This will result in alarm-triggering events being generated only from stations at which the physical misconfiguration has occurred. That shall trigger a signal fail for the link in protection. When a matched ringlet ID topology message from its neighbor is received, that shall clear the signal fail in protection.

10.2.8.2 Neighbor inconsistency

The following will trigger a neighbor inconsistency alarm or indication:

a) East station's west neighbor MAC address is not equal to local station MAC address, or
b) West station's east neighbor MAC address is not equal to local station MAC address.

The zero MAC address is an exceptional case for this inconsistency check. The inconsistency check shall be performed when the neighbor information is updated due to a new received protection message. An alarm will be triggered to operator if this neighbor inconsistency is detected.

10.2.8.3 Duplicate MAC address

A MAC address is the unique identification for a station on an RPR ring. If a station receives a protection message from another station with the same MAC address as its MAC address, an alarm will be triggered to the operator for this duplicate MAC address. An alarm will also be triggered to the operator if a station detects multiple stations anywhere on the ring with the same MAC address.

10.2.8.4 The number of stations exceed MAX_STATIONS

If the total number of entries in the topology database exceeds MAX_STATIONS, an alarm or indication shall be triggered to the operator. This alarm can be triggered under two conditions. In a ring configuration,

the number of stations from which topology messages are received may equal (MAX_STATIONS + 1). In a string configuration, the number of stations from which topology messages are received can significantly exceed MAX_STATIONS.

10.3 Topology discovery process

10.3.1 Topology discovery process description

Editors' Notes: *To be removed prior to final publication.*

This section will be removed from the next version of the draft, as per comment #252 from November. The appropriate additional flow chart and/or state machine description needs to be added into 11.7.1 before this subclause is deleted. Because of this, please do not comment on this subclause. In addition, the following resolved comments from D1.1 have not been implemented in the state machine below, but will be followed in the state machine description to be provided by the PAH.

See #248, #250, #251. #243 is covered in 10.4 and 10.5.

An exact definition of the topology discovery process is specified in Table 10.2, where rows are evaluated in top-to-bottom order.

Table 10.2—Topology discovery state table

Current State		Row	Next State	
State	**Conditions**		**Action**	**State**
Dormant (Waiting for initiation)	System receives a wakeup signal (or any other event)	1	Start Topology Timer. Clear Topology Database	Init
Init	Incoming TD message on ringlet0 (TTL == MAX_TTL)	2	Insert West Neighbor information into Topology Database: Source MAC, station capabilities, and with ringlet0 hop count = 1. Send TD Message on both Ringlets, East Neighbor MAC = 0, West Neighbor MAC as received. Initiate Fast Topology Timer	WNHF
	Incoming TD message on ringlet1 (TTL == MAX_TTL)	3	Insert East Neighbor information into Topology Database: Source MAC, station capabilities, and with ringlet1 hop count = 1. Send TD Message on both Ringlets, West Neighbor MAC = 0, East Neighbor MAC as received Initiate Fast Topology Timer	ENHF

Table 10.2—Topology discovery state table *(continued)*

	Incoming TD message on ringlet0 (TTL != MAX_TTL) and source node MAC IS NOT in DB	4	Insert entry for this node in DB, with hop count = (MAX_TTL minus TTL) for ringlet1. Send TD Message on both Ringlets, Neighbor MACs = 0, Initiate Fast Topology Timer	Init
	Incoming TD message on ringlet1 (TTL != MAX_TTL) and source node MAC IS NOT in DB	5	Insert entry for this node in DB, with hop count = (MAX_TTL minus TTL) for ringlet0. Send TD Message on both Ringlets, Neighbor MACs = 0, Initiate Fast Topology Timer	Init
	Incoming TD message on ringlet0 (TTL != MAX_TTL) and source node MAC EXISTS in DB	6	If Station Capabilities have changed, Update DB with entry for this node hop count = (MAX_TTL minus TTL) on ringlet1, Initiate Fast Topology Timer? Else - No Action.	Init
	Incoming TD message on ringlet1 (TTL != MAX_TTL) and source node MAC EXISTS in DB	7	If Station Capabilities have changed, Update DB with entry for this node, hop count = (MAX_TTL minus TTL) on ringlet0, Initiate Fast Topology Timer? Else – No Action.	Init
	Fast Topology Timer Expired	8	Send TD Message on both Ringlets, Neighbor MACs = 0, If Timer has expired 8 times, start Slow Topology Timer, Else restart Fast Topology Timer	Init
	Slow Topology Timer Expired	9	Send TD Message on both Ringlets, Neighbor MACs = 0, Start Slow Topology Timer	Init
	Locally detected Signal Fail Indication on either Ringlet	10	Send TD Message on both Ringlets, Neighbor MACs = 0, Initiate Fast Topology Timer	Init
	Incoming TD message with Invalid Station Capabilities or Ringlet ID parameter	11	Send Mis-configuration Notification	Init
WNHF (West Neighbor heard from)	Incoming TD message on ringlet0 (TTL = MAX_TTL), and MAC SA or Station Capabilities have not changed	12	No Action	WNHF

Table 10.2—Topology discovery state table *(continued)*

Incoming TD message on ringlet0 (TTL = MAX_TTL), where MAC SA or Station Capabilities have changed	13	Send Neighbor Changed Notification to Operator, Update Topology DB with new neighbor SA or capabilities. Send TD message on both Ringlets with new neighbor information. Initiate Fast Topology Timer.	WNHF
Incoming TD message on ringlet1 (TTL = MAX_TTL)	14	Update Database with East Neighbor Send TD message on both Ringlets with new East Neighbor address as well as West Neighbor address. Initiate Fast Topology Timer.	BNHF
Incoming TD message on ringlet0 (TTL != MAX_TTL) where source node MAC EXISTS in DB	15	If Station Capabilities have changed, Update DB with entry for this node hop count = (MAX_TTL minus TTL) on ringlet1. Send TD Message on both Ringlets, Initiate Fast Topology Timer? Else - No Action.	WNHF
Incoming TD message on ringlet1 (TTL != MAX_TTL) where source node MAC EXISTS in DB	16	If Station Capabilities have changed, Update DB with entry for this node hop count = (MAX_TTL minus TTL) on ringlet0. Send TD Message on both Ringlets, Initiate Fast Topology Timer? Else - No Action.	WNHF
Incoming TD message on ringlet0 (TTL != MAX_TTL) where source node MAC does NOT exist in DB	17	Insert entry for this node in DB, with hop count = (MAX_TTL minus TTL) for ringlet1. Send TD Message on both Ringlets, Initiate Fast Topology Timer	WNHF
Incoming TD message on ringlet1 (TTL != MAX_TTL) where source node MAC does NOT exist in DB	18	Insert entry for this node in DB, with hop count = (MAX_TTL minus TTL) for ringlet0. Send TD Message on both Ringlets, Initiate Fast Topology Timer	WNHF
Fast Topology Timer Expired	19	Send TD Message on both Ringlets, If Timer has expired 8 times, start Slow Topology Timer, Else restart Fast Topology Timer	WNHF
Slow Topology Timer Expired	20	Send TD Message on both Ringlets. Restart Slow Topology Timer	WNHF

Table 10.2—Topology discovery state table *(continued)*

	Locally detected Signal Fail Indication, ringlet0 unavailable	21	Send TD Message on both Ringlets, with West Neighbor's MAC = 0, East Neighbor's MAC = 0. Initiate Fast Topology Timer	Init
	Locally detected Signal Fail Indication, ringlet1 unavailable	22	No action	WNHF
	Incoming TD message with Invalid Station Capabilities or Ringlet ID on ringlet0	23	Send Mis-configuration Notification	Init
	Incoming TD message with Invalid Station Capabilities or Ringlet ID on ringlet1	24	Send Mis-configuration Notification	WNHF
ENHF (East Neighbor heard from)	Incoming TD message on ringlet0 (TTL = MAX_TTL), and MAC SA or Station Capabilities have not changed	25	No Action	ENHF
	Incoming TD message on ringlet1 (TTL = MAX_TTL), where MAC SA or Station Capabilities have changed	26	Send Neighbor Changed Notification to Operator, Update Topology DB with new neighbor SA or capabilities. Send TD message on both Ringlets with new neighbor information. Initiate Fast Topology Timer.	ENHF
	Incoming TD message on ringlet0 (TTL = MAX_TTL)	27	Update Database with West Neighbor Send TD message on both Ringlets with new East Neighbor address as well as West Neighbor address. Initiate Fast Topology Timer.	BNHF
	Incoming TD message on ringlet0 (TTL != MAX_TTL) where source node MAC EXISTS in DB	28	If Station Capabilities have changed, Update DB with entry for this node hop count = (MAX_TTL minus TTL) on ringlet1. Send TD Message on both Ringlets, Initiate Fast Topology Timer? Else - No Action.	ENHF
	Incoming TD message on ringlet1 (TTL != MAX_TTL) where source node MAC EXISTS in DB	29	If Station Capabilities have changed, Update DB with entry for this node hop count = (MAX_TTL minus TTL) on ringlet0. Send TD Message on both Ringlets, Initiate Fast Topology Timer? Else - No Action.	ENHF

Table 10.2—Topology discovery state table *(continued)*

	Incoming TD message on ringlet1 (TTL != MAX_TTL) where source node MAC does NOT exist in DB	30	Insert entry for this node in DB, with hop count = (MAX_TTL minus TTL) for ringlet0. Send TD Message on both Ringlets, Initiate Fast Topology Timer	ENHF
	Fast Topology Timer Expired	31	Send TD Message on both Ringlets, If Timer has expired 8 times, start Slow Topology Timer, Else restart Fast Topology Timer	ENHF
	Slow Topology Timer Expired	32	Send TD Message on both Ringlets. Restart Slow Topology Timer	ENHF
	Locally detected Signal Fail Indication, ringlet1 unavailable	33	Send TD Message on both Ringlets, with West Neighbor's MAC = 0, East Neighbor's MAC = 0. Initiate Fast Topology Timer	Init
	Locally detected Signal Fail Indication, ringlet0 unavailable	34	No action	ENHF
	Incoming TD message with Invalid Station Capabilities or Ringlet ID on ringlet1	35	Send Mis-configuration Notification	Init
	Incoming TD message with Invalid Station Capabilities or Ringlet ID on ringlet0	36	Send Mis-configuration Notification	ENHF
BNHF (Both Neighbors heard from)	Incoming TD message on ringlet0 (TTL = MAX_TTL), where MAC SA or station capabilities have not changed	37	No action	BNHF
	Incoming TD message on ringlet1 (TTL = MAX_TTL), where MAC SA or station capabilities have not changed	38	No action	BNHF
	Incoming TD message on ringlet1 or 0 (TTL = MAX_TTL), where MAC SA or Station Properties have changed	39	Send Neighbor Changed Notification to Operator Update Topology DB with new neighbour. Send TD message on both Ringlets with new neighbor information. Initiate Short Topology Timer.	BNHF
	Incoming TD message on ringlet0 (TTL != MAX_TTL) where source node MAC EXISTS in DB	40	If Station Capabilities have changed, Update DB with entry for this node hop count = (MAX_TTL minus TTL) on ringlet1. Send TD Message on both Ringlets, Initiate Fast Topology Timer? Else - No Action.	BNHF

Table 10.2—Topology discovery state table *(continued)*

Incoming TD message on ringlet1 (TTL != MAX_TTL) where source node MAC EXISTS in DB	41	If Station Capabilities have changed, Update DB with entry for this node hop count = (MAX_TTL minus TTL) on ringlet0. Send TD Message on both Ringlets, Initiate Fast Topology Timer? Else - No Action.	BNHF
Incoming TD message on ringlet1 (TTL != MAX_TTL) where source node MAC does NOT exist in DB	42	Insert entry for this node in DB, with hop count = (MAX_TTL minus TTL) for ringlet0. Send TD Message on both Ringlets, Initiate Fast Topology Timer	BNHF
Incoming TD message on ringlet0 (TTL != MAX_TTL) where source node MAC does NOT exist in DB	43	Insert entry for this node in DB, with hop count = (MAX_TTL minus TTL) for ringlet1. Send TD Message on both Ringlets, Initiate Fast Topology Timer	BNHF
Fast Topology Timer Expired	44	Send TD Message on both Ringlets, If Timer has expired 8 times, start Slow Topology Timer, Else restart Fast Topology Timer	BNHF
Slow Topology Timer Expired	45	Send TD Message on both Ringlets. Restart Slow Topology Timer	BNHF
Locally detected Signal Fail Indication, ringlet1 unavailable	46	Send TD Message on both Ringlets, with West Neighbor's MAC = 0, East Neighbor's MAC = 0. Initiate Fast Topology Timer	WNHF
Locally detected Signal Fail Indication, ringlet0 unavailable	47	Send TD Message on both Ringlets, with West Neighbor's MAC = 0, East Neighbor's MAC = 0. Initiate Fast Topology Timer	ENHF
Incoming TD message with Invalid Station Capabilities or Ringlet ID on ringlet0	48	Send Mis-configuration Notification	ENHF
Incoming TD message with Invalid Station Capabilities or Ringlet ID on ringlet1	49	Send Mis-configuration Notification	WNHF

10.4 Topology message format

The topology message is a variable length message. The topology message is not used for discovery of the physical topology of the ring, but rather is used to convey additional information that needs to be reported by a station to the rest of the ring on a slower timeframe, such as reserved bandwidth configuration information.

The topology message is used for reporting changes in station status that are not as time critical as those reported in the protection message. The topology message is sent as a MAC Control message with a control type value of 1. It is sent as a broadcast frame on all ringlets, with a TTL of MAX_STATIONS, removed by the source station, and with the source MAC set to the actual MAC of the sending station.

The topology message frame format is outlined in Figure 10.2.

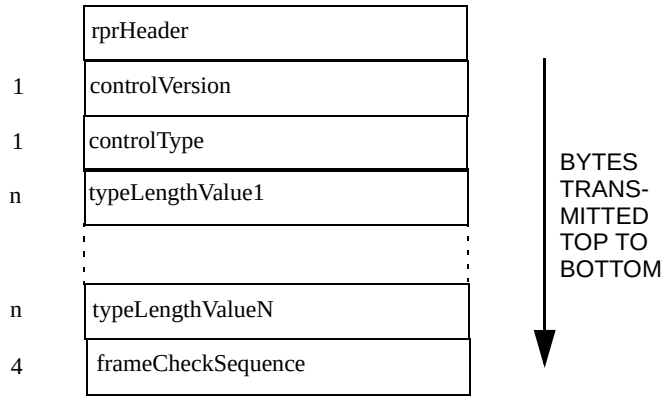


Figure 10.2—topology message frame format

The *rprHeader* field corresponds to the RPR header of the RPR control frame defined in Clause 8. The *controlVersion* and *controlType* fields correspond to the fields with those names that are part of the RPR payload of the RPR control frame. The topology specific fields comprise the control PDU portion of the RPR payload, and are detailed below.

10.4.1 TLV entries

A Type-Length-Value (TLV) encoding scheme is used to encode additional station information in topology messages. The TLVs may appear in the topology frame in any order.

A TLV is encoded as a 2 byte *tlvType* field, followed by a 2 byte *tlvLength* field, followed by a variable length *tlvValue* field. The general format of a TLV is shown in Figure 10.3.

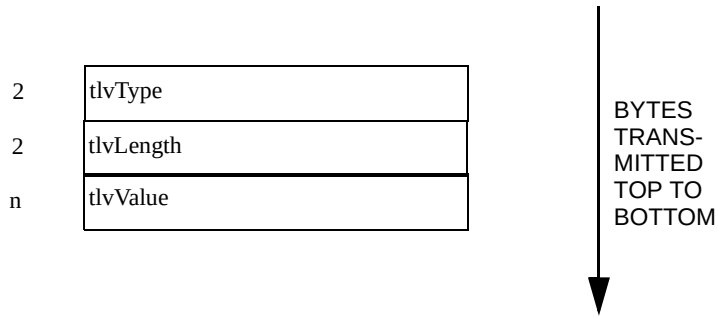


Figure 10.3—TLV format

10.4.1.1 tlvType

The *tlvType* field consists of 1 bit to specify if the TLV if mandatory, followed by 5 reserved bits, followed by 10 bits to specify a type.

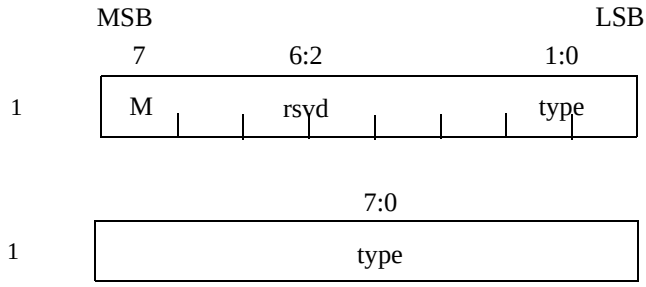


Figure 10.4—TLV type format

10.4.1.1.1 Mandatory (M) bit

The M bit indicates whether the receiving station may ignore the TLV, or whether it must process it.

The values of M are defined in Table 10.4.

Table 10.3—Mandatory bit (M) values

Value	Name	Description
0	NOTMANDA-TORY	Processing of the following TLV is optional
1	MANDATORY	Processing of the following TLV is mandatory

10.4.1.1.2 rsvd

The *rsvd* field is 5 bits reserved for future use.

10.4.1.1.3 type

The *type* field encodes how the *tlvValue* field is to be interpreted. The type field is 10 bits.

10.4.1.2 tlvLength

The *tlvLength* field consists of 6 reserved bits, followed by 10 bits to specify a length.

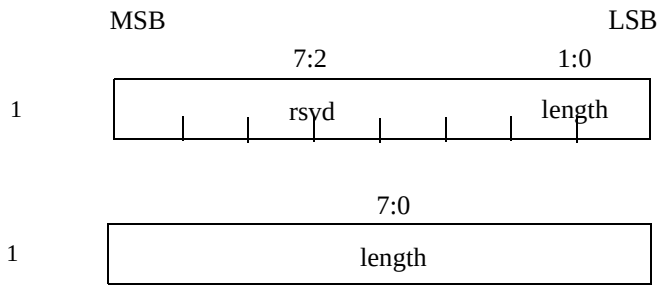


Figure 10.5—TLV length format

10.4.1.2.1 rsvd

The *rsvd* field is 6 bits reserved for future use.

10.4.1.2.2 length

The *length* field specifies the length of the Value field in bytes. The length of any tlvValue field shall not exceed 1024 bytes.

10.4.1.3 tlvValue

The *tlvValue* field is a byte string, of length bytes, as specified in the length field, that encodes information to be interpreted as specified by the type field.

10.4.2 Defined TLV encodings

The following subclauses define the TLV encodings supported by this standard. Any additional TLV encodings used are beyond the scope of this standard.

10.4.2.1 Weight TLV

The Weight TLV encodes the station weight in each ringlet, and is mandatory.

10.4.2.1.1 M

The M bit is set to 1.

10.4.2.1.2 tlvType

The *type* field within *tlvType* is set to 0000000001_2.

10.4.2.1.3 tlvLength

The *length* field within *tlvLength* is set to 0000000010_2.

10.4.2.1.4 tlvValue

The format for the *tlvValue* field of the Weight TLV is shown in Figure 10.6.

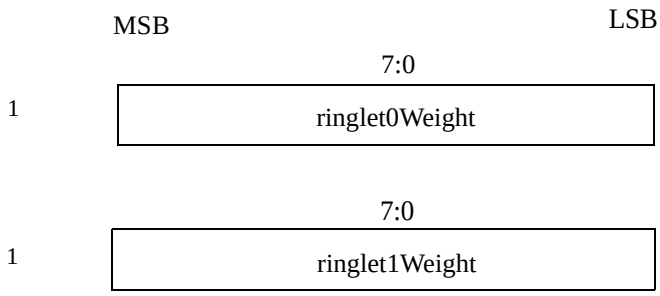


Figure 10.6—Weight TLV tlvValue format

The *ringlet0Weight* field carries the weight of the station on ringlet0.

The *ringlet1Weight* field carries the weight of the station on ringlet1.

10.4.2.2 Total reserved bandwidth TLV

The total reserved bandwidth TLV encodes the total station reserved subclassA0 bandwidth in each ringlet, and is mandatory.

10.4.2.2.1 M

The M bit is set to 1.

10.4.2.2.2 tlvType

The *type* field within *tlvType* is set to 0000000010_2.

10.4.2.2.3 tlvLength

The *length* field within *tlvLength* is set to 0000000100_2.

10.4.2.2.4 tlvValue

The format for the *tlvValue* field of the total reserved bandwidth TLV is shown in Figure 10.7

.

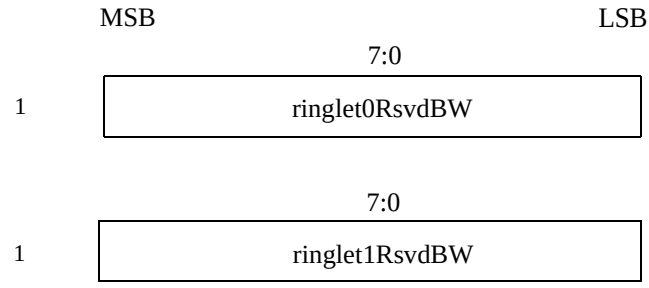


Figure 10.7—Total reserved bandwidth TLV tlvValue format

The *ringlet0RsvdBW* field carries the total reserved subclassA0 bandwidth of the station in ringlet0.

The *ringlet1RsvdBW* field carries the total reserved subclassA0 bandwidth of the station in ringlet1.

The reserved bandwidth shall use the same normalization as the *controlValue* field in fairness messages defined in Clause 9, except that the normalization shall not include weight normalization.

10.4.2.3 Station capabilities TLV

The station capabilities TLV encodes the station capabilities of the sending station, and is mandatory.

10.4.2.3.1 M

The M bit is set to 1.

10.4.2.3.2 tlvType

The *type* field within *tlvType* is set to 0000000011_2.

10.4.2.3.3 tlvLength

The *length* field within *tlvLength* is set to 0000000010_2.

10.4.2.3.4 tlvValue

The format for the *tlvValue* field of the station capabilities TLV is shown in Figure 10.8.

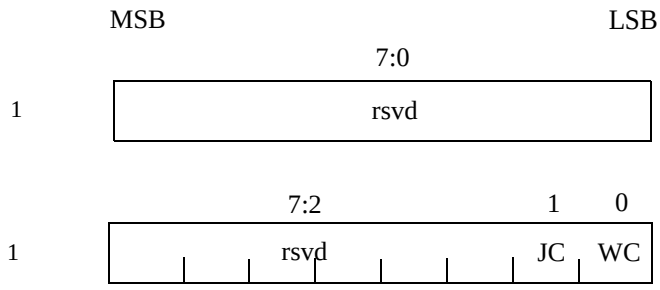


Figure 10.8—Station capabilities TLV tlvValue format

The values of the *rsvd*, jumbo frame capable (JC), and wrap capable (WC) fields are defined as follows.

Bits 7 through 0 of byte 1 and bits 7 through 2 of byte 2 are reserved for future use.

The JC bit is used to indicate if a station is jumbo frame receive capable. See 8.1 for the definition of jumbo frame receive capable.

The values of JC are defined in Table 10.4.

Table 10.4—Jumbo frame receive capable (JC) values

Value	Name	Description
0	NOTJUMB-OCAP	Not capable of receiving jumbo frames
1	JUMBOCAP	Capable of receiving jumbo frames

The WC bit is used to indicate if a station is wrap protection capable.

The values of WC are defined in Table 10.5.

Table 10.5—Wrap protection capable (WC) values

Value	Name	Description
0	NOTWRAPCAP	Not capable of wrap protection
1	WRAPCAP	Capable of wrap protection

10.4.2.4 Neighbor address TLV

The neighbor address TLV encodes the MAC addresses of the neighbors of the sending station, and is mandatory.

10.4.2.4.1 M

The M bit is set to 1.

10.4.2.4.2 tlvType

The *type* field within *tlvType* is set to 0000000100_2.

10.4.2.4.3 tlvLength

The *length* field within *tlvLength* is set to 0000001100_2.

10.4.2.4.4 tlvValue

The format for the *tlvValue* field of the neighbor address TLV is shown in Figure 10.9.

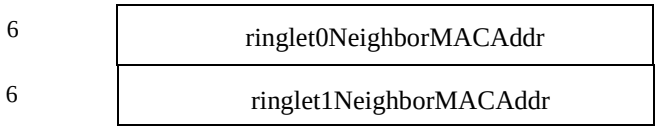


Figure 10.9—Neighbor address TLV tlvValue format

The *ringlet0NeighborMACAddr* field carries the MAC address of the neighboring station downstream on ringlet0. If the station's MAC address is unknown, it shall be all 0's.

The *ringlet1NeighborMACAddr* field carries the MAC address of the neighboring station downstream on ringlet1. If the station's MAC address is unknown, it shall be all 0's.

10.4.2.5 Individual reserved bandwidth TLV

The individual reserved bandwidth TLV encodes the individual station reserved subclassA0 bandwidth for each link on each ringlet, and is mandatory.

10.4.2.5.1 M

The M bit is set to 1.

10.4.2.5.2 tlvType

The *type* field within *tlvType* is set to 0000000101_2.

10.4.2.5.3 tlvLength

The *length* field within *tlvLength* is set to $(4xN)_2$.

10.4.2.5.4 tlvValue

The format for the *tlvValue* field of the individual reserved bandwidth TLV is shown in Figure 10.10.

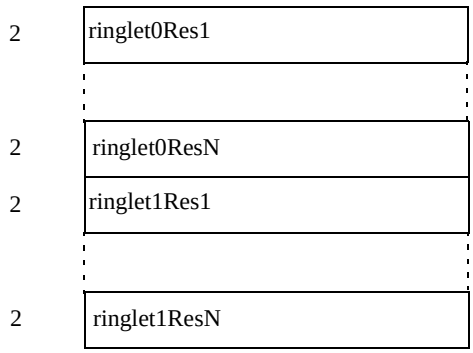


Figure 10.10—Individual reserved bandwidth TLV tlvValue format

The *ringlet0Res1* fields carry the amount of reserved subclassA0 bandwidth reserved by the sending station for each link on ringlet0. The fields are ordered by the number of hops away, such that field 1 is for the link 1 hop away, and field N is for the link N hops away.

The *ringlet1Res1* fields carry the amount of reserved subclassA0 bandwidth reserved by the sending station for each link on ringlet1. The fields are ordered by the number of hops away, such that field 1 is for the link 1 hop away, and field N is for the link N hops away.

The reserved bandwidth shall use the normalization defined in 10.4.2.2.

10.4.2.6 Station name TLV

The station name TLV encodes the operator assigned station name, and is optional. The station name is an arbitrary printable ASCII string.datory.

10.4.2.6.1 M

The M bit is set to 0.

10.4.2.6.2 tlvType

The *type* field within *tlvType* is set to 0000000110_2.

10.4.2.6.3 tlvLength

The *length* field within *tlvLength* is set to $(N)_2$.

10.4.2.6.4 tlvValue

The format for the *tlvValue* field of the station name TLV is shown in Figure 10.11.

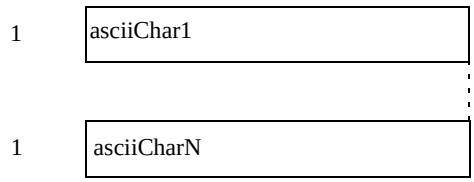


Figure 10.11—Station name TLV tlvValue format

The *asciiChar* fields contain the numeric ASCII values of the characters in the station name, ordered from 1 to N.

10.4.2.7 Vendor specific TLV

The Vendor specific TLV encodes the vendor specific information of the station, and is optional.

10.4.2.7.1 M

The M bit is set to 0.

10.4.2.7.2 tlvType

The *type* field within *tlvType* is set to 0100000000_2.

10.4.2.7.3 tlvLength

The *length* field within *tlvLength* is set to $(N+3)_2$.

10.4.2.7.4 tlvValue

The format for the *tlvValue* field of the vendor specific TLV is shown in Figure 10.12.

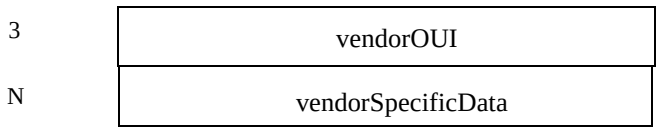


Figure 10.12—Vendor specific TLV tlvValue format

The vendor organizationally unique identifier (OUI) field contains the IEEE company identifier for the vendor.

The vendor specific data field includes vendor defined data. The exact definition of the vendor specific data field is outside of the scope of this standard.

10.5 Topology message handling

10.5.1 When generated

The topology message is broadcast:

a) On the initial start of RPR topology discovery.
b) Periodically.

The topology message is sent on a periodic timer. The topology message period is configurable from 50 ms to 10 seconds with 50 ms resolution and a default value of 100 ms.

10.5.1.1 Effect of receipt

The receipt of this message on the same ringlet as which it was sent (as indicated in the ringletID field) from any station causes the MAC Control sublayer to update its current local topology image.

10.5.1.2 Handling of topology messages during protection

Editors' Notes: *To be removed prior to final publication.*

The implications of the rule in this clause that a wrapping station shall strip topology messages upon receipt must be considered, given that no comparable rule is in effect for steering stations. This may result, in some double failure cases, in station capabilities information not being available for all stations in the topology database stored at one or more stations in wrapping rings. However, it is guaranteed that all stations will see the station capabilities information of all stations in the data reachable topology.

A station in wrapped protection state shall not wrap a topology message, and shall strip it after receiving it. The wrap eligible (WE) bit in the RPR header shall be set to zero for topology messages to ensure that they are not wrapped. Topology messages shall continue to be delivered and received on links that are in non-idle protection states.

Editors' Notes: *To be removed prior to final publication.*

The PAH is discussing the potential inclusion of a paragraph describing receive buffer size. This paragraph would read as follows:

To ensure rapid and bounded completion of topology discovery, each MAC control queue shall be capable of storing one topology discovery message from each of the possible sources on the ring; the minimal storage for one source shall not be compromised by the offered load from other stations.

The implication of including a statement like this in this clause is that similar statements would need to be included in other clauses defining MAC control protocols.

The above statement would not apply to topology messages.

11. Protection

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	Addressed most comments from draft 0.1. Those not yet addressed are specifically mentioned in editor's notes. Added Scope and moved normative portions of introduction to later in clause. Merged initial descriptive clauses into Overview. Added Steering and Wrapping clauses with detailed state diagrams for each. Copied requirements relevant to each from Draft 0.1 into each of the clauses.
Draft 0.3, June 2002	Addressed most comments from draft 0.2. Those not yet addressed are specifically mentioned in editor's notes. Added general protection overview. Merged common steering and wrapping requirements into a common protection requirements clause. Added extra definition to the protection hierarchy clause. Added sequence number to the protection frame format. Included new detailed state diagrams in tabular form for steering and wrapping clauses.
Draft 1.0, August 2002	Addressed most comments from draft 0.3. Those not yet addressed are specifically mentioned in editor's notes. The protection ad-hoc (PAH) is meeting to address the list of issues described in the editorial note immediately following this one. Major modifications made in D1.0 include: Clean-up of figures Clean-up of protection message section Addition of new wrapping state diagram
Draft 1.1, October 2002	Addressed most comments from draft 1.0. Those not yet addressed are specifically mentioned in editor's notes. The protection ad-hoc (PAH) is meeting to address the list of issues described in the editorial note immediately following this one. Primary modifications made in D1.1 include: Clean-up of figures Fixes in steering and wrapping state diagrams
Draft 2.0, December 2002	Addressed all comments from draft 1.1. The protection ad-hoc (PAH) is meeting to address the list of issues described in the editorial note immediately following this one. Primary modifications made in D2.0 include: Adding of overview flow chart Replacement of steering and wrapping state machines with a unified state machine Modifications across the clause to take into account that the fast topology message and the protection message are combined

Editors' Notes: *To be removed prior to final publication.*

The following issues remain to be addressed by the protection ad-hoc (PAH):

1. Comment #287, November: Determine any necessary modifications to the unified protection state machine to handle strict mode scenarios.

2. Agree on modifications to protection message to handle passthrough scenarios.

The following issues have been resolved by the protection ad-hoc (PAH):

1. Comment #491, September: Determine the functional partitioning between topology and protection.
Status: Completed. Additional modifications will come about through commenting.

2. Comment #565, July: Determine if and how non-revertive operation will be supported, and what the configuration options are for reversion (per interface and/or per station per class of service). If non-revertive operation is supported for wrapping on an interface of a station, it will apply to locally initiated wraps only. If a station on one end of the span wraps/unwraps, the station on the other end of the span will passively wrap/unwrap independent of whether it is configured as revertive or non-revertive on that interface.
Status: Completed based on resolution of comment #1504 from November.

3. Comment #567, July: Include description of what is covered by the scenario descriptions to be added later in this clause to resolve #581. Description will also be included of how protection switching interacts with topology discovery and ringlet selection.
Status: The PAH is continuing to work on this, via the annex containing scenarios.

4. Comment #576, July: Specify the behavior of a ring configured as a wrapping ring that has a steering-only station inserted into it.
Status: Completed. The behavior is known. The editors will add in the necessary requirements.

5. Comments #578 and #579, July: Determine the requirements of retransmission of protection messages. The options are exponential back-off and constant periodic transmission.
Status: Completed. The PAH determined that a bi-level periodic timer will be used.

6. Comment #581, July: Reorganize subclauses 11.4, 11.8.1, and 11.7.1 to illustrate key scenarios that illustrate the relevance of the underlying rules. These scenarios will include (for example) protection commanding, protection triggering and hold-off time, protection clearing and wait to restore time, protection hierarchy scenarios, and revertive/non-revertive behavior. The rules and the need for them will be described within the scenario descriptions. In addition, the scenario descriptions will indicate the relationship between protection switching, topology discovery, and ringlet selection.
Status: Completed. Additional modifications will come about through commenting.

7. Comment #582, July: Examine the issue of whether the protection protocol can by itself determine whether a newly added station is the same as the station that was previously connected on the interface, or whether this is fundamentally a function that should be provided by topology discovery.
Status: Completed. This is addressed by the former MAC address entries in the topology database.

8. Comment #585, July: Determine how to tie configured absolute time for loss of keepalives to variability in the interval for duration of fairness messages.
Status: Completed based on #276 from November.

9. Comment #589, July: Determine if the wrapping status code bit should be kept for informational and diagnostic purposes, and if there is direct need for this bit in the protection protocol itself.
Status: Completed. The wrapping status code bit will be kept for informational and diagnostic purposes.

10. Comment #594, July; comments #504, #509, and #510, September: Determine the exact behavior of revertive and non-revertive modes for steering protection.
Status: Completed based on #1504 from November.

Editors' Notes: *To be removed prior to final publication.*

11. Comment #595, July: Determine whether a station in a steering ring reports its local link protection state to the rest of the ring irrespective of protection requests present on the rest of the ring.
Status: Completed. A station in a steering ring will report its local link protection state to the rest of the ring irrespective of protection requests present on the rest of the ring.

12. Comment #596, July: Determine whether rule 5 from 11.7.1.2 should apply to steering or not.
Status: Completed. Rule 5 will apply to steering as well to facilitate the unification of steering and wrapping in a uniform state machine.

13. Comment #601: Determine the desired behavior and provide a scenario description for multiple SD conditions on a ring.
Status: Completed. The desired behavior is that stations detecting a SD condition will wrap immediately and will report SD immediately to the rest of the ring (if there is not already a wrap due to SD or a higher priority defect elsewhere on the ring). In the event that there is a simultaneous SD detection elsewhere on the ring, then there will be wraps on both degraded spans for a short interval, followed by unwrapping on both spans. The scenario description is in progress and will be included in the annex containing scenarios.

14. Comment #513, July: Determine whether there can be a common state diagram for steering and wrapping.
Status: Completed. Modifications to the state machine inserted in D2.0 will be made through commenting.

11.1 Scope

This clause describes the RPR protection protocol, which implements resiliency, an important RPR objective. The RPR protection protocol provides reliable mechanisms for sub-50 ms protection switching for all protected traffic on an RPR ring. It consists of a mandatory protection mechanism called steering, and an optional protection mechanism called wrapping. The protection protocol is completely defined for both mechanisms, and either by itself is sufficient to meet RPR requirements. The protocol resides in the MAC control sublayer, as shown in the shaded region of Figure 11.1.

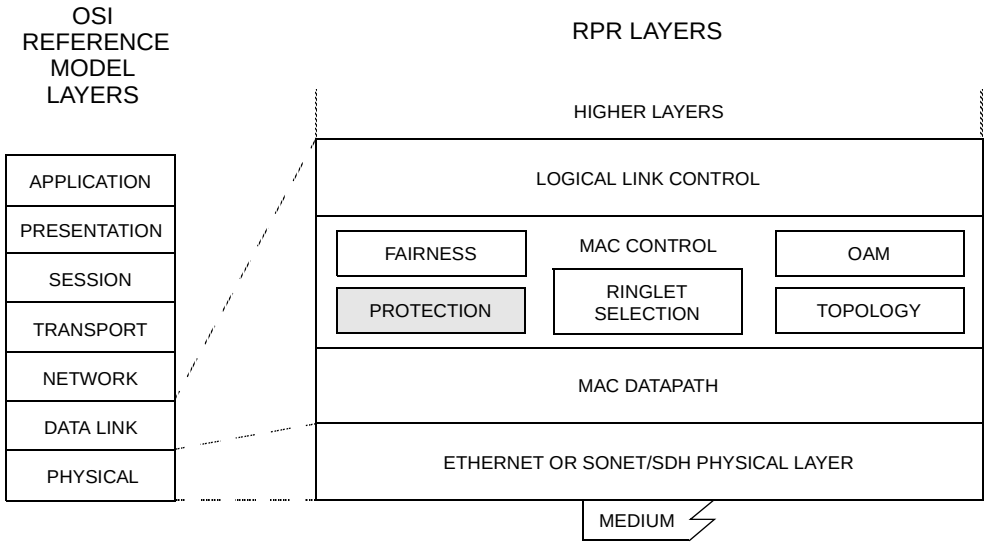


Figure 11.1—RPR layer diagram

The services and features provided are:

a) Sub-50 ms protection switching for unicast and multicast traffic based on media “hard” or “soft” failures or based on operator invoked command
b) Quick dissemination on the ring of information indicating media failures or operator invoked commands impacting use of the media
c) Support of a standard protection hierarchy
d) Tolerant of message loss but with minimal overhead
e) Greater bandwidth efficiency for unprotected services than path switching or bidirectional line switching
f) Operate independently of and in the absence of any management systems
g) Support dynamic addition and removal of stations to/from the ring
h) Scalable from one to hundreds of stations
i) Support of revertive and non-revertive protection switching operational modes
j) Operate without any master station on the ring

11.2 Overview

Editors' Notes: *To be removed prior to final publication.*

Additional description in the overview needs to be provided.

11.2.1 General protection overview

A protection switching protocol that provides the services and features described in 11.1 must define:

a) A hierarchy of protection requests that requires local action by a station and reporting of the protection state to the rest of the ring. 11.5 includes description of the protection hierarchy, and of the definition of triggers resulting in specific protection requests.
b) The local actions that can be taken by stations on the ring to prevent traffic loss. As described in the rest of 11.2, these consist of steering or wrapping.
c) A mechanism and control frame format for reporting local protection state to the rest of the ring. This is given in 11.6.
d) State machine and related requirements. These are given in 11.7. The state machine defines in detail what protection state information is reported to the rest of the ring, exactly when it is reported, and exactly when action is taken at a station to protect traffic.

The protection protocol does not directly switch traffic on the data plane. Rather, it supplies essential information to entities within the MAC that do, such as ringlet selection, or directly controls when wrapping occurs. It ensures that parameters such as hold-off time and wait to restore time specified by the network operator translate into correct station behavior. It ensures the robust interaction of operator-initiated protection requests and protection requests triggered by physical layer and MAC layer monitoring.

11.2.2 Steering protection

For steering protection, a station shall not wrap a failed segment when a failure is detected. Instead, a protection request message is sent to every station to indicate a link failure, as in the wrap protection scheme. When stations receive a protection request message indicating a failure, their steering database will be updated accordingly. It is the responsibility of each source station to direct its traffic onto ringlet0 or ringlet1, whichever avoids the failed link.

Frames destined to a station beyond the point of failure that have been transmitted onto the ring before the steering database is updated at the source station will be dropped at the failure point, since there is no delivery mechanism available.

For steering rings, when a link fails, it is simplest for multicast frames to be sent in both directions, with the TTL set to the number of stations on the ring between the source station and the defective span on each direction. Duplicate frames must not be allowed to arrive at any station on the ring as a result of a protection condition ceasing to exist.

Editors' Notes: *To be removed prior to final publication.*

The steering protection description above will be expanded to correspond to the wrapping description as stipulated in comments 89 and 613 from May.

11.2.3 Wrap protection

An overview of the behavior of a RPR wrap capable ring is given in this subclause. If an equipment or facility failure is detected, traffic going towards the failure is wrapped (looped) back to go in the opposite direction on the other ringlet (subject to the protection hierarchy). Wrapping takes place on the stations adjacent to the failure, under control of the protection switch protocol. The wrap re-routes the traffic away from the failure.

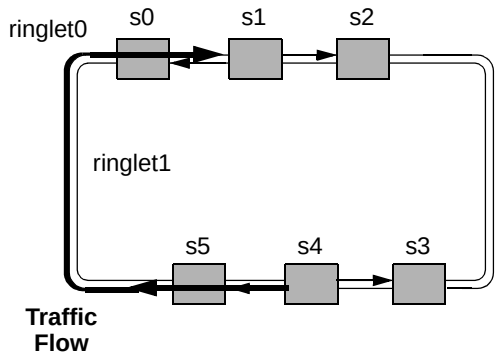


Figure 11.2—Data flow before fiber cut.

An example of the data paths taken before and after a wrap is shown in Figure 11.2 and Figure 11.3, respectively. Before the fiber cut, Station 4 sends to Station 1 via the path S4->S5->S0->S1.

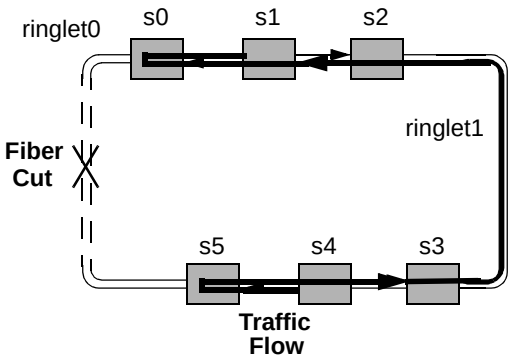


Figure 11.3—Data path after wrap

If there is a fiber cut between Station 5 and Station 0, Station 5 will wrap traffic on ringlet0 to ringlet1, and Station 0 will wrap traffic on ringlet1 to ringlet0. After the wraps have been set up, traffic from Station 4 to Station 1 initially goes through the non-optimal path S4->S5->S4->S3->S2->S1->S0->S1. S1 does not strip frames from the protection ringlet (ringlet0 in this case) to avoid frame mis-ordering during the protection event.

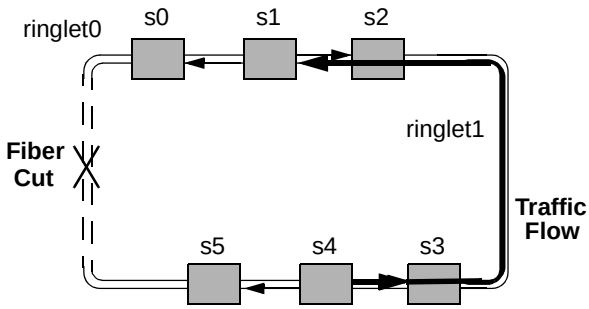


Figure 11.4—Data path after new topology discovery

Subsequently a new ring topology is discovered and a new optimal path S4->S3->S2->S1 amy be used, as shown in Figure 11.4. Note that the topology discovery and the subsequent optimal path selection are not part of the protection protocol. The client, using the options defined in Clause 6, may choose to re-steer the traffic as shown in Figure 11.4, or may choose to keep the traffic wrapped, as shown in Figure 11.3

Editors' Notes: *To be removed prior to final publication.*

As per comment 357 from March, references will be fixed later based on where references appear (in Clause 2 or Appendix A).

The ring wrap is controlled through SONET BLSR style protection switch signaling. It is an objective to perform the wrapping at least as fast as specified in ANSI T1.105-01 for BLSR.

11.3 Flow chart for the protection protocol

A flow chart showing the overall flow of the protection protocol and its relationship to the topology discovery protocol is shown in Figure 11.5. Detailed specification of the parts of this flow chart are given in various subclauses within Clause 10 and Clause 11. An overview of the parts of this flow chart is given in the remainder of this subclause.

11.3.1 Protection-related events at local station

Protection-related events at the local station that cause entry to the protection state machine include:

a) Local operator administrative request on a span of a station.
b) Local operational failure on a receive link of a station.
c) Clearing of a local administrative request or operational failure.
d) WTR timeout on a receive link of a station.
e) Administrative request or operational failure from other span of station.

The criteria for setting administrative request conditions, setting operational failure conditions, and clearing either type of condition are given in 11.5.

11.3.2 Receipt of protection messages

Receipt of a protection message first causes actions, including a sequence number check, described in 11.7.2. If the check passes, the protection state machine is entered. If the check fails, the protection message is discarded.

11.3.3 Protection state machine

The unified protection state machine for both steering and wrapping is defined in 11.8. The actions resulting from the protection state machine include:

a) Setting of the wrap status bit for a span of the station. This will be used later in the flow chart to cause a station to wrap.
b) Determination of the protection request value to be set into the link availability fields of the topology database for each receive link of the station.
c) Determination of whether a protection message will be triggered due to a change in either the wrap status bit for a span of the station, or due to a change in the protection request value for a receive link of the station.

11.3.4 Modification of topology database

The definition of the topology database and the rules for modifying and validating it are given in 10.2.6.

11.3.5 Triggering of protection message transmission

The triggers for protection message transmission are given in 11.7.1.

11.3.6 Wrapping/steering action

Based on the setting of the wrap status bit for a span of the station and whether a ring is a wrapping ring, a wrap or unwrap on a span of the station is triggered at this point in the flow chart. For steering, the topology database has already been updated with modified link availability information, which provides the necessary information for ringlet selection to perform steering, as defined in 6.3.

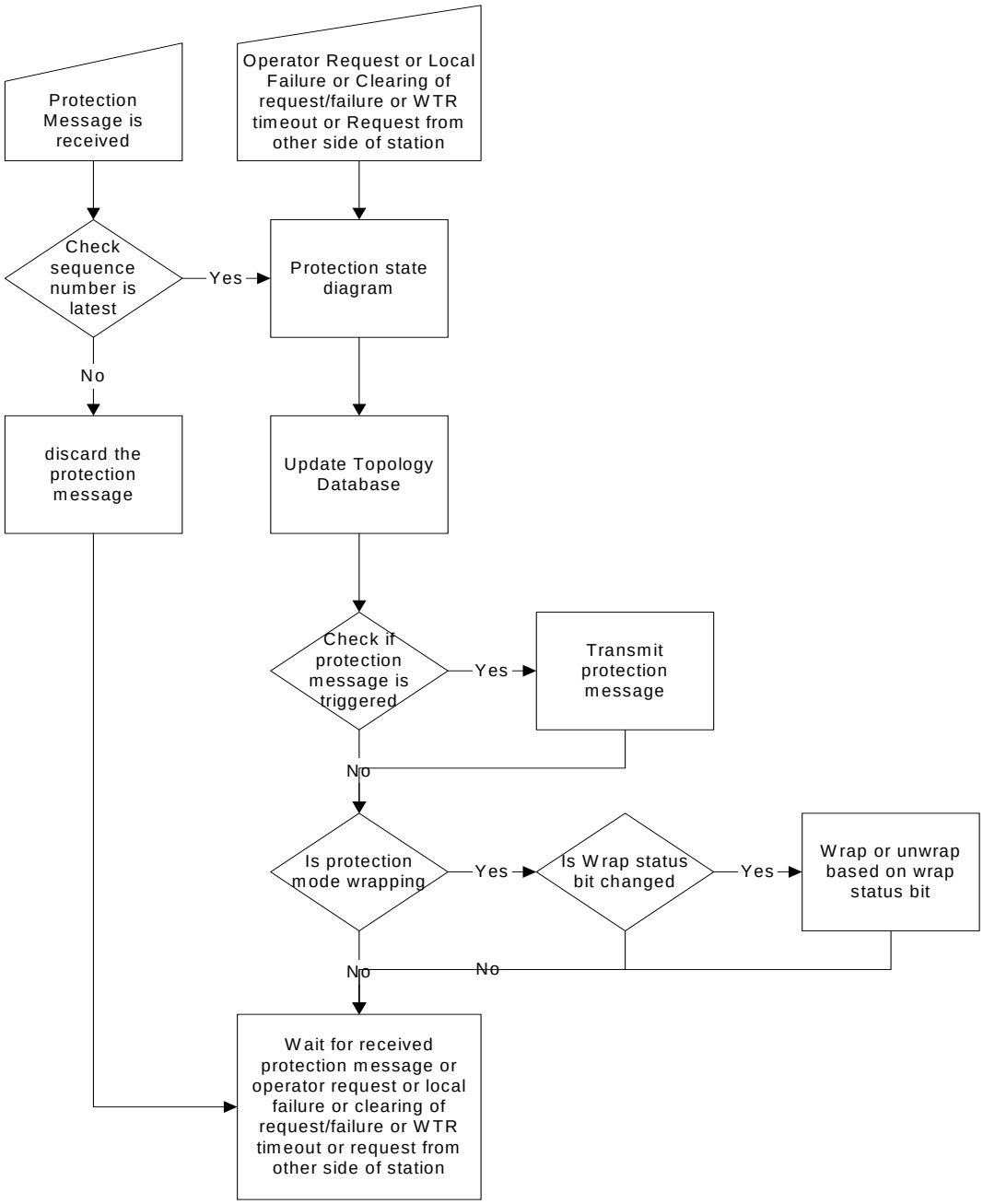


Figure 11.5—Flow chart for the protection protocol

11.4 Common protection rules

Editors' Notes: *To be removed prior to final publication.*

The rules below need to be reformatted to conform with the IEEE editorial template and style.

Fundamental protection rules in RPR include:

a) An RPR ring shall provide protection within 50ms of detection of a link or station failure.
b) All RPR stations shall provide support for steering, with support for wrapping optional.
c) All stations within the same RPR ring shall choose the same protection mechanism. Via the topology discovery protocol, every RPR station shall indicate if it supports wrapping protection or not. If all stations on an RPR ring are able to support wrapping protection, the choice of wrapping or steering for the protection mechanism shall be based on configuration by the system operator.

Editors' Notes: *To be removed prior to final publication.*

The determination of the PAH is that if all stations on an RPR ring are able to support wrapping protection, then wrapping shall be selected as the protection scheme. This addition will be submitted as a comment to the upcoming version of the draft.

Accordingly, no wrapping protection can be initiated until the full topology is learned. Otherwise, steering shall be selected as the default protection scheme on the RPR ring.

Editors' Notes: *To be removed prior to final publication.*

The editors will add in requirements to specify the behavior of a ring configured as a wrapping ring that has a steering-only station inserted into it, corresponding to the scenario in the topology and protection Annex.

4. Revertive operation shall be the default mode of operation in RPR rings. Non-revertive mode is supported through the configuration of an infinite WTR time. A particular value for WTR in the MIB (such as all 1's) will be interpreted by the protection protocol as being an infinite WTR time. The behavior of a non-revertive station in terms of reporting and clearing of WTR shall be fully consistent with all rules pertaining to WTR.

11.4.1 RPR protection frame transfer mechanism

a) Protection frames are transferred in a broadcast frame format between stations on the ring. A received frame is passed to the receiving station's MAC control sublayer.
b) Triggering rules for protection messages are given in 11.7.1.
c) Protection messages shall continue to be delivered on links that are in non-idle protection states.

11.4.2 RPR protection signaling mechanism

a) Protection switch signaling is performed using protection control frames as defined in Figure 11.6.
b) A station executing a local request signals the protection request on both short (opposite ringlet from the link for which the state is being reported) and long (same ringlet as the link for which the state is being reported) paths.

11.4.3 RPR protection protocol rules:

a) The protection request hierarchy values are listed in Table 11.4 (listed from highest priority to lowest priority). In general, a higher priority request preempts a lower priority request within the ring,

with exceptions noted as rules. The 4 bit values shown in the table correspond to the protection message request type (PMRT) field in the protection frame.

b) When a station which initially detected a failure discovers the disappearance of the failure, it enters WTR (for a user-configured WTR time-period). The configurable range for WTR is defined in 11.5.

c) When a station is in WTR mode, and detects (via matching the MAC address of the new neighbor to the former neighbor MAC address on that interface stored in the topology database) that a new neighbor is not the same as the old neighbor (stored while the old neighbor was still recognized as part of the topology), the station drops the WTR.
The information from the topology database is needed so that protection knows when to apply WTR without requiring protection to separately store topology information.

d) When a station receives a local protection request of type SD or SF and it cannot be executed (according to protocol rules), it keeps the request pending. (The request can be kept pending outside of the protection protocol implementation.)

e) If a local non-failure request (WTR, MS, FS) clears, and if there are no other requests pending, the station enters idle state.

f) If there are two link failures and two resulting WTR conditions on a single segment, the second WTR to time out brings both the links up. (After a WTR time expires, a station does not unprotect automatically, but waits until it receives idle messages from its neighbor on the previously failed segment.)

g) The WTR on any link is dropped when a higher priority request is detected elsewhere on the ring. This requires the station monitoring the link with the WTR condition to check whether there are higher priority requests elsewhere on the ring.

h) A configurable hold-off time must be supported with a range of the hold-off time is zero to at least 200 ms with 10 ms resolution. The default value is zero.

i) An MS command on an interface of a station will be rejected if there is another MS on the ring.

Editors' Notes: *To be removed prior to final publication.*

A subclause is needed here to put this clause in context with respect to the MAC reference model. This subclause will cover description of how protection switching relates to other components in the MAC reference model, the service interface to the client (to address comments 93 and 351 from March), and interactions between components of the MAC control sublayer. This will address comment 91 from May.
The description of how the components of the MAC control sublayer interact and make information available to each other may tie to a common topology/protection database. The determination of whether such a database is sufficient for this purpose depends on the definition of this database, which is tied to topology discovery as per comment 611 from March.
The MAC data path module may need notification from the protection module when a wrapping-based ring has returned to a fully normal operating state, as per comment 707 from May. This notification is required to enable the capability to immediately strip frames from the protection ringlet upon restoration of the complete bi-directional ring without requiring TTL to decrement fully.

11.5 Protection hierarchy and triggers

The protection switch protocol processes the following request types (in the order of priority, from highest to lowest). The triggers for these requests are defined below. All requests are signaled using protection control messages defined in 11.6.

1) Forced Switch (FS): Operator originated. Results in a protection switch away from a requested link (steering or wrapping all traffic away from the link). An example use of this request type is to add another station to the ring in a controlled fashion.
2) Signal Fail (SF): Automatic. Caused by a media Signal Failure or RPR keepalive failure. A media SF shall be detected based on the PHY_LINK_STATUS.indicate defined in 7.2.3. Results in a protection switch away from the impacted link (steering or wrapping all traffic away from the link). SONET examples of SF triggers are: Loss of Signal (LOS), Loss of Frame (LOF), Line Bit Error Rate (BER) above a preselected SF threshold, and Line Alarm Indication

Signal (AIS). Explicit definition of the SF triggers and SF clearing criteria for SONET are provided in the Telcordia GR-253-CORE and ANSI T1.105-01 standards, among others. An RPR keepalive failure is defined later in this clause. The configurable hold-off time, if non-zero, starts after the physical SF condition is detected and shall elapse before SF triggers. As described in 11.7, incorrect connection of interfaces of neighboring stations through miscabling shall also result in a MAC layer SF condition.

3) Signal Degrade (SD): Automatic. Caused by a media Signal Degrade (e.g. excessive Bit Error Rate). A media SD shall be detected based on the PHY_LINK_STATUS.indicate defined in 7.2.3. Results in a protection switch away from the impacted link (steering or wrapping all traffic away from the link). SONET examples of SD triggers are: Line BER or Path BER above a preselected SD threshold. Explicit definition of the SD triggers and SD clearing criteria for SONET are provided in the Telcordia GR-253-CORE and ANSI T1.105-01 standards, among others. The configurable hold-off time, if non-zero, starts after the physical SD condition is detected and shall elapse before SD triggers.
4) Manual Switch (MS): Operator originated. Like Forced Switched but of a lower priority. An example use of this request type is to force down a marginally operating link, but allow it to come back into use in the case of a more serious failure elsewhere on the ring.
5) Wait to Restore (WTR): Automatic. Entered after a link meets the restoration criteria following exit from a SF or SD condition. The protection switch protocol waits for the WTR time-out before restoring traffic. An example use of this request type is to prevent protection switch oscillations. The configurable range for WTR is defined later in this clause.

The protection module finds out about operator originated protection requests and clearing from the Layer Management Entity defined in Clause 13. It finds out about automatic protection requests via link status primitives defined in Annex B and Annex C for the Ethernet and SONET/SDH reconciliation sublayers, respectively. The protection module may also utilize automatic protection triggers mapped into SF or SD not explicitly defined in Annex B and Annex C.

The purpose of RPR keepalives is to provide an indication that a station has an acceptable degree of operational capability. The transmission of RPR keepalives ideally should occur only if all implemented checks of normal MAC operation are fulfilled.

An RPR keepalive failure on a receive span is defined by the failure to receive any SC-FCM messages (as defined in Clause 9) for a configurable time range from 2 ms to 10 ms, with resolution 1 ms and a default value of 3 ms. This means that to trigger an RPR keepalive failure, at least four consecutive SC-FCM messages must be missed, based on the 400 microsecond largest advertisement interval between SC-FCM messages.

Editors' Notes: *To be removed prior to final publication.*

The editors will ensure that the protection clause definition works if there is no signal degrade indication available, e.g. for PHYs that do not support signal degrade.

A SF condition due to loss of keepalives is not cleared solely by the re-start of reception of keepalives from a newly connected or re-connected neighbor station. Upon receipt of a single keepalive, the link transitions to WTR state. For a new neighbor, if a protection message is received from the neighbor before the WTR expires, the link is brought up upon receipt of the protection message. If the WTR expires prior to receipt of a protection message, then the link is brought up based on WTR regardless of whether the neighbor is a new neighbor or unchanged from before the SF condition.

An SF condition due to miscabling is triggered by a ringlet ID mismatch on a protection message received from a neighboring station (TTL == MAX_STATIONS).

The SF condition due to miscabling is cleared by the receipt of a single protection message received from a neighboring station with the correct ringlet ID.

WTR shall be configured with values in the range of 0-1440 sec. with resolution of 1 second and with a default value of 10 seconds.

The following applies for both steering and wrapping rings unless stated otherwise.

Link failure or degradation indications (SF or SD) are reported to the entire ring, independent of whether there is a higher priority request in existence on the ring. The purpose of this reporting is to enable each station to have as complete a picture as possible of the status of all links on the ring, which is beneficial from a diagnostic perspective. This reporting is independent of whether, for wrapping rings, wrapping is caused by link degradation indications. A wrap shall always be initiated based on SF. A wrap shall be initiated based on SD only if there is no equal or higher priority request elsewhere on the ring, as detailed in 11.8.

A forced switch (FS) indication always goes into effect (and causes a wrap in wrapping rings) and is reported to the entire ring, except in the case described in 11.8 (to prevent a single-ended wrap). A manual switch shall go into effect only if there is no equal or higher priority request elsewhere on the ring, as detailed in 11.8.

A wait to restore (WTR) is cleared immediately if a higher priority request is initiated from elsewhere on the ring.

Protection requests from any station travel around the ring and are stored in the topology and status database at all stations on the ring. For steering rings, to determine which ringlet is preferred for the transmission of frames from a given source to a given destination, the highest protection request on each path connecting the source and destination must be compared. For example, if there are links with protection request SD on the portion of ringlet0 connecting station A to station B, and there is a link with protection request SF on the portion of ringlet1 connecting station A to station B, then ringlet0 would be selected for steering traffic from station A to station B.

All protection switches are performed bidirectionally (protect at both ends of a segment for both transmit and receive directions, even if a failure is only unidirectional).

11.6 Protection message frame format

The protection switch message is used for signaling various link failures and degradations. The protection switch message frame format is outlined in Figure 11.6. The rprHeader is the standard RPR header for RPR control frames as defined in 8.3. The controlVersion and controlType fields correspond to the fields with those names that are part of the RPR payload of the RPR control frame. The combination of the protectionMessageByte and the sequenceNumber fields comprise the control PDU portion of the RPR payload defined in 8.3.

In rprHeader, the destination MAC Address is the broadcast address defined in Clause 6. Protection switch messages are broadcast in order to minimize the transmission delay. The source MAC address is the MAC address of the originator of the protection message. The wrap eligible (WE) bit shall be set to zero for protection messages to ensure that they are not wrapped. The protection message is sent as a MAC Control message with a control type value of 2, for protection. It is sent as a broadcast frame on all ringlets, with a TTL of MAX_STATIONS, removed by the source station.

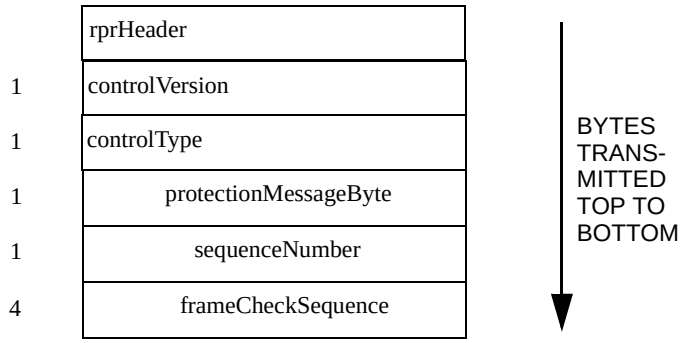


Figure 11.6—Protection frame format

The protection switch specific fields are detailed below.

11.6.1 Protection message byte

The protection message byte is shown in Figure 11.7. The subfields of the protection message byte are described in the subclauses below.

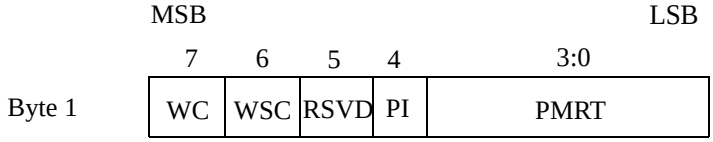


Figure 11.7—Protection message byte format

11.6.1.1 Wrap protection capable (WC)

The WC bit is used to indicate if a station is wrap protection capable. The WC bit is included in the protection message to minimize frame loss for scenarios where a steering station is inserted into a wrapping ring (see Annex J). Since protection messages are stripped by stations receiving messages on a wrapped span, the first protection message sent by a newly inserted steering station will be received by the neighboring stations (which are wrapped). Those stations will immediately unwrap, but other stations on the ring will not find out that the ring has switched to steering mode until the next protection message is transmitted by the steering station. To minimize frame loss, the WC bit is included in the protection message, which has the shortest fast transmission period.

The values of WC are defined in Table 11.1

Table 11.1—Wrap protection capable (WC) values

Value	Name	Description
0	NOTWRAPCAP	Not capable of wrap protection
1	WRAPCAP	Capable of wrap protection

11.6.1.2 Wrapping status code (WSC)

The Status Code is used on rings utilizing wrapping protection to indicate whether a wrap is present on an any interface to a station. More precisely, it indicates whether a station is in Wrapping state in 11.7.

The values of WSC are defined in Table 11.2. A WSC value of 0 is used in steering rings

Table 11.2—Wrapping status code values

Value	Name	Description
0	IDLE	No wrap in effect
1	WRAPPED	Protection switch completed in wrapping ring- traffic wrapped (W)

11.6.1.3 Reserved (RSVD)

Bit 5 is reserved for future use.

11.6.1.4 Path indicator (PI)

The Path Indicator indicates on which path a protection message is sent, long and short. Short path messages are sent towards the failed link through the opposite ringlet. They indicate a failure on the other ringlet on the link immediately preceding the station whose address is given in the source address of the protection request message. For example, a protection message with a short path indicator received on ringlet0 from a station indicates that the protection state is for the receive link to that station on ringlet1. Long path messages are sent away from the failed link on the same ringlet. They indicate a failure on the same ringlet on the link immediately preceding the station whose address is given in the source address of the protection request messages. For example, a protection message with a long path indicator received on ringlet0 from a station indicates that the protection state is for the receive link to that station on ringlet0.

The values of PI are defined in Table 11.3.

Table 11.3—Path indicator values

Value	Name	Description
0	SHORT	Message sent on short path
1	LONG	Message sent on long path

11.6.1.5 Protection message request type (PMRT)

A definition of the protection message request types is given in 11.5. The values of PMRT are defined in Table 11.4

The bit ordering shown in Table 11.4 is from most significant bit to least significant bit. The PMRT values are the same as those defined in GR-1230 and used for SONET protection.

The protection control messages are shown in this document as:

Table 11.4—Protection message request type values

Value (bin)	Name	Description
1101_2	FS	Forced switch (FS)
1011_2	SF	Signal fail (SF)
1000_2	SD	Signal degrade (SD)
0110_2	MS	Manual switch (MS)
0101_2	WTR	Wait to restore (WTR)
0000_2	IDLE	No request (IDLE)

{requestType, sourceAddress, wrapStatus, pathIndicator}

11.6.2 Sequence number

The sequence number is an 8-bit field that is incremented each time that the contents of a protection control message change, e.g. the same sequence number is used for the all copies of a given protection control message until its contents change. This is needed to prevent processing of out of order protection control messages. This can occur due to messages being received on the short path and on the long path in a transient scenario.

Editors' Notes: *To be removed prior to final publication.*

The text of the below subclauses as written are generally descriptive rather than normative. The appropriate portions of the text need to be made normative.

11.7 Protection message handling

11.7.1 When generated

Editors' Notes: *To be removed prior to final publication.*

The PAH is discussing adding a protection message trigger upon topology validation failure at a station.

Passthrough mode can be handled by topology discovery through the inclusion of a trigger bit in protection messages. The trigger bit would be set based on the detection of a passthrough trigger such as a change in neighbor identity for TTL==MAX_STATIONS indicating that the immediate neighbor of a station has changed. All other stations on the ring would respond to the trigger bit by sending copies of their existing protection messages without the trigger bit set, and by clearing their stored sequence numbers for every other station on the ring. This approach enables topology to be quickly rediscovered during passthrough events.

The protection message is broadcast:

a) On the initial start of RPR topology discovery.
b) At any point that a station determines that there is a change in protection information, as defined in the protection state machine defined in 11.8. A change in protection information refers to a change in the reported protection status on a receive link monitored by the station, or a change in the wrap status of the span of which that link is a part. A change in the wrap capable bit shall not trigger trans-

mission of the protection message; this information will be disseminated through the regular periodic transmissions of the topology and protection messages.

c) At any point that a station detects a new station on the ring.

d) Periodically.

The protection messages are sent on a bi-level periodic timer, starting with the fast period. The fast protection message period is configurable from 1 ms to 20 ms with 1 ms resolution and a default value of 10 ms. The fast rate timer is used for the first 8 messages sent after triggering. The slow protection message period is configurable from 50 ms to 10 seconds with 50 ms resolution and a default value of 100 ms. Each time a protection message is triggered by a change in the information contained in the message or upon initialization, the fast protection message period is used, followed by the slow protection message period until a new message is triggered from the station.

11.7.2 Effect of receipt

The receipt of this message on the same ringlet as which it was sent (as indicated in the ringletID field) from any station causes the MAC Control sublayer to update its current local topology image, as long as the sequence number check has passed. The sequence number check is described in the flow chart shown in Figure 11.8.

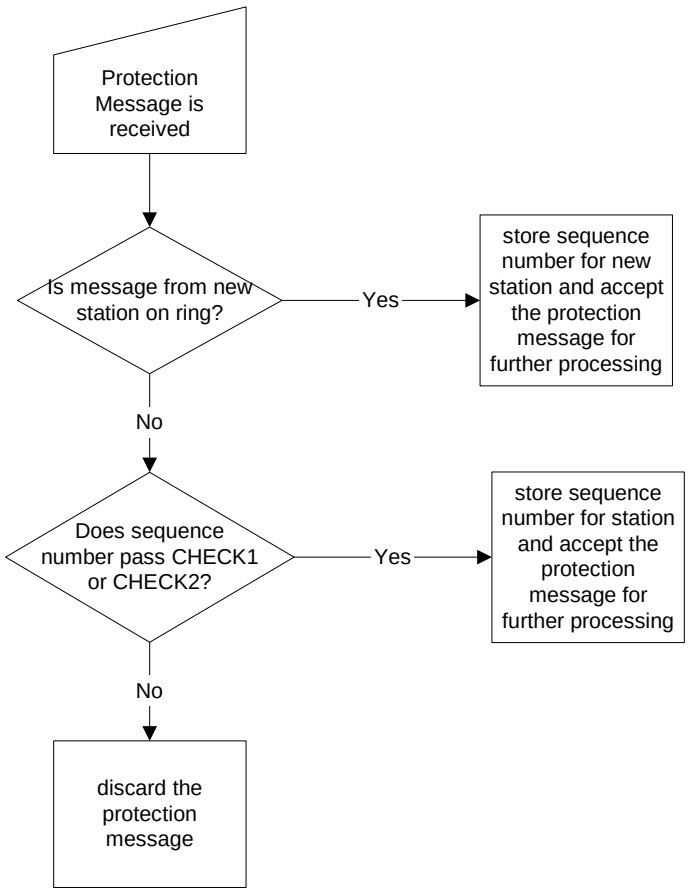


Figure 11.8—Flow chart for sequence number check

CHECK1 is ((seqNumStored >= ((int)(MAX_STATIONS/2)+1)) && (seqNumStored <= MAX_STATIONS)) && ((seqNumRcvd > seqNumStored) || (seqNumRcvd <= seqNumStored - ((int)(MAX_STATIONS/2)+1))).

CHECK2 is (!((seqNumStored >= ((int)(MAX_STATIONS/2)+1)) && (seqNumStored <= MAX_STATIONS)) && ((seqNumRcvd > seqNumStored) && (seqNumRcvd <= seqNumStored +((int)(MAX_STATIONS/2)+1))))

In CHECK1 and CHECK2, seqNumStored is the sequence number stored for the station from which the protection message is received. seqNumRcvd is the sequence number in the received protection message. If either check passes, seqNumStored is set equal to seqNumRcvd, and the protection message is accepted for further processing.

The receipt of this message on the same ringlet as which it was sent (as indicated in the ringletID field) from a neighbor station (as determined by TTL==MAX_STATIONS) causes the MAC Control sublayer to validate and (if needed) update the identity of its neighbor, as long as the sequence number check has passed.

The receipt of this message on a ringlet other than that on which is was sent (as indicated in the ringletID field) from a neighbor station (as determined by TTL==MAX_STATIONS) causes the MAC Control sublayer to discard the message, place the link upon which the message was received into a non-operational state, and generate a miscabling error.

11.7.3 Handling of protection messages during protection

A station in wrapped protection state shall not wrap a protection message, and shall strip it after receiving it. The wrap eligible (WE) bit in the RPR header shall be set to zero for protection messages to ensure that they are not wrapped. Protection messages shall continue to be delivered and received on links that are in non-idle protection states.

11.8 Protection state machine

This subclause contains a unified state machine for both steering and wrapping protection. State transitions that are specific to either steering or wrapping are noted within the state machine.

11.8.1 Listing of rules

Editors' Notes: *To be removed prior to final publication.*

The rules below need to be reformatted to conform with the IEEE editorial template and style.

11.8.1.1 RPR protection protocol rules

The following rules are applicable to steering protection only:

a) A protection control frame shall be accepted only if the sequence number contained within the control header as defined in 11.6 meets the conditions defined in the state diagram in 11.8.4.
b) When there is more than one request of priority < SF, the first request to complete long path signaling will take priority. However, a higher priority request can preempt the request as long as its long path signal is completed.

The following rules are applicable to wrapping protection only:

a) Requests >= SF can coexist. All requests above SF need to be cleared before the state is transferred into idle state.
b) Requests < SF can not coexist with other requests. A higher priority request will preempt a lower priority request.

   For wrapping, two different spans with SF conditions will result in simultaneous wraps on both spans. The desired behavior for two or more different spans with SD conditions, or two or more different spans with MS conditions, is that the affected spans are equally usable, and therefore no wraps shall result. If two or more SD conditions, or two or more MS conditions, occur nearly simultaneously, then there may be a transient condition where there are wraps on some of the affected spans for a brief interval, but stations of these affected spans will unwrap inmediately when they find out that there is more than one SD condition, or more than one MS condition, in the ring. There is no corresponding condition for steering, as ringlet selection for traffic entering a ring with two or more spans with SD conditions, or two or more spans with MS conditions, can be handled independently at each source station.
c) When there is more than one request of priority < SF, the first request to complete long path signaling will take priority. However, a higher priority request can preempt the request as long as its long path signal is completed.

d) If a short path FS request is present on a given segment, and a SF/SD condition takes place on the same segment, a station shall accept and process the SF/SD condition ignoring the FS. (Without this rule, a single ended wrap condition could take place, wrapping on only one end of a segment.)

11.8.2 Parameters

The parameters below are used in the protection state diagram presented later in this document.

a) FS
Forced switch is in effect for a given local receive link to a station.
b) SF
Signal fail is in effect for a given local receive link to a station.
c) SD
Signal degrade is in effect for a given local receive link to a station.
d) MS
Manual switch is in effect for a given local receive link to a station
e) WTR
Wait to restore
f) IDLE
Idle (no protection request).
g) REQ
Any protection request except IDLE.
h) SELF
Source MAC address for station sending protection message.
i) Wrap status bit
j) I
Not wrapped on the interface.
k) W
Wrapped on the interface.
l) S
Short path indication, e.g. protection message sent on other ringlet from that on which the link with the protection request lies.
m) L
Long path indication, e.g. protection message sent on same ringlet on which the link with the protection request lies.
n) Neighbor
MAC address of station on the other end of the affected span.
o) NewNbr
MAC address of station on the other end of the affected span after connectivity is regained on the span.
p) sideProtectionState
The protection state of a receive link of a given station on the ringlet to which this state diagram applies. This corresponds to the receive link availability in the topology database for the given station (west for ringlet 0, east for ringlet 1).
q) otherSideProtectionState
The protection state of the other side's receive link of a given station. This corresponds to the receive link availability in the topology database for the given station that is not used for sideProtectionState.
r) otherSideProtectionREQ
Other side protection request.
s) neighborProtectionState
The protection state for the neighbor station connected to the receive link of a given station. The protection state is for the receive link of the neighbor station on the opposite ringlet.

t) Other
Other station on the ring (not the given station or the neighbor station).
u) localFailure
Self detect failure like SF, SD.
v) localRequest
Local protection request like FS, MS, WTR expired.
w) localFailureClear
Self detect failure like SF, SD clears.
x) localRequestClear
Local protection request like FS, MS clears.
y) sidePendingReq
The pending request for the receive link.

11.8.3 Functions

The functions below are used in the protection state diagram presented later in this document. A brief description of the purpose of this functions is:

avoidSingleEndedWrap: The purpose of this function is to prevent an FS request on a span of a station from causing a single-ended wrap. The FS request is rejected if the neighbor on the same span detects an SF on its receive link. This prevents situations where the clearing of the FS cannot be communicated to the neighbor station. If TRUE, an FS request is accepted on a span of a station.

allowCoexistingProtection: The purpose of this function is to determine if protection action should be taken based on a local FS request or local SF condition, both of which are protection conditions that can coexist with other such conditions on the rest of the ring. The protection action consists of wrapping in the case of a wrapping ring. (Modification of link protection status reported to the rest of the ring for both steering and wrapping rings is specified within the protection state machine). If TRUE, REQ (FS or SF) is accepted (and will cause a wrap in a wrapping ring). This function will return false if an SF condition occurs on a span that is already FS.

allowNonCoexistingProtection: The purpose of this function is to determine if protection action should be taken based on a local MS request or local SD condition, both of which are protection conditions that cannot coexist with other such conditions on the rest of the ring. The protection action consists of wrapping in the case of a wrapping ring. (Modification of link protection status reported to the rest of the ring for both steering and wrapping rings is specified within the protection state machine). If TRUE, REQ is accepted (and will cause a wrap in a wrapping ring). This function pertains to conditions below SF in the protection hierarchy. These conditions will not cause a wrap if there is any other condition elsewhere on the ring equal to or higher than this condition in terms of the protection hierarchy.

isProtectionGranted: The purpose of this function is to combine avoidSingleEndedWrap, allowCoexistingProtection, and allowNonCoexistingProtection to determine if protection action needs to be taken. The protection action consists of wrapping in the case of a wrapping ring. (Modification of link protection status reported to the rest of the ring for both steering and wrapping rings is specified within the protection state machine). If TRUE, a wrap will be caused in a wrapping ring.

isProtectionPreempted: The purpose of this function is to determine, due to a protection event elsewhere on the ring, if an existing wrap should be removed, and if the existing protection status of a link should be set to IDLE (in the case of MS and WTR). If TRUE, a REQ from elsewhere on the ring will preempt wrapping protection on this link and in the cases of MS and WTR (as specified in the protection state machine) the protection status of the appropriate link will be set to IDLE.

a) avoidSingleEndedWrap(REQ)
TRUE: ((REQ==FS)&&(neighborProtectionState!=SF)) || (REQ!=FS)

b) allowCoexistingProtection(REQ)
TRUE: ((REQ >=SF)&&(REQ >sideProtectionState)) || (REQ<SF)
c) allowNonCoexistingProtection(REQ)
TRUE: ((REQ < SF)&&(REQ > all link availability values in topology database not corresponding to sideProtectionState and neighborProtectionState)) || (REQ>=SF)
d) isProtectionGranted(REQ)
avoidOneSideWrap(REQ)&&
isCoexistedAllowProtection(REQ)&&
isNonCoexistedAllowProtection(REQ)
e) isProtectionPreempted(REQ)
TRUE: ((sideProtectionState < SF) * (sideProtectionState <= REQ))

11.8.4 State machine

The state machine in Table 11.5 shows the required conditions for the following types of protection actions to be taken:

a) Setting of the wrap status bit for a span of the station. This will be used later in the flow chart to cause a station to wrap.
b) Determination of the protection request value to be set into the link availability fields of the topology database for each receive link of the station.
c) Determination of whether a protection message will be triggered due to a change in either the wrap status bit for a span of the station, or due to a change in the protection request value for a receive link of the station.

Protection-related events at the local station that cause entry to the protection state machine include:

a) Local operator administrative request on a span of a station.
b) Local operational failure on a receive link of a station.
c) Clearing of a local administrative request or operational failure.
d) WTR timeout on a receive link of a station.
e) Administrative request or operational failure from other span of station.

The variable sideProtectionState reflects the link protection state that is reported to the rest of the ring (and used internally to the station to update the topology and status database defined in 10.2.6) for each incoming or receive link to a given station. As there are two receive links (or spans) to the station, two instances of this state machine are being executed independently and in parallel, one for each receive link (or span). The state machine takes into account wait to restore time for a given link. The state machine also takes into account a pending request value per link for existing protection requests underlying the protection request reported to the rest of the ring (that is highest in the protection hierarchy).

Table 11.5—Protection state machine for an receive link

Current state		Row	Next state	
state	condition		action	state
IDLE	isProtection-Granted(localRequest)	1	sideProtectionState = localRequest Set Wrap Status bit (wrap only) Transmit Protection Message	FS/MS

Table 11.5—Protection state machine for an receive link *(continued)*

Current state		Row	Next state	
state	**condition**		**action**	**state**
	isProtection-Granted(localFailure)	2	sideProtectionState = localFailure Set Wrap Status bit (wrap only) Transmit Protection Message	SF/SD
	!isProtection-Granted(localFailure)	3	sideProtectionState = localFailure Transmit Protection Message	IDLE
	Rx{REQ,Neigh-bor,W,S}	4	Set Wrap Status bit Transmit Protection Message	Wrapping-FromNeigh-bor(WFN)
	--	5	--	IDLE
FS/MS	localRequestClear * !isProtection-Granted(sidePendin-gReq) * (sidePendingReq != IDLE)	6	sideProtectionState = sidePendingReq sidePendingReq = IDLE Clear Wrap Status bit (wrap only) Transmit Protection Message	SF/SD
	localRequestClear * isProtection-Granted(sidePendin-gReq) * (sidePendingReq != IDLE)	7	sideProtectionState = sidePendingReq sidePendingReq = IDLE Set Wrap Status bit (wrap only) Transmit Protection Message	SF/SD
	localRequestClear * (sidePendingReq == IDLE)	8	sideProtectionState = IDLE Clear Wrap Status bit (wrap only) Transmit Protection Message	IDLE
	isProtection-Granted(localRequest)	9	sideProtectionState = localRequest Set Wrap Status bit (wrap only) Transmit Protection Message.	FS/MS
	isProtection-Granted(localFailure)	10	sideProtectionState = localFailure Set Wrap Status bit (wrap only) Transmit Protection Message.	SF/SD
	!isProtection-Granted(localFailure) * (sideProtectionState == FS)	11	sidePendingReq = localFailure	FS/MS
	!isProtection-Granted(localFailure) * (sideProtectionState == MS)	12	sideProtectionState = localFailure Transmit Protection Message.	SF/SD
	Rx{SF,Neigh-bor,W/I,S/L} * (sideProtectionState == FS) * (sidePendingReq > IDLE)	13	sideProtectionState = sidePendingReq Transmit Protection Message	SF/SD

Table 11.5—Protection state machine for an receive link *(continued)*

Current state		Row	Next state	
state	**condition**		**action**	**state**
	Rx{SF,Neighbor,W/I,S/L} * (sideProtectionState == FS) * (sidePendingReq == IDLE)	14	sideProtectionState = sidePendingReq Transmit Protection Message	IDLE(steer) WFN(wrap)
	Rx{REQ,Neighbor,W/I,S/L} * (sideProtectionState == MS) * (REQ > MS)	15	sideProtectionState = IDLE Set Wrap Status bit if W set on short path message (wrap only) Transmit Protection Message	IDLE
	Rx(REQ, Other,W,S/L) *isProtectionPreempted(REQ)	16	sideProtectionState = IDLE Clear Wrap Status bit (wrap only) Transmit Protection Message	IDLE
	isProtectionPreempted(otherSideProtectionReq)	17	sideProtectionState = IDLE Clear Wrap Status bit (wrap only) Transmit Protection Message	IDLE
	sidePendingReq cleared	18	sidePendingReq = IDLE	FS/MS
	--	19	--	FS/MS
SF/SD	localFailureClear * sidePendingReq == IDLE	20	sideProtectionState = WTR Transmit Protection Message	WTR
	localFailureClear * (sidePendingReq != IDLE) * isProtectionGranted(sidePendingReq)	21	sideProtectionState = sidePendingReq sidePendingReq = IDLE Set Wrap Status bit (wrap only) Transmit Protection Message	SF/SD
	localFailureClear * (sidePendingReq != IDLE) * !isProtectionGranted(sidePendingReq)	22	sideProtectionState = sidePendingReq sidePendingReq = IDLE Clear Wrap Status bit (wrap only) Transmit Protection Message	SF/SD
	isProtectionGranted(localRequest)	23	sidePendingReq = sideProtectionState sideProtectionState = localRequest Set Wrap Status bit (wrap only) Transmit Protection Message.	FS/MS
	Rx{FS,Neighbor,W,S} * (sideProtectionState != SF)	24	Set Wrap Status bit (wrap only) Transmit Protection Message	SF/SD
	Rx(REQ, Other,W/I,S/L) *isProtectionPreempted(REQ)	25	Clear Wrap Status bit Transmit Protection Message	SF/SD

Table 11.5—Protection state machine for an receive link *(continued)*

Current state		Row	Next state	
state	**condition**		**action**	**state**
	isProtectionPre-empted(other-SideProtectionREQ)	26	Clear Wrap Status bit Transmit Protection Message	SF/SD
	--	27	--	SF/SD
WFN	Rx(IDLE,Neigh-bor,I,S/L)	28	Clear Wrap Status bit Transmit Protection Message	IDLE
	isProtection-Granted(localRequest)	29	sideProtectionState = localRequest Set Wrap Status bit Transmit Protection Message	FS/MS
	isProtection-Granted(localFailure)	30	sideProtectionState = localFailure Set Wrap Status bit Transmit Protection Message	SF/SD
	!isProtection-Granted(localFailure)	31	sideProtectionState = localFailure Transmit Protection Message	SF/SD
	--	32	--	WFN
WTR	Rx{IDLE,Neigh-bor,W,S/L}* WTR expired	33	sideProtectionState = IDLE Clear Wrap Status bit Transmit Protection Message	IDLE
	Rx{REQ, Neigh-bor,W,S/L}*(REQ>WTR)	34	sideProtectionState = IDLE Set Wrap Status bit Transmit Protection Message	WFN
	In steering mode * WTR expired	35	sideProtectionState = IDLE Transmit Protection Message	IDLE
	Rx{IDLE, NewNbr,I,S}	36	sideProtectionState = IDLE Clear Wrap Status bit (wrap only) Transmit Protection Message	IDLE
	isProtection-Granted(localRequest)	37	sideProtectionState = localRequest Set Wrap Status bit (wrap only) Transmit Protection Message	FS/MS
	isProtection-Granted(localFailure)	38	sideProtectionState = localFailure Set Wrap Status bit (wrap only) Transmit Protection Message	SF/SD
	Rx(REQ, Other,I/W,S/L) *isProtectionPre-empted(REQ)	39	sideProtectionState = IDLE Clear Wrap Status bit (wrap only) Transmit Protection Message	IDLE
	isProtectionPre-empted(other-SideProtectionReq)	40	sideProtectionState = IDLE Clear Wrap Status bit Transmit Protection Message	IDLE

Table 11.5—Protection state machine for an receive link *(continued)*

Current state		Row	Next state	
state	**condition**		**action**	**state**
	--	41	--	WTR

Row 11.5-1: Local administrative request granted. Causes transmission of protection messages reporting the new link protection state, and wrap for wrapping rings.

Row 11.5-2: Local failure causes wrap. Wrap check is not necessary for steering rings. Causes transmission of protection messages reporting the new link protection state, and wrap for wrapping rings.

Row 11.5-3: Local failure does not cause wrap. Wrap check is not necessary for steering rings. Causes transmission of protection messages reporting the new link protection state.

Row 11.5-4: Wrap Only. Protection request received from neighbor on the short path stating that the neighbor is wrapped. This causes this station to wrap and to transition to WFN state. The WFN state corresponds to when a station is wrapped but reporting an IDLE protection state on the receive link that it is monitoring.

Row 11.5-5: Remain in IDLE state.

Row 11.5-6: FS/MS cleared but wrap protection not granted for non-IDLE pending request (either failure or administrative request). Wrap check is not necessary for steering rings. This can occur if the pending condition is SD but there is already a SD elsewhere on the ring. This results in reporting of the new protection condition, and in clearing of any existing wrap on the interface.

Row 11.5-7: FS/MS cleared and wrap protection granted for non-IDLE pending request (either failure or administrative request). Wrap check is not necessary for steering rings. This results in reporting of the new protection condition, and in wrapping on the interface.

Row 11.5-8: FS/MS cleared and no pending request. This results in reporting of IDLE for this link, and in clearing of any existing wrap on the interface.

Row 11.5-9: Local administrative request granted. Causes transmission of protection messages reporting the new link protection state, and wrap for wrapping rings.

Row 11.5-10: Local failure causes wrap (check is not necessary for steering rings). Causes transmission of protection messages reporting the new link protection state, and wrap for wrapping rings.

Row 11.5-11: Local failure does not cause wrap and existing link protection state is FS. Wrap check is not necessary for steering rings. Causes failure condition to go pending.

Row 11.5-12: Local failure does not cause wrap and existing link protection state is MS. Wrap check is not necessary for steering rings. Causes MS to be cleared and transmission of protection messages reporting the new link protection state.

Row 11.5-13: SF protection request received from neighbor on the short or long path stating that the neighbor is wrapped or not wrapped. If the current link protection state is FS, then the FS is cleared and replaced by the pending request (SF or SD). Transmission of protection messages is initiated reporting the new link protection state.

Row 11.5-14: SF protection request received from neighbor on the short or long path stating that the neighbor is wrapped or not wrapped. If the current link protection state is FS, then the FS is cleared and replaced by the pending request (IDLE). Transmission of protection messages is initiated reporting the new link protection state. In terms of the state machine, WFN state is entered for wrapping because WFN state corresponds to the situation where a span of a station is wrapped but its receive link on that span has a protection status of IDLE.

Row 11.5-15: Protection request greater than MS received from neighbor on the short or long path stating that the neighbor is wrapped or not wrapped. If the current link protection state is MS, then the MS is cleared. The wrap status bit is set if the message is a short path message indicating that the neighbor station is wrapped. Transmission of protection messages is initiated reporting the new link protection state.

Row 11.5-16: Protection request received from non-neighbor station on the short or long path stating that the non-neighbor station is wrapped, and the request is high enough in the hierarchy to preempt local protection. This results in the clearing of the MS condition and clearing of any existing wrap. Transmission of protection messages is initiated reporting the new link protection state.

Row 11.5-17: Protection request from the other interface of the station preempts protection on this interface.This results in the clearing of the MS condition and clearing of any existing wrap. Transmission of protection messages is initiated reporting the new link protection state.

Row 11.5-18: sidePendingReq is cleared.

Row 11.5-19: Remain in FS/MS state.

Row 11.5-20: SF/SD cleared and sidePendingReq is IDLE. This results in reporting of WTR for this link.

Row 11.5-21: SF/SD cleared and sidePendingReq is granted. This results in reporting of WTR for this link.

Row 11.5-22: SF/SD cleared and sidePendingReq is not granted. This results in reporting of WTR for this link.

Row 11.5-23: Local administrative request granted. Causes existing local failure condition to go pending. Causes transmission of protection messages reporting the new link protection state, and wrap.

Row 11.5-24: Wrap only. FS protection request received from neighbor on the short path stating that the neighbor is wrapped, and local link protection state is not SF. Causes transmission of protection messages reporting the new link protection state, and wrap.

Row 11.5-25: Wrap only. Protection request received from non-neighbor station on the short or long path stating that the non-neighbor station is wrapped or not wrapped, and the request is high enough in the hierarchy to preempt local protection. This results in the clearing of any existing wrap. Transmission of protection messages is initiated reporting that the wrap has been removed.

Row 11.5-26: Wrap only. Protection request received from the other interface of the station that preempts protection on this interface. This results in the clearing of any existing wrap. Transmission of protection messages is initiated reporting that the wrap has been removed.

Row 11.5-27: Remain in SF/SD state.

Row 11.5-28: Wrap only. IDLE protection request received from neighbor on the short or long path stating that the neighbor is no longer wrapped. Causes transmission of protection messages reporting the new link protection state, and clearing of the existing wrap.

Row 11.5-29: Wrap only. Local administrative request granted. Causes transmission of protection messages reporting the new link protection state, and wrap.

Row 11.5-30: Wrap only. Local failure causes wrap. Causes transmission of protection messages reporting the new link protection state, and wrap.

Row 11.5-31: Wrap only. Local failure does not cause wrap. Causes transmission of protection messages reporting the new link protection state.

Row 11.5-32: Wrap only. Remain in WFN state.

Row 11.5-33: Wrap only. IDLE protection request received from neighbor on the short or long path stating that the neighbor is wrapped, and local WTR has expired. Causes transmission of protection messages reporting the a new link protection state of IDLE, and clearing of the existing wrap.

Row 11.5-34: Wrap only. Protection request greater than WTR received from neighbor on the short or long path stating that the neighbor is wrapped. Causes transmission of protection messages reporting the a new link protection state of IDLE (clearing of WTR).

Row 11.5-35: Ring is in steering mode and WTR expires. Causes transmission of protection messages reporting the a new link protection state of IDLE.

Row 11.5-36: Protection request of IDLE received from new neighbor on the short path stating that the neighbor is not wrapped. Causes transmission of protection messages reporting the a new link protection state of IDLE (clearing of WTR), and clearing of any existing wrap.

Row 11.5-37: Local administrative request granted. Causes transmission of protection messages reporting the new link protection state, and wrap.

Row 11.5-38: Local failure causes wrap. Causes transmission of protection messages reporting the new link protection state, and wrap.

Row 11.5-39: Protection request received from non-neighbor station on the short or long path stating that the non-neighbor station is wrapped or not wrapped, and the request is high enough in the hierarchy to pre-empt local protection. This results in the clearing of the WTR and of any existing wrap. Transmission of protection messages is initiated reporting the clearing of WTR and that the wrap has been removed.

Row 11.5-40: Wrap only. Protection request received from the other interface of the station that preempts protection on this interface. This results in the clearing of the WTR and of any existing wrap. Transmission of protection messages is initiated reporting the clearing of WTR and that the wrap has been removed.

Row 11.5-41: Remain in WTR state.

12. Operations, administration and maintenance (OAM)

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	First draft for P802.17 WG review.
Draft 0.2, April 2002	Second draft for TF review, modified according to WG comments.
Draft 0.3, June 2002	Draft 0.3 for WG review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for TF review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1

12.1 Scope

This section provides OAM (operations, administration and maintenance) 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 region of Figure 12.1.

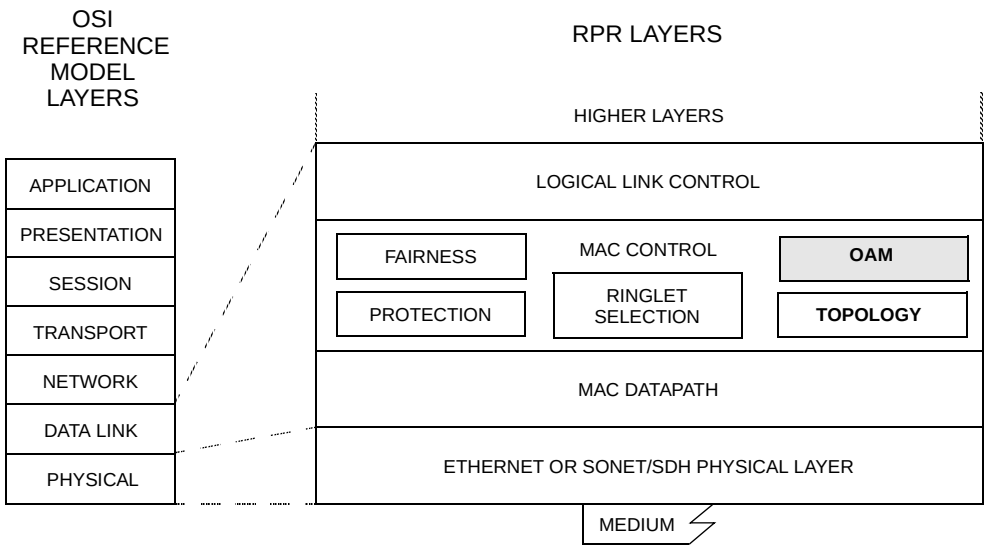


Figure 12.1—RPR layer diagram

The services and features provided are:

a) Determine/validate connectivity between any two station on the ring
b) Determine/validate transmit path operation for single and dual queue implementations
c) Determine/validate transit path operation for any service class
d) Operate without any master station in the ring
e) Provide mechanism to help in misorder prevention

f) Cause insignificant overhead

It is not within the scope of the OAM functions to detect intermittent failures.

12.2 Overview

Management Functional Areas pertinent to RPR are:

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 (configuration) and provisioning.

Fault (or Maintenance) Management enables the detection, isolation, and correction of abnormal operation of the stations and its 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 of stations and the effectiveness of the network and stations for the support of services. 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 envisaged.

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, while a physical layer based on Ethernet PHYs has a lower level of OAM support.

This standard re-uses 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.

12.2.1 OAM functions supported by RPR

The OAM function in RPR is based on special frames sent between stations on a ring. These frames are used to test the operational status of a path between stations.

OAM frames are used to perform echo request/response operations and they serve also as a method to avoid frame misorder when changing the connection path.

The OAM frame types supported by RPR are:

a) Echo - For on demand connectivity monitoring and fault localization on path between stations
b) Flush - For misorder prevention and determining RTT.

All RPR compliant stations shall support echo request/response frames and flush frames.

12.3 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 RPR stations should 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 may 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. For RPR, in case of fiber cut or station failure, a protection switching is used to restore service.

Testing performs repair functions using some 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 and can be activated upon request.

The fault management mechanisms are useful to check the reachability at the MAC layer between two RPR stations on the ring, especially when there are some failures (e.g. one station on the ring steals frames addressed to some other stations) that are not detected at layer 1.

This section defines one Fault Management mechanism. It is an on-demand in-service echo request/response mechanism (see 12.3.1) used for troubleshooting the RPR network (reactive mechanism).

In addition, echo frames may be used for fault detection (proactive mechanism) by continuously running to verify connectivity between any pair of stations. This is described in Annex K.

12.3.1 RPR echo request/response capability

This standard allows the management system to request an echo request/response operation to a specified destination in order to check the reachability of an RPR station. Figure 12.2 shows an example of this operation.

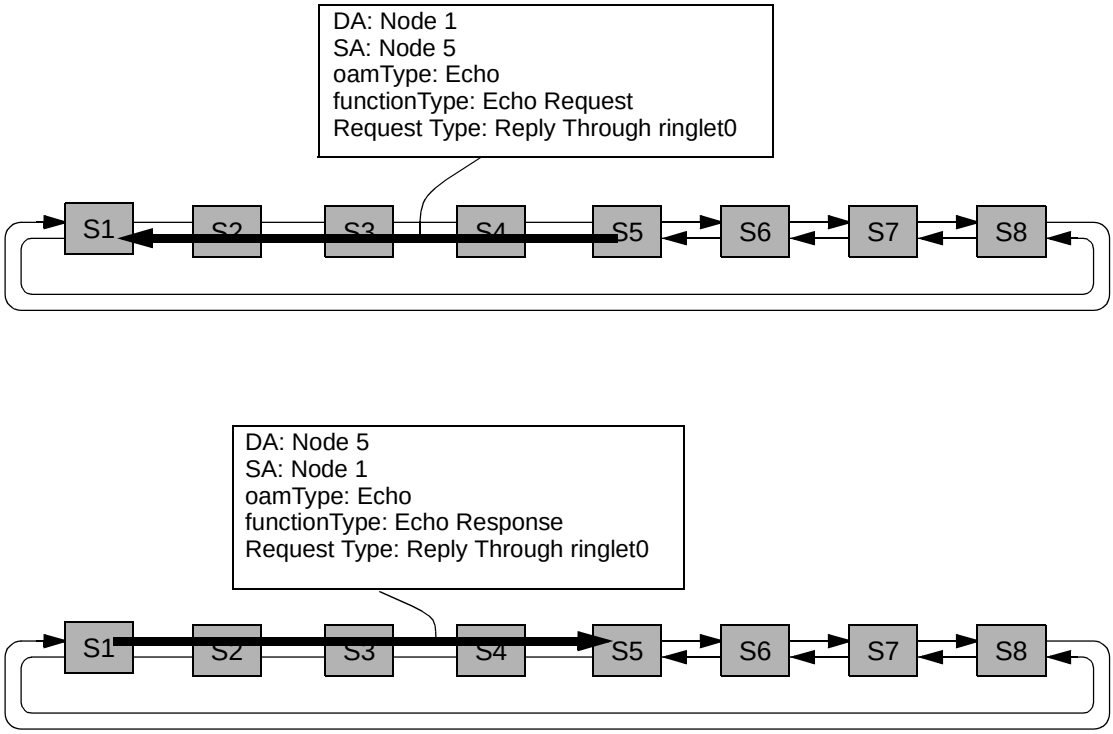


Figure 12.2—Echo request/response example

The RPR echo request capability allows for 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, without impairing the data flow between stations. Echo request/response frames can be assigned any service class. The echo request frame may contain any number of user specified bytes up to the maximum permitted frame size, and the userData is copied into the reply frame.

The echo request/response command can be sent through the default ringlet, ringlet0 or ringlet1. An example of these options is shown in Figure 12.3.

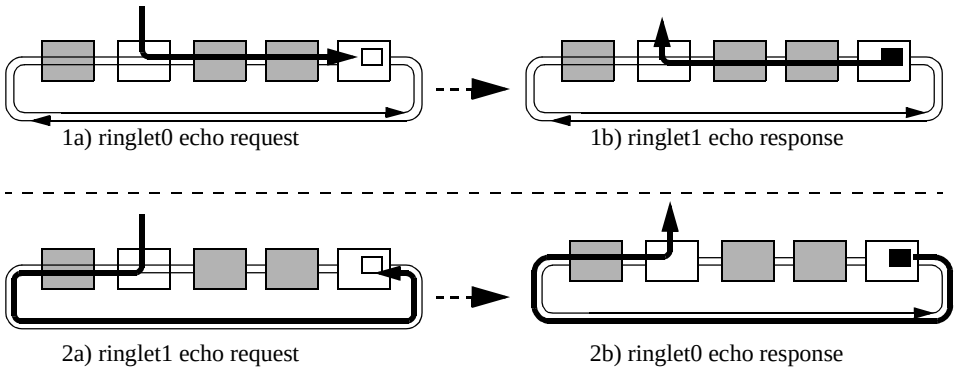


Figure 12.3—Default echo ringlet routing

12.3.1.1 Echo source station operation

The echo request source station shall set:

a) The DA to the echo response target MAC address
b) The SA to its own MAC address
c) The functionType to echo request command
d) The service class
e) The response control field to indicate the desired ringlet, class of service, and protectionMode for the Echo response frame
f) Validate the DA is a ring local unicast address and not the local stations address
g) Insert the provided userData in the frame

The ringlet selection shall be as specified in 6.2.

12.3.1.2 Echo target station operation

The Echo target station shall perform the following operations:

a) Set the functionType to echo response
b) Change the SA to its MAC address
c) Set the DA to the original echo request frame SA
d) Change the ring ID (only if the response goes back on a different ring)
e) Copy all other received bytes to the transmit frame
f) Echo the resulting frame.

The echo request/response command can require the target station to reply either on:

a) the default ringlet,
b) the ringlet0,
c) the ringlet1.

The target station shall select the transmission ringlet according to the protection field as specified in 6.3

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 frames.

12.3.2 RPR Flush capability

The RPR flush capability allows for a frame to be inserted by any station in the ring, to travel around the ring and arrive back to the source station. Flush frames can be activated for each service class and for each ringlet. The flush frame may contain any number of user specified bytes up to the maximum permitted frame size.

The control in a station may request a flush operation to prevent frame misorder when changing the selected ringlet for a given flow. Figure 12.4 shows an example of this operation.

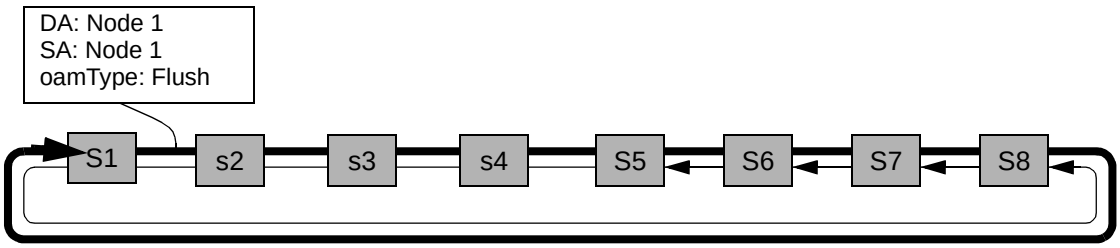


Figure 12.4—Flush operation example

The ringlet selection shall be as specified in 6.3

12.4 OAM frame handling during failures

OAM frame handling during failures shall comply with the protection schemes described in 6.3.

Echo request or echo response frames can be sent on the default ringlet or on a specified ringlet, and they may be protected or unprotected.

OAM frames shall not be transmitted over failed links even if the failure is unidirectional.

12.5 OAM frame

RPR OAM uses frames in the control frame format as described in Section 8.3. Figure 12.5 illustrates the OAM control payload.

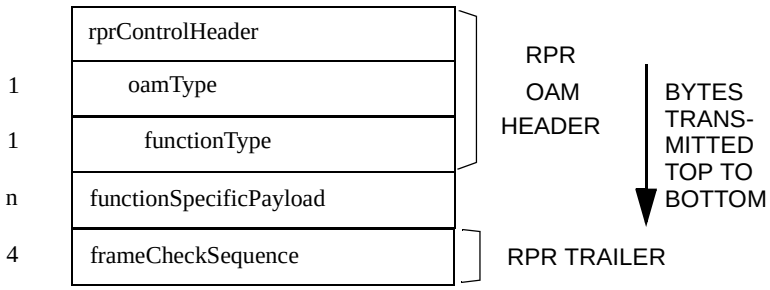


Figure 12.5—OAM frame format

OAM frames are control frames, and as such the minimum length of an OAM frame shall be 24 bytes. If necessary, padding shall be added to the OAM frame payload to comply with this requirement.

12.5.1 OAM Class Of Service

OAM service class is indicated in the SC field. OAM frames can be sent for any service class.

12.5.2 oamType

The oamType identifies the OAM group of the OAM frame. Table 12.1 shows the possible values of the oamType field.

12.5.3 functionType

This field indicates the actual function performed by this frame within the group indicated by the oamType. Table 12.1 shows the possible values of the functionType field.

Table 12.1—oamType field values

oamType	Coding	functionType	Code
fault management	0001_2	echo request	1000_2
		echo response	1001_2
		flush	0001_2

12.5.4 Specific fields for OAM frames

The definition of the specific fields for the different OAM frames are provided in the subclauses that follow.

12.5.4.1 Echo request/response frame

The function specific fields for echo request/response frames are illustrated in Figure 12.6.

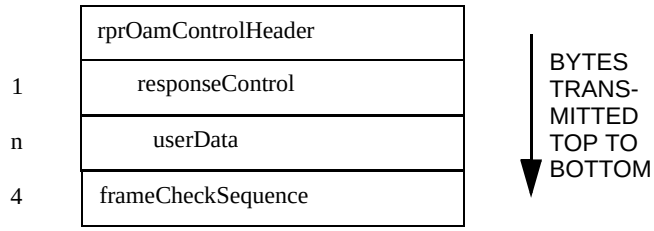


Figure 12.6—Echo request/response frame format

The response control field is shown in Figure 12.7

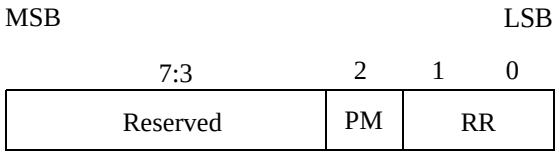


Figure 12.7—Response control field format

The subfields of the response control fields are described in the subclauses below.

12.5.4.1.1 responseRinglet (RR)

This 2 bit field shall be interpreted by the receiving station to decide through which ringlet the echo response frame should be transmitted back to the source station. The responseRinglet (RR) field shall be encoded as shown in Table 12.2.

Table 12.2— responseRinglet values

Value	Description
0	Reply on default ringlet
1	Reply on ringlet0
2	Reply on ringlet1

12.5.4.1.2 protectionMode (PM)

This 1 bit field shall be interpreted by the receiving station to decide through which ringlet the echo response frame should be transmitted back to the source station in case of a ring failure. The protectionMode (PM) field shall be encoded as shown in Table 12.3.

Table 12.3—protectionMode values

Value	Description
0	Protected
1	Unprotected

12.5.4.1.3 userData

This is a variable length optional field. The contents of this field are not defined by this standard.

User specific data must be copied by the receiving station from the echo request frame to the echo response frame.

12.5.4.2 Flush frame

The function specific fields for Flush frames are illustrated in Figure 12.8.

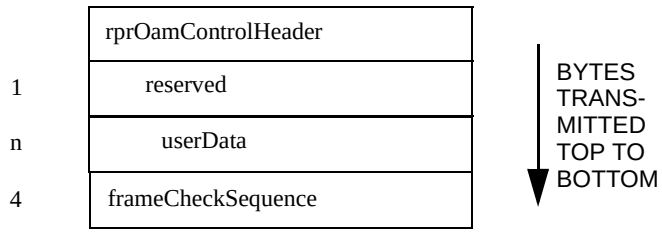


Figure 12.8—Flush frame format

The first byte is reserved for future use.

12.5.4.2.1 userData

This is a variable length optional field. The contents of this field are not defined by this standard.

12.6 OAM frame detection procedure

OAM frames are detected through the following procedure (no specific ordering is implied):

a) Check RPR header to determine if it is a control frame of type OAM, and if it is for this station
b) Check the oamType and functionType values according to Table 12.1 to determine the type of OAM frame received
c) Discard OAM frames with unsupported oamType and/or functionType.
d) Report this discard to station OAM

13. Layer management entity interface

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for P802.17 WG review.
Draft 0.2, April 2002	Revised according to WG comments for TF review.
Draft 0.3, June 2002	Draft 0.3 for WG review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for TF review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1

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) must 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 would typically perform such functions on behalf of general system management entities and would implement standard management protocols. Figure 13.1 depicts the relationship among management entities.

The management service interface within this model is the SME-MLME service interface.

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.

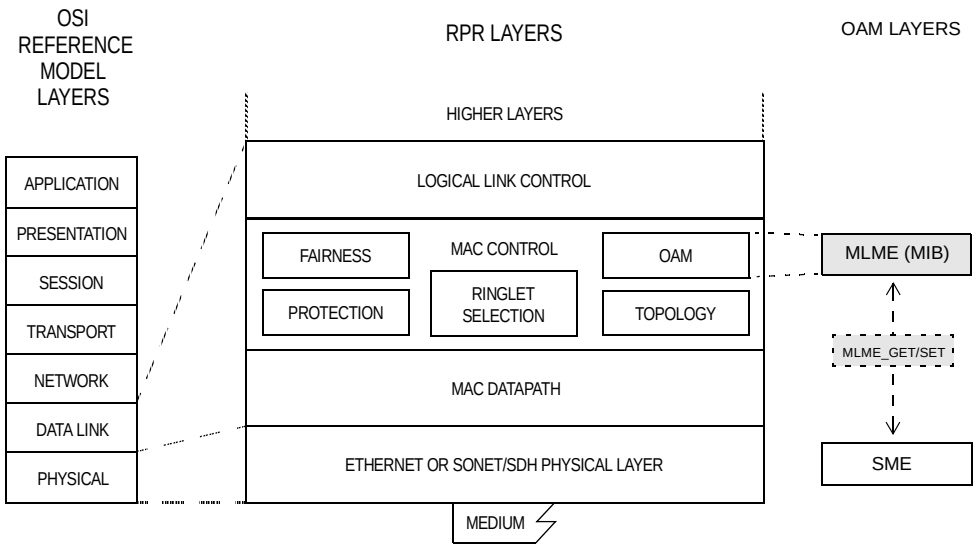


Figure 13.1—RPR layer diagram & OAM

13.2 Generic management primitives

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 “containing” 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, or to SET the value of a MIB attribute. The invocation of a SET.request primitive may require that the layer entity perform certain defined actions.

The following two primitives are defined for the MAC layer management:

— MLME-GET.request
— MLME-SET.request

The primitives MLME-GET.request and MLME-SET.request described in this subclause are mandatory.

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

```
MLME-GET.request (
    MIBattribute,
    MIBattributevalue,
    status
)
```

13.2.1.3 When generated

This primitive is generated by a MAC client 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

```
MLME-SET.request (
    MIBattribute,
    MIBattributevalue,
    status
)
```

13.2.2.3 When generated

This primitive is generated by a MAC client whenever it wishes to set attributes within 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.

13.3 MLME service interface

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

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 packets to and from the ring.

The RPR interface has its own operational state derived from the combination of:

a) span interface operational state,
b) RPR interface administrative state,
c) RPR interface operational state, affected by:
    1) Keepalive availability
    2) Protection messages from peer station.

The administrative state of an interface is independent of the state of the underlying interfaces. The operational state is defined by the state of the underlying interfaces, local administration and operating state information. For more details see IETF RFC 2863,section 3.1.14 "IfOperStatus in an Interface Stack ".

13.3.2 Topology discovery monitoring

The configuration management should allow monitoring, for maintenance purposes, the state of the auto-configuration and topology discovery protocols. It should also allow disabling the support of some features, even if supported by all the stations on the ring.

The detailed configuration requirements for the auto-configuration 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 Protection switching

Editors' Notes: *To be removed prior to final publication.*

This protection switching portion of the LME was inconsistent with the architecture and approach of the remainder of the LME. As a result this section was been mostly deleted after D0.3. Additional text is requested for this section when the protection details are stable.

The protection management mechanisms that will be 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.

13.3.4 Performance and accounting measurements

Performance and accounting measurements involving counting selected frame flows through ClientIn, ClientOut, SpanIn, or SpanOut locations, illustrated by the ellipses within Figure 13.2. Counter names are based on the measuring location and other frame-content-dependent information.

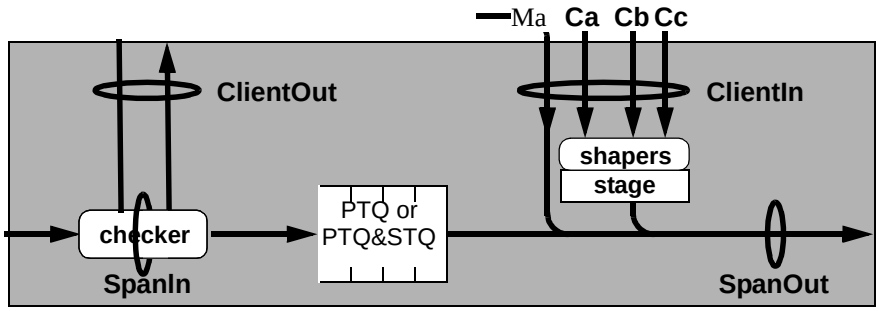


Figure 13.2—Counter measurement points

The RPR statistics that are kept depend on the properties of the processed frame. The general format of a counter name is *<Place><Period><Direction><Address><Class><Units>*, allowing the collection of names to be summarized in Table 13.1 and Table 13.2.

Table 13.1—Statistics definitions

Name	Variable	Values	Description
Location	<loc>	Span, Client	Places where statistics are collected
Reporting Period	<period>	Current Interval, Past Intervals, Total Intervals	Time periods statistics are reported for.
Address Type	<addr>	Unicast, Multicast+Broadcast	Frame types sent/received
Class	<class>	ClassA, ClassB-CIR, ClassB-EIR, ClassC	Service class
Units	<unit>	Octets, Packets	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, UnknownPacketType, PmdAborted	Error type

Table 13.2—MAC statistics

Group	MAC Counter Instances	MAC Counter Names
Mac Client Packets	Reporting Period Direction Address Type Class / Subclass	rprClient<period><dir><addr><class>Pkts *e.g. rprClientCurrentInUcastClassAPkts* rprClientIntervalOutMcastClassCPkts
Mac Client Bytes	Reporting Period Direction Address Type Class / Subclass	rprClient<period><dir><addr><class>Octets *e.g. rprClientCurrentInUcastClassAOctets* rprClientIntervalOutMcastClassCOctets
Span Packets	Reporting Period Direction Address Type Class / Subclass	rprSpan<period><dir><addr><class>Pkts *e.g. rprSpanCurrentInUcastClassAPkts* rprSpanIntervalOutMcastClassCPkts
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. rprSpanCurrentErrorTtlExpPkts* rprSpanIntervalErrorBadHecPkts

13.3.5 Notifications and fault management

Fault management includes alarm surveillance, fault localization, fault correction and testing as described in 12.3. Upon detecting a failure, in addition to generating and sending alarms to systems, network elements may send notifications (in the backward direction) to the peer node to indicate that a failure has occurred (and some action is required).

13.3.6 RPR echo request/response management

An echo request/response operation is described in 12.3.1 that allows sending an ‘echo’ to a specified destination in order to check the reachability of an RPR station. The operation uses the echo request/response primitives to access the OAM echo attributes. This clause introduces the attributes required to support this functionality.

13.3.6.1 Echo request

Using the MLME-SET.request primitive to set the OAM Echo attributes can activate the RPR OAM Echo request. The set of attributes for the OAM echo request operation are specified in Table 13.3.

This primitive is generated by the SME to implement a user request for a station to perform an echo request/response operation.

Table 13.3—Echo request parameters

Attribute	Type	Valid range	Description
Destination Station	MAC Address	Any valid unicast MAC address	The MAC address of the RPR station to be sent an echo request
Request Ringlet	Enumeration	RINGLET 0, RINGLET 1 DEFAULT	The ringlet over which the RPR echo request message should be sent
Response Ringlet	Enumeration	RINGLET 0, RINGLET 1 DEFAULT	The ringlet over which the addressed station should send the RPR echo response message
Class	Enumeration	CLASS A, ClassB-CIR, ClassB-EIR, CLASS C	The service class to be used in the RPR OAM frames carrying the RPR echo request and reply messages.
Control	Enumeration	IDLE, ACTIVE ABORT	Control of the OAM echo process
Protection	Enumeration	PROTECTED, UNPROTECTED	If protection is applied on OAM echo request/response

13.3.6.1.1 Effect of receipt

This request causes the station to send an OAM RPR echo request message.

13.3.6.2 Echo response

The MLME-GET.request primitive enables the MLME to return the result status of the OAM echo request. It is not generated until the OAM RPR echo response frame is received or the timer expires. The set of attributes for OAM Echo confirmation are specified in Table 13.4.

Table 13.4—Echo response parameters

Name	Type	Valid range	Description
Status	Enumeration	SUCCESS, INVALID_PARAMETERS, FAILURE	Indicates the result of the echo request

This primitive is generated by the MLME as a result of an MLME-SET.request by the SME to send an echo.

13.3.6.2.1 Effect of receipt

The SME is notified of the success or of the failure of the echo request/response procedure.

Annex A

(informative)

Bibliography

Editors' Notes: To be removed prior to final publication.

References:
None

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for WG review.
Draft 0.2, April 2002	Draft 0.2 for TF review, modified according to comments on D0.1.
Draft 0.3, June 2002	Draft 0.3 for TF review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for WG review, no comments from D0.3, no submissions
Draft 1.1, October 2002	Draft 1.1 for WG review, no comments from D1.0, no submissions
Draft 2.0, December 2002	Draft 2.0 for WG ballot, no comments from D1.1, no submissions

Annex B

(normative)

Ethernet reconciliation sublayers

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	Revision for RPR TF review.
Draft 0.3, June 2002	Revised draft for WG review.
Draft 1.0, August 2002	Revised draft for TF review.
Draft 1.1, October 2002	Revised draft for WG review.
Draft 2.0, December 2002	Revised draft for WG ballot.

B.1 Overview

B.1.1 Scope

This annex defines two reconciliation sublayers for use with Ethernet physical layer entities (PHYs). The first is a Gigabit Ethernet Reconciliation Sublayer (GERS) that maps the logical primitives at the RPR MAC physical layer service interface to the Gigabit Media Independent Interface (GMII), for use with gigabit Ethernet PHYs. The GMII and the gigabit Ethernet PHYs are defined in IEEE Std 802.3-2000. The second is a 10 Gigabit Ethernet Reconciliation Sublayer (XGERS) that maps the logical primitives at the RPR MAC physical layer service interface to the 10 Gigabit Media Independent Interface (XGMII), for use with 10 gigabit Ethernet PHYs. Optionally, an XGMII Extender Sublayer (XGXS) may be used to provide a 10 Gigabit Attachment Unit Interface (XAUI) instead of the XGMII. The XAUI, XGMII, XGXS, and 10 gigabit Ethernet PHYs are defined in IEEE Draft P802.3ae/D5.0.

B.2 Gigabit Ethernet Reconciliation Sublayer (GERS)

The Gigabit Ethernet Reconciliation Sublayer (GERS) converts the logical physical layer service interface of the RPR MAC to the Gigabit Media Independent Interface (GMII) defined in IEEE Std 802.3-2000.

B.2.1 General requirements

B.2.1.1 Summary of major concepts

a) The GERS 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 GERS accepts frames from the MAC, inserts preamble and interpacket gap, and generates corresponding signals on the GMII;
d) In the receive direction, the GERS 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 GERS.

B.2.1.2 Relationship to other sublayers

The relationship of the GERS to RPR and IEEE Std 802.3 sublayers is shown in Figure B.1.

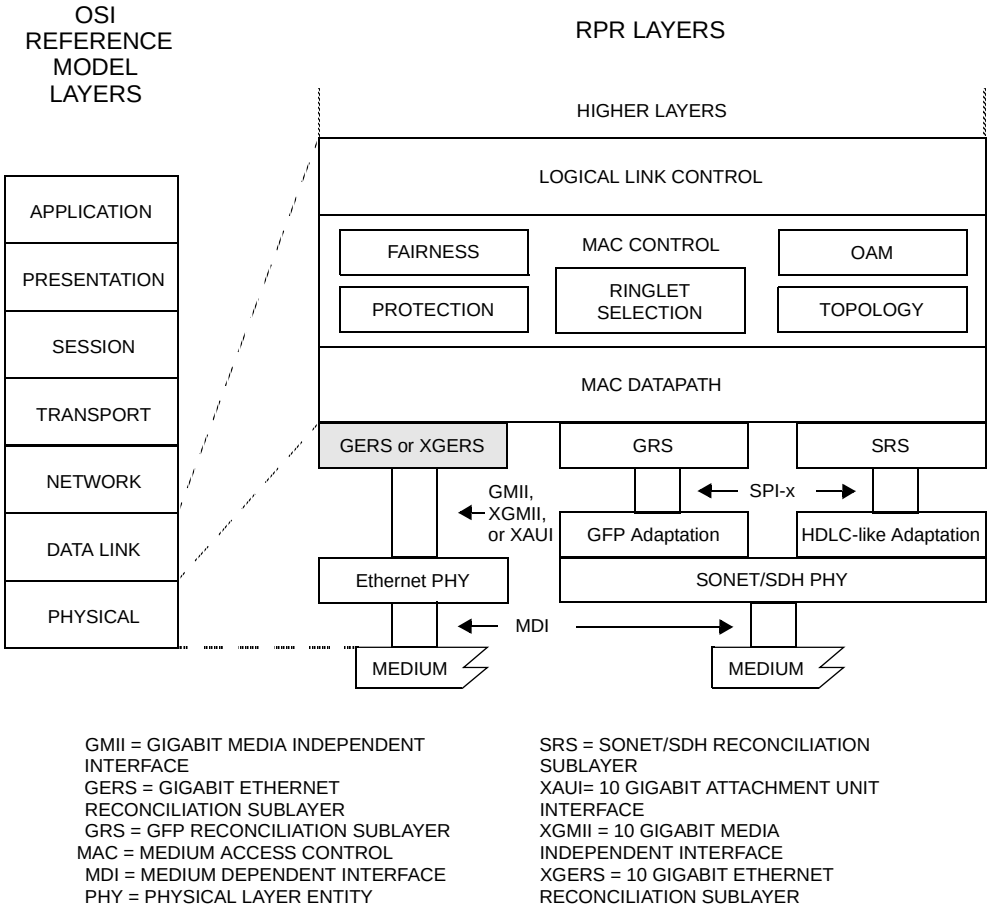


Figure B.1—Reconciliation Sublayer relationship to the ISO/IEC Open Systems Interconnection (OSI) reference model

B.2.1.3 Rate of operation

The GMII provided by the GERS is capable of supporting a data rate of 1000 Mbps. This does not correspond to a fixed MAC data rate, because preamble and interpacket gap are generated by the GERS.

Editors' Notes: *To be removed prior to final publication.*

A requirement for "delay constraints" appears to be needed (similar to IEEE 802.3), because an RPR network has similar requirements to ensure stable and predictable operation.

Editors' Notes: *To be removed prior to final publication.*

Draft 1.0 comment #425 requests that each RS specify a "LINK_RATE" value. However, it is unclear if this is intended to correspond to the line rate, nominal data rate, or other rate. The resolution to Draft 1.0 comment #345 asks the rate ad-hoc to consider these issues before new text is added to the annexes.

B.2.1.4 GMII structure

The GMII is composed of independent transmit and receive paths, and a management interface controlled by a station management entity. The transmit data path consists of the following signals:

TXD<7:0> are the transmit data bits. TXD<0> represents the least significant data bit, and is the first bit of each byte transmitted by the PHY. TXD<7> represents the most significant data bit, and is the last bit of each byte transmitted by the PHY. This is shown pictorially in Figure B.2
TX_EN is a transmit enable signal, and is asserted when valid data is present on TXD<7:0>;
TX_ER is a transmit error signal, and is used to propagate errors and control carrier extension;
GTX_CLK is a continuous transmit clock provided by the reconciliation sublayer;
COL is a collision detect signal and is not used by this standard.

The receive data path consists of the following signals:

RXD<7:0> are the receive data bits. RXD<0> represents the least significant data bit, and is the first bit of each byte received by the PHY. RXD<7> represents the most significant data bit, and is the last bit of each byte received by the PHY. This is shown pictorially in Figure B.2;
RX_DV is a received data valid signal, and is asserted when valid data is available on RXD<7:0>;
RX_ER is a received error signal, and is used to indicate that an error was detected in the frame currently being transferred by RXD<7:0>;

RX_CLK is a continuous receive clock provided by the PHY;
CRS is a carrier sense signal and is not used by this standard.

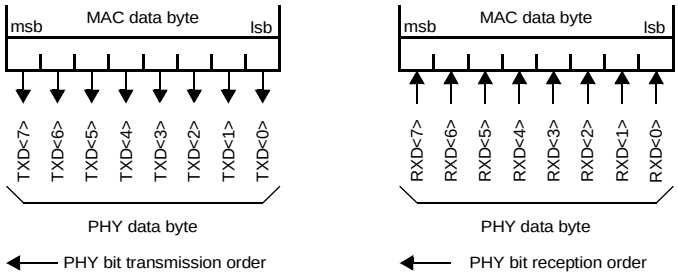


Figure B.2—GERS transmit and receive signal mapping

The GMII includes carrier sense (CRS) and collision detected (COL) signals that are not defined for the GERS since only full duplex PHYs are supported. The GMII also includes a management interface consisting of bi-directional data (MDIO) and clock (MDC) signals. All of the GMII signals are fully defined in IEEE Std 802.3-2000, Clause 35. Figure B.3 shows a schematic view of the GERS inputs and outputs.

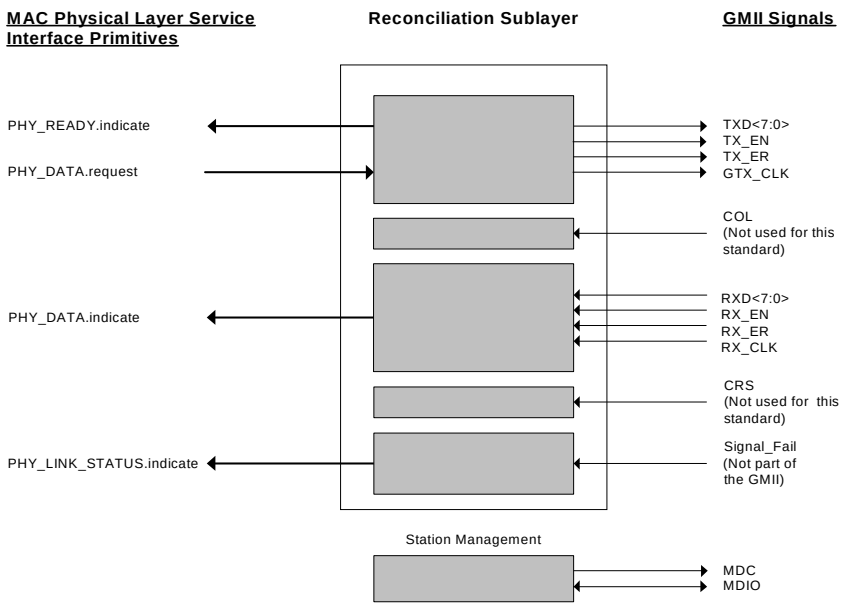


Figure B.3—GERS inputs and outputs

B.2.1.5 Link status signals

B.2.1.5.1 Signal_fail

In addition to the GMII, a signal_fail signal is used to convey link status from the PHY to the GERS. The GERS shall include a signal_fail 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 802.3-2000, 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 802.3-2000, Clause 22.

The signal_fail signal may not be directly available from the PHY. An RPR system implementation is required to generate the signal_fail signal from the MDIO or from optional signals that may be provided by the PHY.

B.2.1.5.2 Signal_degrade

Signal_degrade is not defined for the GERS.

B.2.1.6 Mapping of GMII and link status signals to service interface primitives

The GERS shall map the signals provided at the GMII and link status signals to the MAC physical layer service interface primitives defined in Clause 7. Mappings for the following primitives are defined:

— PHY_DATA.request
— PHY_DATA.indicate
— PHY_LINK_STATUS.indicate
— PHY_READY.indicate.

B.2.1.6.1 Mapping of PHY_DATA.request

PHY_DATA.request (OUTPUT_FRAME, LENGTH)

The OUTPUT_FRAME is 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 is not present. EXTEND and EXTEND_ERROR functions are not supported. TX_ER is only asserted to indicate the propagation of a transmit error.

The LENGTH parameter is not used by the GERS. The value of this parameter is undefined.

The GERS shall prepend a preamble and generate an interpacket gap for each frame as described in C.2.3.1.

B.2.1.6.2 Mapping of PHY_DATA.indicate

PHY_DATA.indicate (INPUT_FRAME, STATUS, LENGTH)

The INPUT_FRAME is 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 GERS and the PHY is achieved using the RX_CLK signal.

The STATUS parameter indicates whether an error was conveyed by the PHY during frame reception. An error from the GMII corresponds to both RX_DV and RX_ER asserted during frame reception. If an error is indicated, the STATUS parameter assumes the value ERROR. If no error is indicated, the status parameter assumes the value OK.

The LENGTH parameter is not used by the GERS. The value of this parameter is null.

B.2.1.6.3 Mapping of PHY_LINK_STATUS.indicate

PHY_LINK_STATUS.indicate (LINK_STATUS)

The PHY_LINK_STATUS.indicate service primitive is generated by the GERS to indicate PHY link status. LINK_STATUS assumes the value of OK when the signal_fail signal is deasserted, and the value of FAIL when the signal_fail signal is asserted. DEGRADE is not generated by the GERS.

B.2.1.6.4 Mapping of PHY_READY.indicate

PHY_READY.indicate (READY_STATUS)

PHY_READY.indicate is generated in response to PHY_DATA.request to indicate the beginning of frame transmission by the GMII, and the completion of frame transmission by the GMII. READY_STATUS assumes the value NOT_READY if a frame has been received from the MAC, and a preamble, the frame, and a minimum interpacket gap have not been fully transmitted by the GMII. READY_STATUS assumes the value READY any time a preamble, frame, or minimum interpacket gap are not being transmitted.

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 IEEE Std 802.3-2000, Clause 35, section 35.2.3, with the exception of sections 35.2.3.1 and 35.2.3.5. Carrier extension is not defined for the GERS.

B.2.3 Functional specifications

B.2.3.1 Transmit

B.2.3.1.1 General requirements

The GERS shall meet the transmit functional requirements specified in IEEE Std 802.3-2000, Clause 35, sections 35.2.2.1, 35.2.2.3, 35.2.2.4, and 35.2.2.5.

B.2.3.1.2 Preamble

The RPR MAC frame does not contain a preamble. The GERS shall prepend a preamble to each frame transmitted on the GMII in accordance with IEEE Std 802.3-2000, Clause 35, section 35.2.3.2.1.

The reconciliation sublayer requests the MAC to pause between the transmission of frames to allow insertion of the preamble using the PHY_READY.indicate primitive.

B.2.3.1.3 Interpacket gap

The RPR MAC does not transmit interpacket gap or Idle characters. The GERS shall insert a minimum interpacket gap in accordance with IEEE Std 802.3-2000, Clause 4 for operation at 1000 Mbps.

The reconciliation sublayer requests the MAC to pause between the transmission of frames to allow insertion of the interpacket gap using the PHY_READY.indicate primitive.

B.2.3.2 Receive

The GERS shall meet the receive functional requirements specified in IEEE Std 802.3-2000, Clause 35, sections 35.2.2.2, 35.2.2.6, 35.2.2.7, and 35.2.2.8.

B.2.3.3 Error and fault handling

The GERS shall meet the error and fault handling functional requirements specified in IEEE Std 802.3-2000, Clause 35, sections 35.2.1.5 and 35.2.1.6.

B.2.3.4 CRS (carrier sense) and COL (collision) signals

The GERS supports only full duplex PHYs. The behavior of the CRS and COL signals is undefined.

B.2.4 Electrical characteristics

B.2.4.1 GMII

The GMII provided by the GERS shall meet the electrical requirements of IEEE Std 802.3-2000, Clause 35, section 35.4.

B.2.4.2 Link status signals

The link status signals shall meet the electrical characteristics specified in IEEE Std 802.3-2000, Clause 35, for the GMII MDIO and MDC signals.

B.3 10 Gigabit Ethernet Reconciliation Sublayer (XGERS)

B.3.1 General requirements

The 10 Gigabit Ethernet Reconciliation Sublayer (XGERS) converts the logical physical layer service interface of the RPR MAC to the 10 Gigabit Media Independent Interface (XGMII). Optionally, an XGMII Extender Sublayer (XGXS) may be used to provide a 10 Gigabit Attachment Unit Interface (XAUI) instead of the XGMII. The XAUI, XGMII, and XGXS are defined in IEEE Draft P802.3ae/D5.0. Though the XGMII is an optional interface, it is used as a basis for specification of the XGERS, 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 XGERS 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 XGERS 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 XGERS 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 XGERS participates in link fault detection and reporting;
f) The XGERS may use an XGXS to provide a XAUI instead of the XGMII.

Editors' Notes: *To be removed prior to final publication.*

The following note was included in the previous version of Figure C.3, which was replaced with a new layer diagram by a motion passed at the July 2002 plenary meeting:
"10 Gigabit LAN and WAN PHYs operate at different data rates. An RPR MAC management variable (to be defined) is used to select the appropriate data rate."

Editors' Notes: *To be removed prior to final publication.*

D0_3 comment 648 requested addition of the 10 GbE XGXS/XAUI sublayers to the previous version of Figure C.3, which was replaced with a new layer diagram by a motion passed at the July 2002 plenary meeting. However, the newer layer diagram no longer shows individual 802.3 sublayers. Commenter is requested to re-submit the comment if deemed to still be applicable.

B.3.1.2 Relationship to other sublayers

The relationship of the XGERS to RPR and IEEE Draft P802.3ae/D5.0 sublayers is shown in Figure B.4

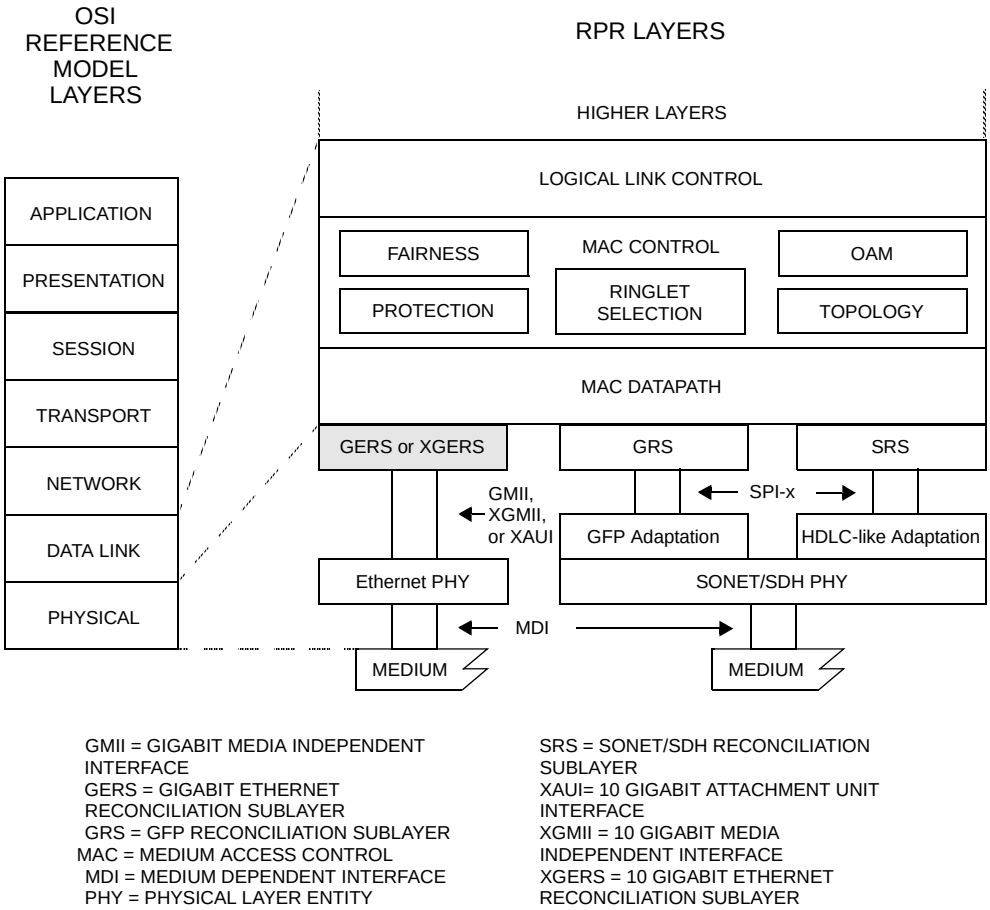


Figure B.4—Reconciliation Sublayer relationship to the ISO/IEC Open Systems Interconnection (OSI) reference model

B.3.1.3 Rate of operation

The XGMII provided by the XGERS transfers data or control characters at a fixed rate of 10.0 Gbps. This does not correspond to a fixed MAC data rate, because preamble and interpacket gap are generated by the XGERS.

Editors' Notes: *To be removed prior to final publication.*

A requirement for "delay constraints" appears to be needed (similar to IEEE 802.3), because an RPR network has similar requirements to ensure stable and predictable operation.

Editors' Notes: *To be removed prior to final publication.*

Draft 1.0 comment #425 requests that each RS specify a "LINK_RATE" value. However, it is unclear if this is intended to correspond to the line rate, nominal data rate, or other rate. The resolution to Draft 1.0 comment #345 asks the rate ad-hoc to consider these issues before new text is added to the annexes.

B.3.1.4 XGMII structure

The XGMII is composed of independent transmit and receive paths. The transmit data path consists of the following signals:

TXD<31:0> represent a group of four bytes used to transmit data or control characters. TXD<0> represents the least significant bit, and is the first bit of each group of four bytes transmitted by the PHY. TXD<31> represents the most significant bit, and is the last bit of each group of four bytes transmitted by the PHY. This is shown pictorially in Figure B.5
TXC<3:0> are control signals that indicate whether the four bytes conveyed by TXD<31:0> represent 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 is a continuous transmit clock provided by the reconciliation sublayer. A group of four data bytes or control characters are conveyed from the reconciliation sublayer to the PHY on each edge of TX_CLK.

The receive data path consists of the following signals:

RXD<31:0> represent a group of four bytes used to receive data or control characters. RXD<0> represents the least significant bit, and is the first bit of each group of four bytes received by the PHY. RXD<31> represents the most significant bit, and is the last bit of each group of four bytes received by the PHY. This is shown pictorially in Figure B.6
RXC<3:0> are control signals that indicate whether the four bytes conveyed by RXD<31:0> represent 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 is a continuous receive clock provided by the PHY. A group of four data bytes or control characters are conveyed from the PHY to the reconciliation sublayer on each edge of RX_CLK.

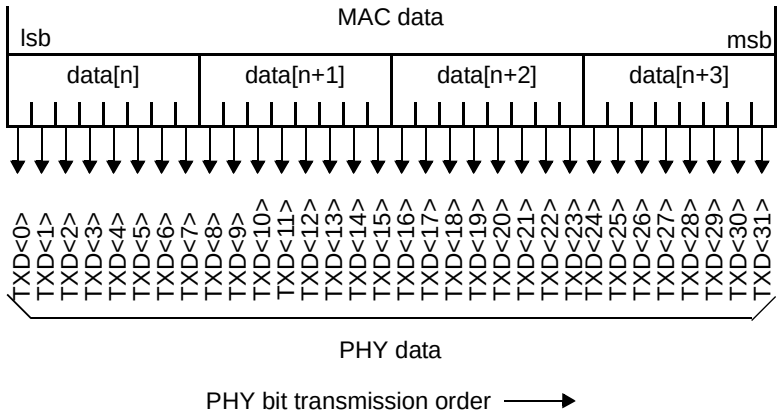


Figure B.5—XGERS transmit signal mapping

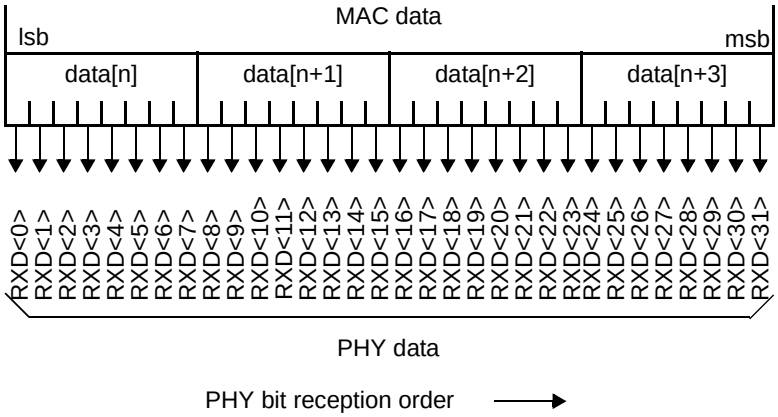


Figure B.6—XGERS receive signal mapping

The 32 data and four control signals in each direction shall be organized into four lanes, as shown in Table C—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, etc. 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.

Table B.1—Transmit and receive lane associations

TXD RXD	TXC RXD	Lane
<7:0>	<0>	0
<15:8>	<1>	1
<23:16>	<2>	2
<32:24>	<3>	3

The XGMII signals are fully defined in IEEE Draft P802.3ae/D5.0, Clause 46. Figure B.7 shows a schematic view of the XGERS inputs and outputs.

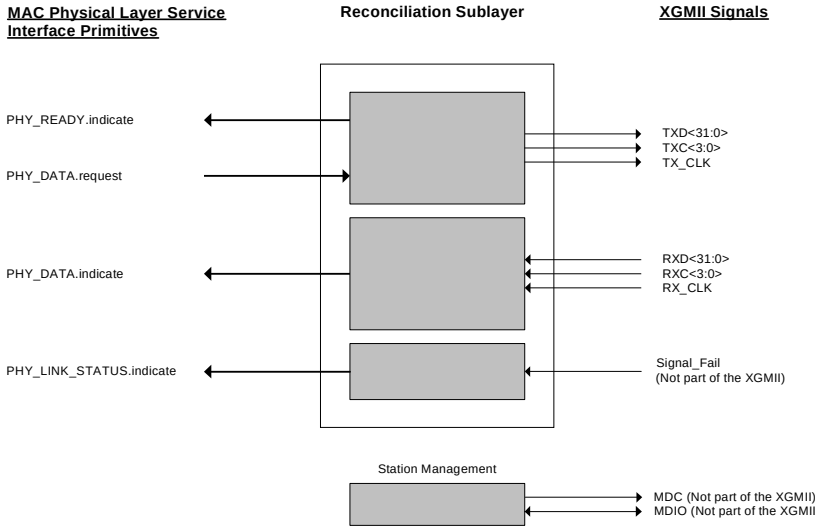


Figure B.7—XGERS inputs and outputs

B.3.1.5 Link status signals

B.3.1.5.1 Signal_fail

In addition to the XGMII, a signal_fail signal is used to convey link status from the PHY to the XGERS. The XGERS shall include a signal_fail 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 Draft P802.3ae/D5.0, 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 Draft P802.3ae/D5.0, Clause 45.2.1.2.

The signal_fail 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 may generate the signal_fail signal from the MDIO or from optional signals that may be provided by the PHY.

B.3.1.5.2 Signal_degrade

Signal_degrade is not defined for the XGERS.

B.3.1.6 Mapping of XGMII signals to MAC physical layer service interface primitives

The XGERS shall map the signals provided at the XGMII to the MAC physical layer service interface primitives defined in Clause 7. Mappings for the following primitives are defined:

— PHY_DATA.request
— PHY_DATA.indicate
— PHY_LINK_STATUS.indicate
— PHY_READY.indicate.

B.3.1.6.1 Mapping of PHY_DATA.request

PHY_DATA.request (OUTPUT_FRAME, LENGTH)

The OUTPUT_FRAME is 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 XGERS shall prepend a Start control character and preamble to each frame as described in C.3.3.1.2, and append a Terminate control character and Idle control characters to each frame as described in C.3.3.1.3.

The LENGTH parameter is not used by the XGERS. The value of this parameter is undefined.

B.3.1.6.2 Mapping of PHY_DATA.indicate

PHY_DATA.indicate (INPUT_FRAME, length)

The INPUT_FRAME is 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 seven bytes following a valid Start control character, and continuing until a Terminate control character. The Start control character, preamble, Terminate control character, and Idle control characters are not part of an RPR frame, and are removed prior to transfer of a frame to the MAC.

To assure robust operation, the value of the data transferred to the MAC may be changed by the XGERS as required by XGMII error indications (see C.3.3.3). Sequence ordered_sets are not indicated to the MAC (see C.3.3.3).

The LENGTH parameter is not used by the XGERS. The value of this parameter is null.

B.3.1.6.3 Mapping of PHY_LINK_STATUS.indicate

PHY_LINK_STATUS.indicate (LINK_STATUS)

The PHY_LINK_STATUS.indicate service primitive shall be generated by the XGERS whenever there is a change in the link_fault variable defined in IEEE Draft P802.3ae/D5.0, Clause 46.3.4, or any time there is a change in the signal_fail signal described in B.3.1.5B.3.1.5 . LINK_STATUS assumes the value OK when the link_fault variable has the value OK and the signal_fail signal is deasserted. LINK_STATUS assumes the value FAIL when the link_fault variable has the values Local Fault or Remote Fault, or if the signal_fail signal is asserted.

DEGRADE is not generated by the XGERS.

B.3.1.6.4 Mapping of PHY_READY.indicate

PHY_READY.indicate (READY_STATUS)

PHY_READY.indicate is generated in response to PHY_DATA.request to indicate the beginning of frame transmission by the XGMII, and the completion of frame transmission by the XGMII. READY_STATUS assumes the value NOT_READY if a frame has been received from the MAC, and a Start control character, preamble, the frame, a Terminate control character, and the minimum interpacket gap have not been fully transmitted by the XGMII. READY_STATUS assumes the value READY any time a Start control character, preamble, a frame, a Terminate control character, or the minimum interpacket gap are not being transmitted.

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 byte or control character.

The XGMII datastream shall meet the requirements of IEEE Draft P802.3ae/D5.0, Clause 46, section 46.2, with the exception of references to the MAC interpacket gap in section 46.2.1. The interpacket gap for the RPR XGERS is defined in C3.3.1.3.

B.3.3 Functional specifications

B.3.3.1 Transmit

B.3.3.1.1 General requirements

The XGERS shall meet the transmit functional requirements specified in IEEE Draft P802.3ae/D5.0, Clause 46, section 46.3.1.1 through 46.3.1.3.

B.3.3.1.2 Preamble

The MAC RPR frame does not contain a preamble. The XGERS shall prepend a preamble including a Start control character to each frame transmitted on the XGMII in accordance with IEEE Draft P802.3ae/D5.0, Clause 46, section 46.2.2.

The reconciliation sublayer requests the MAC to pause between the transmission of frames to allow insertion of the preamble using the PHY_READY.indicate primitive.

B.3.3.1.3 Interpacket gap

The RPR MAC does not transmit interpacket gap or Idle characters. The XGERs shall insert a minimum interpacket gap including the Terminate control character in accordance with IEEE Draft P802.3ae/D5.0, Clause 4 and Clause 46, Section 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.indicate primitive.

B.3.3.2 Receive

The XGERS shall meet the receive functional requirements specified in IEEE Draft P802.3ae/D5.0, Clause 46, section 46.3.2.

B.3.3.3 Error and fault handling

The XGERS shall meet the error and fault handling functional requirements specified in IEEE Draft P802.3ae/D5.0, Clause 46, section 46.3.3.

B.3.4 Electrical characteristics

The XGMII provided by the XGERS shall meet the electrical requirements specified in IEEE Draft P802.3ae/D5.0, Clause 46, section 46.4.

B.3.5 XGXS and XAUI

The XGMII Extender Sublayer (XGXS) may be implemented to provide a 10 Gigabit Attachment Unit Interface (XAUI) instead of the XGMII.

If implemented, the XGXS and XAUI shall meet all requirements specified in Clause 47 of IEEE Draft P802.3ae/D5.0.

Annex C

(normative)

SONET/SDH reconciliation sublayers

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	Revision for RPR WG review.
Draft 0.3, June 2002	Revised draft for TF review.
Draft 1.0, August 2002	Revised draft for TF review.
Draft 2.0, November 2002	Revised draft for WG ballot.

C.1 Overview

C.1.1 Scope

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 8-bit SPI-3, 32-bit SPI-3, SPI-4 Phase 1, and SPI-4 Phase 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 packet length information to the GFP adaptation sublayer thus eliminates the need to calculate this parameter in the PHY device and incur a store and forward delay.

C.1.2 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.

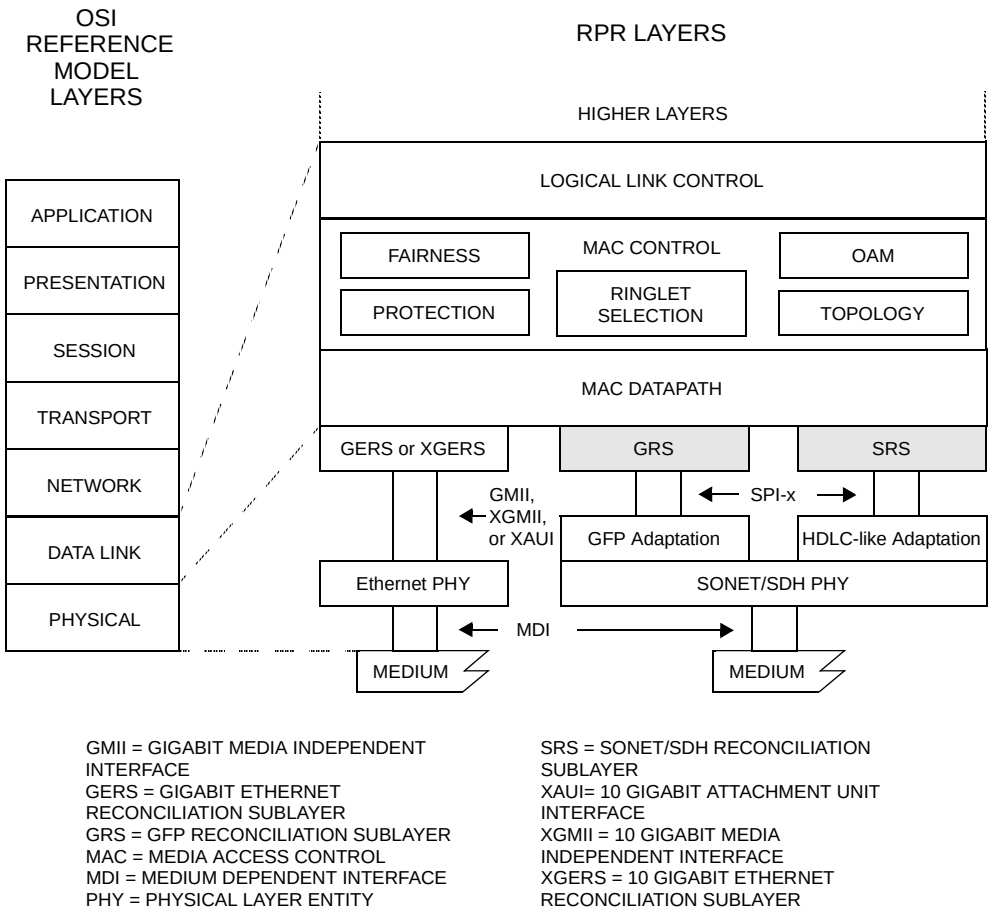


Figure C.1—Reconciliation sublayer relationship to the ISO/IEC Open Systems Interconnection (OSI) reference model

The expected function in the GFP and HDLC-like adaptation sublayer in the PHY device is to perform idle insertion in case of no packet to be sent on the medium in the transmit direction, and idle deletion and packet delineation in the receive direction. Across the SPI interface, only delineated packets are transmitted.

C.1.3 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) using 8-bit transmit and receive data paths and operating at 155 Mbps to 622 Mbps;
b) SPI Level 3 (SPI-3) using 32-bit transmit and receive data paths and operating at 155 Mbps to 2.5 Gbps;
c) SPI Level 4 Phase 1 (SPI-4.1) operating at 200 Mbps to 10 Gbps;
d) SPI Level 4 Phase 2 (SPI-4.2) operating at 622 Mbps to 10 Gbps.

The SRS and GRS for each of the interfaces are electrically identical, except that the GRS conveys packet information. The SRS does not convey packet 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 BW of the attached ringlets.

The figure below shows the designation of the SPI interface with each ringlet connected through a separate SPI interface to its matching MAC datapath entity.

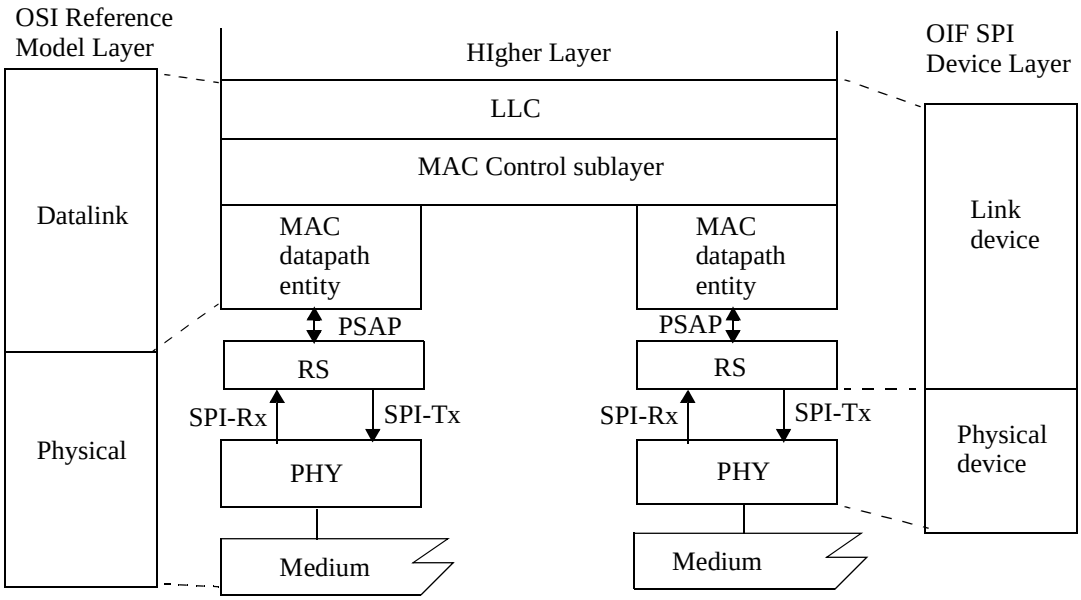


Figure C.2—OSI and OIF reference model layers

C.1.4 Link status signals

In addition to the SPI-3 interface, signal_fail and signal_degrade signals are used to convey link status from the PHY to the reconciliation sublayer.

The signal_fail and signal_degrade signals may not be directly available from the PHY. The RS provides the physical interface and map to physical layer interface primitives, PHY_LINK_STATUS.indicate. A MAC implementation is required to generate these signals based on the specific PHY implementation.

C.1.4.1 Signal_fail

The reconciliation sublayer shall include a signal_fail input. Signal_fail asserted indicates a PHY signal fail condition depending on the medium and the physical medium dependent transmission method signal fail is a loss of optical signal, or loss of carrier. These are just some attributes of signal_fail and it is not comprehensive.

C.1.4.2 Signal_degrade

The reconciliation sublayer shall include a signal_degrade input. Signal_degrade asserted indicates a PHY signal degrade condition. If the PHY provides a signal_degrade status this input is used otherwise, it is not connected and shall be tied to inactive.

C.1.4.3 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 7. Mappings for the following primitives are defined:

— PHY_LINK_STATUS.indicate

The primitive is generated by the physical signal and through the layer management interface. The layer management interface takes precedence over the physical pins.The LINK_STATUS attribute takes on the values: OK, FAILED, and DEGRADED as describe in Clause 7.2.3.2.

C.1.4.3.1 Mapping of layer management MDSF and MDSD PHY_LINK_STATUS.indicate

PHY_LINK_STATUS.indicate (LINK_STATUS)

Editors' Notes: *To be removed prior to final publication.*

coordinate with Layer Management section.

Map the primitive PHY_LINK_STATUS.indicate to the STA signals MDSF and MDSD.

The RS generates the PHY_LINK_OK.indicate primitives whenever the LINK_STATUS parameter changes from one possible value (OK, FAIL, DEGRADE) to a different possible value.

The LINK_STATUS values are derived from the MDSF and MDSD signals received from the Layer Management asynchronously from the data transfer.

The LINK_STATUS parameter assumes the value OK when neither MDSF nor MDSD signal has been received.

It assumes the value FAIL when the MDSF signal is asserted.

The value DEGRADE is assumed when the MDSD signal is asserted and the MDSF has not been received.

C.1.4.3.2 Mapping physical signals to PHY_LINK_STATUS.indicate

PHY_LINK_STATUS.indicate (LINK_STATUS)

There are two physical pins that receives PHY link status if provided by the PHY. These two physical signals are Signal_Fail and Signal_Degrade.

The two physical link status input maps the signals at these pins to the LINK_STATUS attribute.

Table C.1—Signal fail and signal degrade mapping

Signal_Fail	Signal Degrade	LINK_STATUS
inactive	inactive	OK
inactive	active	DEGRADE
active	inactive	FAIL
active	active	DEGARDE and FAIL

C.1.4.3.2.1 When generated

This primitive is invoked by the RS after it determines that the PHY is not performing properly. For example, the received RFCLK has failed. This would cause the SPI receive interface to not be able to receive data. The LINK_STATUS FAIL is also generated up on 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 signal_fail and signal_degrade in Figure C.8.

C.1.4.3.2.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 6.

C.1.5 Electrical specifications

The signal_degrade and signal_fail are intended to be DC status signals. The signal level is LVTTL with threshold dependent on the I/O operating voltage.

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 in GFP adaptation and the HDLC like-adaptation layer is the delineation mechanism and hence different frame format, specifically the header prefix.

C.2.1 SRS physical frame format

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 packet. For consecutive packet transmission the minimum requirement is one escape flag to separate the two packets.

The HDLC-like adaptation sublayer, thus has to perform escaping for any $7E_{16}$ occurrence in the packet. The packet transmitted across this interface is the 802.17 frame as specified in Clause 8.

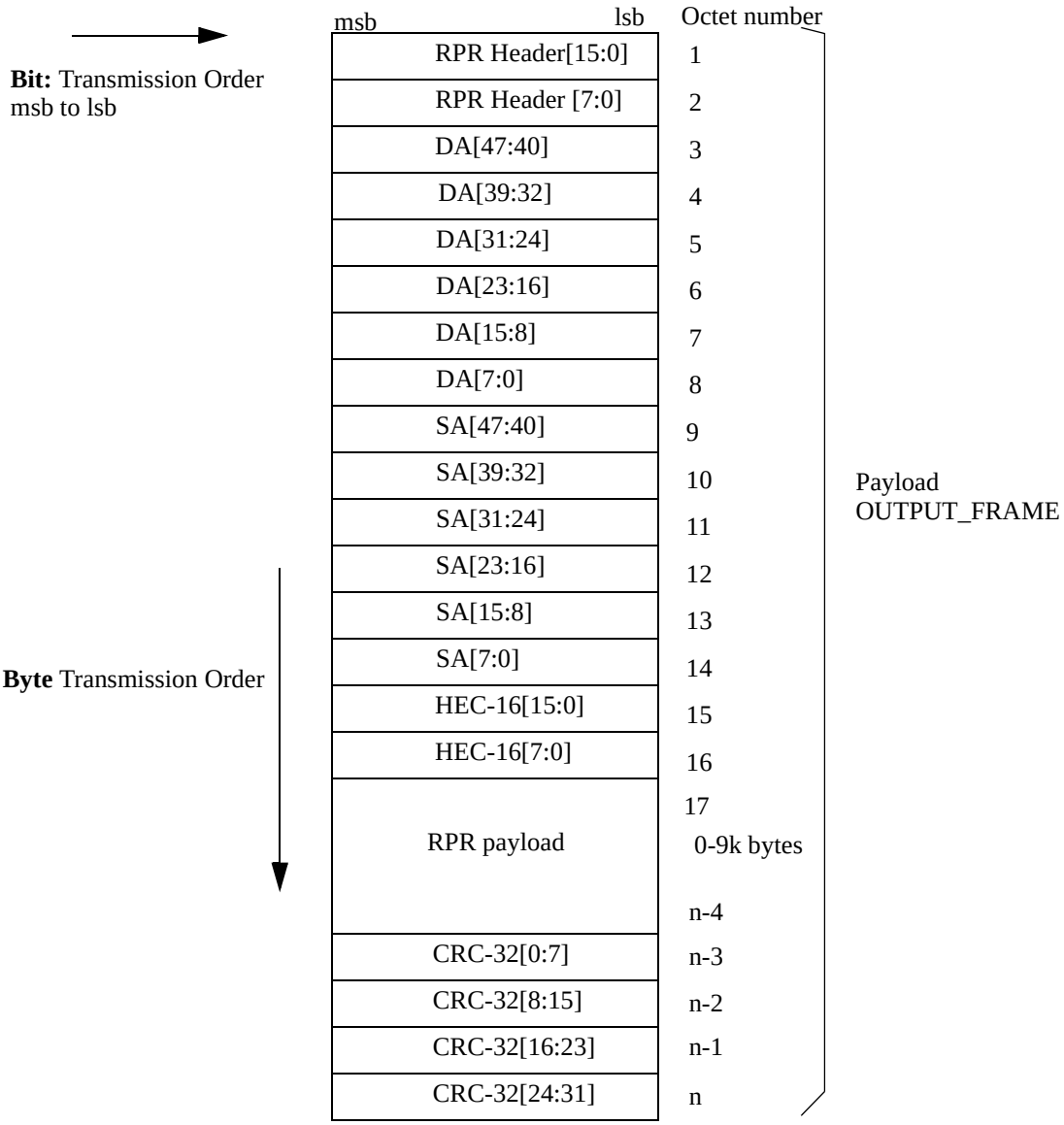


Figure C.3—SRS Frame Structure.

Editors' Notes: *To be removed prior to final publication.*

Update frame format if changes in clause 8.

C.2.2 GRS physical frame format

The GFP delineation method provides a deterministic inflation factor that is not packet content sensitive. It uses a length value protected by a cHEC to provide robust delineation under degrade channel condition. The HEC provides single error correction. In the case of GRS interface, the length and the HEC fields are part of the packet transmitted across the physical interface. The frame format is:

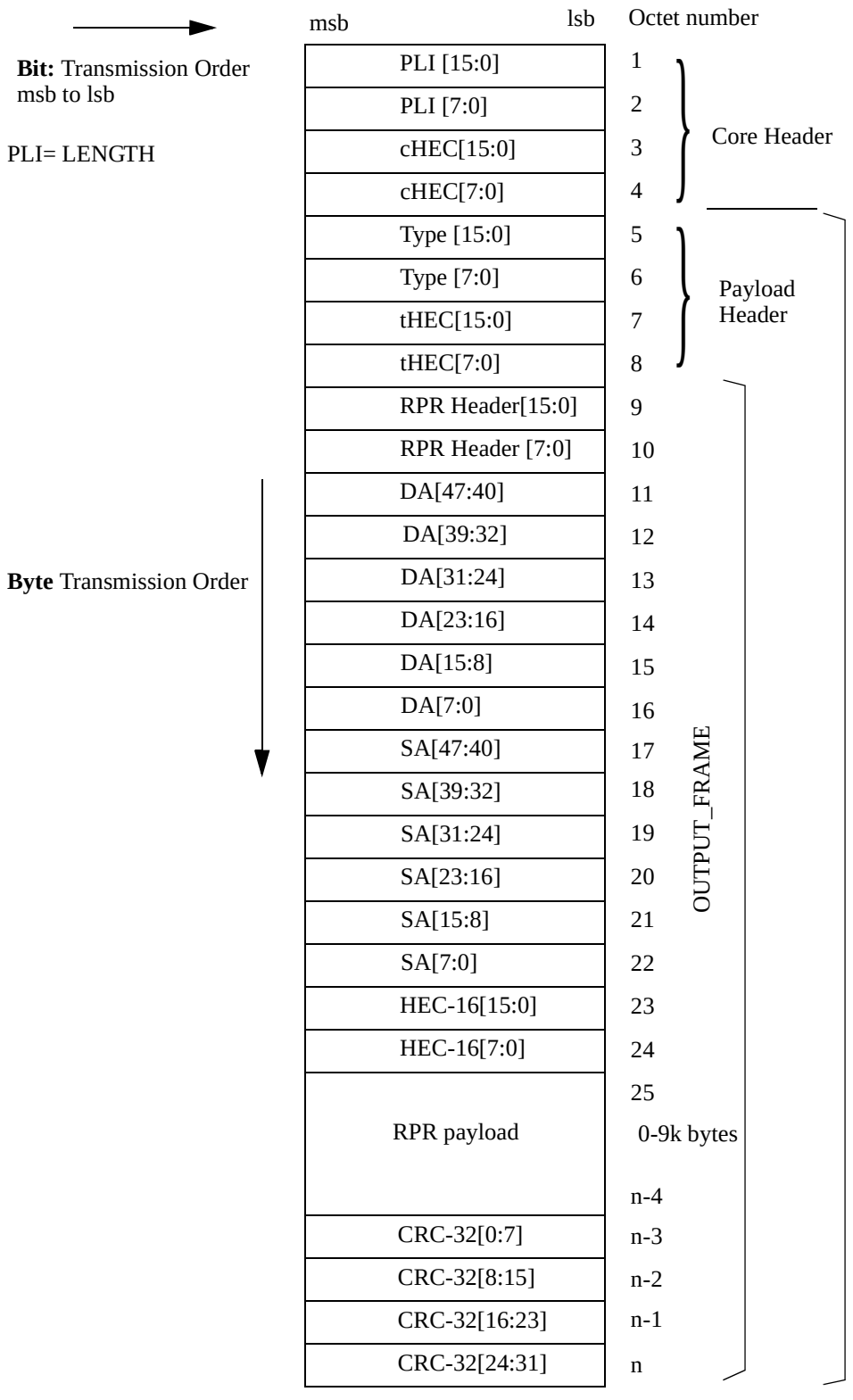


Figure C.4—GRS Frame structure, shown without GFP payload FCS

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 zero's.

The type field is constructed in the RS and prepends the OUTPUT_FRAME.

Operating in the GRS mode, if the length primitive is not received across the PSAP interface, a null value across the logical interface, the GRS shall generate the required GFP core header field. Alternatively, the SRS is used, then the GFP adaption sublayer generates the required GFP headers.

The calculation of the HEC uses the polynomial shown below:

$$CRC - 16 = x^{16} + x^{12} + x^{5} + 1$$

Figure C.5—HEC polynomial

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 2 fields: payload length indicator (PLI) and cHEC.

The core header supports frame delineation. The core header consists of two fields 1) payload length indicator and a cHEC. The PLI is a 16 bits binary representation of the decimal number in octets of the payload area. The cHEC is calculated over the PLI only.

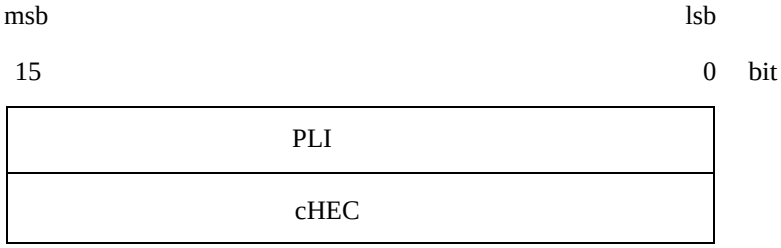


Figure C.6—GFP core header

C.2.2.2 GFP payload header

The basic GFP payload header consists of two fields: type and tHEC. For RPR application 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 sub-fields: 1) a 3 bit payload type identifier (PTI), 2) a 1-bit payload FCS indicator (PFI), 3) a 4-bit extension header identifier and 4) an 8-bit user payload identifier (UPI).

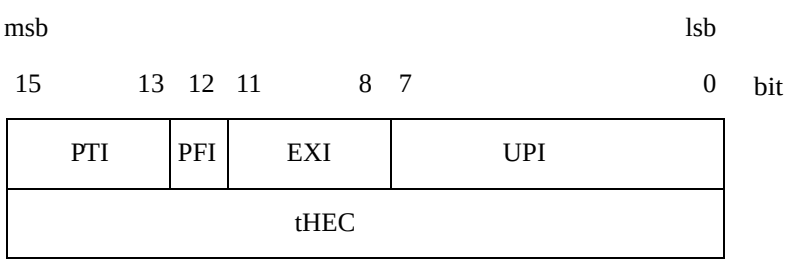


Figure C.7—GFP payload header

The definitions of the values are defined by the ITU-T G.7041 document. The recommend values for the above fields are:

Table C.2—GFP payload header field values

Field	Recommended value	Comments
PTI	000_2	Client data
PFI	0	absent of GFP FCS
EXI	0000_2	null extension header
UPI	See Note	RPR user payload type

Note: The assignment of this number is by ITU-T. Refer to ITU-TG.7041.

Editors' Notes: *To be removed prior to final publication.*

At the time this draft is written, there is no value assigned for RPR user payload.

C.3 SRS and GRS using the 8-bit SPI-3 interface

The SONET/SDH Reconciliation Sublayer and GFP Reconciliation Sublayer (GRS) may be implemented with an 8-bit OIF SPI-3 interface.

C.3.1 General requirements

The clauses in this section provide a brief description of the signals for the SPI3-01.0 interface and their interaction with the MAC physical layer service interface primitive.

Any discrepancy in the signal description the OIF-SPI3-01.0 takes precedence.

C.3.1.1 Summary of major concepts

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 delimiter are synchronous to clock references;
d) The SPI-3 interface supports logical channels with individual word-level or packet-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 Mbps to 622 Mbps, with an effective operation granularity of 155 Mbps. This is not an interface limit but the result of physical coding which maps to increments of STS1 or STS-3 SONET rates and increments of STM-1 for SDH rates.

SONET/SDH PHYs that provide an SPI-3 with 8-bit data path 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 operation mode of the PHY is expected operate in streaming, flow through or cut through mode.

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 interfaces can operate in byte level or packet level mode. In byte-level transfer, the FIFO status information is presented on a cycle by cycle basis. A PHY device indicates the transmit packet available status via the selected or directed method signals, STPA and DTPA[3:0] respectively for 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 one is to be supported to ensure the PHY FIFO does not overrun.

In packet-level transfer, the FIFO status information applies to segments of packet data. The RS polls the status of the PHY for transmit ready status. The RS polls one of multiple available PHYs by asserting the TADR address of the PHY which response 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 8 data signals (TDAT<7:0> and RDAT<7:0>), clocks (TFCLK and RFCLK), start of packet delimiter signals (TSOP and RSOP), and end of packet delimiter signals (TEOP and REOP). These signals are described in C.3.1.3.1 through C.3.1.3.20, and are fully defined in the OIF SPI-3 implementation agreement document.

Figure C.8 shows a schematic view of the SRS and GRS inputs and outputs using the 8-bit SPI-3 interface

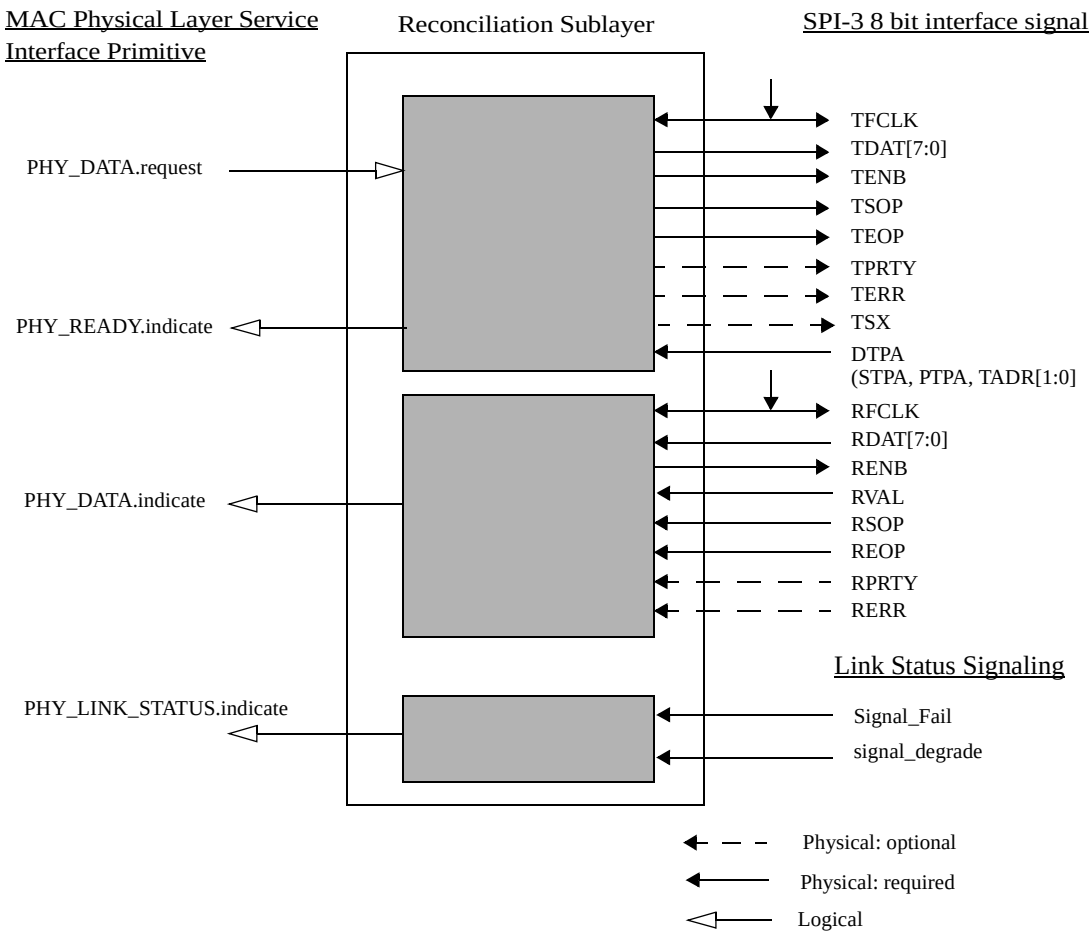


Figure C.8—SRS and GRS inputs and outputs using 8-bit SPI-3

The 8-bit SPI-3 interface signals are described in the following subclauses. All transmit signals are expected to be updated and sampled using the rising edge of the transmit clock, TFCLK.

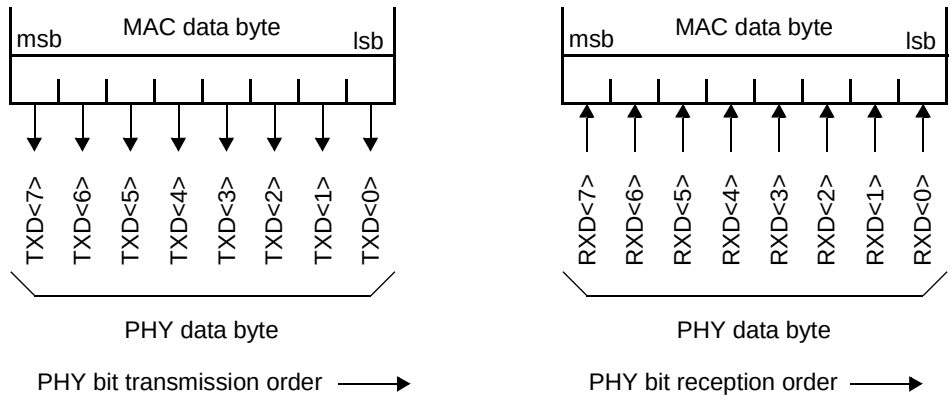


Figure C.9—8-bit SPI-3 transmit and receive signal mapping

C.3.1.3.1 TFCLK (transmit clock)

TFCLK is a continuous clock used to synchronize data transfer transactions between the RS and the PHY layer device. TFCLK may cycle at a rate up to 104 MHz. TFCLK is externally sourced to the RS and to the PHY.

C.3.1.3.2 TERR (transmit error Indicator)

The use of TERR is optional. There is no direct mapping from the PSAP primitives.

This is internal to the RS. When a packet is being transmitted and an internal error occurs then TERR is used to indicate that there is an error in the current packet. TERR should only be asserted when TEOP is asserted; it is considered valid only when TENB is asserted.

C.3.1.3.3 TENB (transmit write enable)

This is a physical interface protocol signal for indicate valid symbol transfer at the interface. The generation of this signal is as result the reception of a PHY_DATA.request and while valid data symbols are being transmitted across the physical interface.

When TENB is high the TDAT, TMOD, TSOP, TEOP and TERR signals are invalid and should be ignored by the PHY. The TSX signal is valid and is processed by the PHY when TENB is high. When TENB is low the TDAT, TMOD, TSOP, TEOP and TERR signals are valid and are processed by the PHY. The TSX signal is ignored by the PHY when TENB is low.

C.3.1.3.4 TDAT[7:0] (transmit packet data bus)

The generation of this signal is as result the reception of a PHY_DATA.request. The OUTPUT_FRAME and LENGTH primitives are formatted into the appropriate frames and the content of the frame are then transmitted across the physical interface.

The OUTPUT_FRAME is mapped into the payload for both SRS and GRS. GRS shall map valid LENGTH primitive into the PLI field of the GFP core header.

TDAT is an 8-bit bus which carries the packet bytes that are written to the selected transmit FIFO and the in-band port address to select the desired PHY transmit FIFO. The TDAT bus is considered valid only when TENB is asserted. Data is transmitted msb first. Mapping of TDAT signals is shown in Figure C.9.

C.3.1.3.5 TPRTY (transmit bus parity)

The use of TPRTY is optional and it is interface specific. There is no direct mapping to the PSAP primitives.

TPRTY indicates the parity calculated over the TDAT bus. TPRTY is considered valid only when TENB or TSX is asserted. The parity calculation is such that odd parity is indicated on this signal.

C.3.1.3.6 TSX (transmit start of transfer)

The use of TSX is optional and it is interface specific. There is no direct mapping from the PSAP primitives.

The use of TSX applies when multi-port PHY application.

TSX indicates when the in-band port address is present on the TDAT bus. When TSX is high and TENB is high, the value of TDAT is the address of the transmit FIFO to be selected. Subsequent data transfers on the

TDAT bus will fill the FIFO specified by this in-band address. For single channel PHY devices the TSX signal is optional. TSX is considered valid only when TENB is not asserted.

C.3.1.3.7 TSOP (transmit start of packet)

This is a physical interface protocol signal for flow control at the interface level. The generation of this signal is as result the reception of a PHY_DATA.request and while valid data symbols are being transmitted across the physical interface.

TSOP is used to delineate the packet boundaries on the TDAT bus. When TSOP is high the start of the packet is present on the TDAT bus. TSOP is required to be present at the beginning of every packet and is considered valid only when TENB is asserted.

C.3.1.3.8 TEOP (transmit end of packet)

This is a physical interface protocol signal for flow control at the interface level. The generation of this signal is as result the reception of a PHY_DATA.request and while the last valid data symbols are being transmitted across the physical interface.

TEOP is used to delineate the packet boundaries on the TDAT bus. When TEOP is high, the end of the packet is present on the TDAT bus. TEOP is required to be present at the end of every packet and is considered valid only when TENB is asserted.

C.3.1.3.9 TADR (transmit PHY address) and PTPA (polled-PHY transmit packet available)

The use of TADR is optional. It is for multi-port PHY application. TADR is used in the packet transfer mode. There is no direct mapping from the PSAP primitives.

TADR bus is used with the PTPA signal to poll the transmit FIFO's packet available status. When TADR is sampled on the rising edge of TFCLK by the PHY the polled packet available indication PTPA is updated with the status of the channel specified by the TADR address on the following rising edge of TFCLK.

The result of the polled status PTPA is used to generate PHY_READY.indicate.

C.3.1.3.10 DTPA (direct transmit packet available)

DTPA is used in the byte transfer mode.

The result of DPTA is used to generate PHY_READY.indicate.

DTPA bus provides direct status indication for the corresponding ports in the PHY device. DTPA transitions high when a predefined minimum number of bytes is available in its transmit FIFO. Once high, the DTPA signal indicates that its corresponding transmit FIFO is not full. When DTPA transitions low it indicates that its transmit FIFO is full or near full. DTPA is updated on the rising edge of TFCLK.

C.3.1.3.11 STPA (selected transmit packet available)

The use of STPA is optional. It is for multi-port PHY application. STPA is used in the byte transfer mode.

The result of DPTA is used to generate PHY_READY.indicate.

STPA provides status indication of the selected port of a PHY device in order to avoid PHY FIFO overflow while polling is performed. The port which STPA reports is updated on the following rising edge of TFCLK after the PHY address on the TDAT is sampled by the PHY.

C.3.1.3.12 RFCLK (receive FIFO write clock)

RFCLK is a continuous clock used to synchronize data transfer transactions between the RS and the PHY. RFCLK may cycle at a rate up to 104 MHz.

C.3.1.3.13 RVAL (receive data valid)

RVAL indicates the validity of the receive data. RVAL is low between transfers and when RSX is asserted. It is also low when the PHY pauses a transfer due to an empty PHY receive FIFO. When a transfer is paused by holding RENB high RVAL will hold its value unchanged although no new data will be present on RDAT until the transfer resumes. When RVAL is high the RDAT, RMOD, RSOP, REOP and RERR signals are valid. When RVAL is low, the RDAT, RMOD, RSOP, REOP and RERR signals are invalid and shall be disregarded. The RSX signal is valid when RVAL is low.

C.3.1.3.14 RENB (receive read enable)

The RENB signal is used to control the flow of data from the PHYs. During data transfer, RVAL shall be monitored as it will indicate if the RDAT, RPRTY, RMOD, RSOP, REOP, RERR and RSX are valid. The system may deassert RENB at anytime if it is unable to accept data from the PHY device. When RENB is sampled low by the PHY device a read is performed from the receive FIFO and the RDAT, RPRTY, RMOD, RSOP, REOP, RERR, RSX and RVAL signals are updated on the following rising edge of RFCLK. When RENB is sampled high by the PHY device a read is not performed and the RDAT, RPRTY, RMOD, RSOP, REOP, RERR, RSX and RVAL remain unchanged on the following rising edge of RFCLK.

The MAC is expected to receive data from the PHY without applying back pressure.

C.3.1.3.15 RDAT[7:0] (receive packet data bus)

RDAT is an 8-bit bus which carries the packet bytes that are read from the PHY and the in-band port address of the selected receive port. RDAT is considered valid only when RVAL is asserted. Mapping of RDAT signals is shown in Figure C.8.

Valid data received on the RDAT bus from RSOP to REOP are mapped into PHY_DATA.indicate.

C.3.1.3.16 RPRTY (receive parity)

The use of RPTRY is optional as it does not affect the flow of data from the PHY to the RS. This is used for interface integrity checking. Parity error detected does not cause the received packet to be discarded.

RPRTY signal indicates the odd parity calculated over the RDAT bus.

C.3.1.3.17 RSOP (receive start of packet)

RSOP is used to delineate the packet boundaries on the RDAT bus. When RSOP is high, the start of the packet is present on the RDAT bus. RSOP is required to be present at the beginning of every packet and is considered valid only when RVAL is asserted.

Valid data received on the RDAT bus from RSOP to REOP are mapped into PHY_DATA.indicate.

C.3.1.3.18 REOP (receive end of packet)

REOP is used to delineate the packet boundaries on the RDAT bus. When REOP is high, the end of the packet, the last byte is present on the RDAT bus. REOP is required to be present at the end of every packet and is considered valid only when RVAL is asserted.

Upon receiving an REOP, the data received from RSOP constitute a received frame. For SRS sublayer, the received frame formulates the INPUT_FRAME attribute of the PHY_DATA.indicate primitive. For a GRS sublayer, 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.indicate primitive.

C.3.1.3.19 RERR (receive error indicator)

The use of RERR is optional, as it does not interfere with the transfer of the packet.

RERR is used to indicate that the current packet is in error. RERR shall only be asserted when REOP is asserted. Conditions that can cause RERR to be set may be, but are not limit to, PHY FIFO overflow, abort sequence detection and PHY state machine error detected.

C.3.1.3.20 RSX (receive start of transfer)

The use of RSX is optional, when a multi-port PHY device and in-band addressing scheme is used.

RSX indicates when the in-band port address is present on the RDAT bus. When RSX is high, the value of RDAT[7:0] is the address of the receive FIFO to be selected by the PHY. Subsequent data transfers on the RDAT bus will be from the FIFO specified by this in-band address. For single channel PHY devices the RSX signal is optional. For multi-port PHY devices, RSX is asserted at the beginning of each transfer. When RSX is high RVAL shall be low.

C.3.1.4 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 7. Mappings for the following primitives are defined:

— PHY_DATA.request
— PHY_DATA.indicate
— PHY_READY.indicate.

C.3.1.4.1 Mapping of PHY_DATA.request

PHY_DATA.request (OUTPUT_FRAME, LENGTH

There are two attributes: OUTPUT_FRAME and LENGTH

The OUTPUT_FRAME is mapped into the payload. The LENGTH applies to GRS only, it is mapped into the PLI field of the GFP core header.

C.3.1.4.1.1 When generated

This primitive is invoked by the MAC_datapath entity when it is performed its tasks after it had received a received a request from its client.

C.3.1.4.1.2 Effect of receipt

The receipt of this primitive shall cause the RS to generate the physical frame format required operating in GRS or SRS mode as described in C.2.1 and C.2.2 . The RS will start transferring the physical frame to the PHY in according to SPI-3 protocol: generating TSOP, TDAT, TMOD, and TEOP signals.

C.3.1.4.2 Mapping of PHY_DATA.indicate

PHY_DATA.indicate (INPUT_FRAME, STATUS, LENGTH)

There are three attributes: INPUT_FRAME, STATUS, and LENGTH

The INPUT_FRAME is mapped from the received payload. The LENGTH applies to GRS only. It is mapped from the PLI field in the GFP core header.

The STATUS parameter takes the value of OK or ERROR and is used to identify frames that are receive with error indications from the PHY, for example, due to assertion of a error signal on the PHY electrical interface. A value of OK indicates no error indication. A value of ERROR indicates that the PHY signaled an error during frame transmission to the reconciliation sublayer.

C.3.1.4.2.1 When generated

This primitive is invoked by the RS after it had received a complete frame from the physical interface. The INPUT_FRAME is derived from the signals RDAT[31:0], RMOD[1:0], as signalled 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.3.1.4.2.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 6.

C.3.1.4.3 Mapping of PHY_READY.indicate

PHY_READY.indicate (READY_STATUS

The READY_STATUS attribute takes on the3 value of READY or NOT_READY.

C.3.1.4.3.1 When generated

Depending on single or multiple PHY mode, STPA, PTPA, or DPTA[], are indications from the PHY to the RS that the PHY is ready to receive data on the TDAT bus. Valid indication from the PHY is mapped to the PHY_READY.indicate primitive and send to RS client.

C.3.1.4.3.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 6.

C.3.2 SRS and GRS 8-bit SPI datastream

There are no electrical difference and signalling protocol between the GRS and the SRS sublayer. The only difference is the physical frame format as specified in C.2.1 and C.2.2 .

C.3.3 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 OIF SPI-3 implementation agreement.

C.3.4 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 OIF SPI-3 implementation agreement.

The electrical characteristics for signal_degrade and signal_fail are LVTTL compatible with the supply voltage of the SPI-3 bus.

C.4 SRS and GRS using the 32-bit SPI-3 interface

The SONET/SDH Reconciliation Sublayer and GFP Reconciliation Sublayer (GRS) may be implemented with a 32-bit OIF SPI-3 interface.

C.4.1 General requirements

This clause provides a brief description of the signals for the SPI3-01.0 interface and their interaction with the MAC physical layer service interface primitive.

Any discrepancy in the signal description the OIF-SPI3-01.0 takes precedence.

C.4.1.1 Summary of major concepts

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 delimiter are synchronous to clock references;
d) The SPI-3 interface supports logical channels with individual word-level or packet-level out of band flow control;
e) The SRS and GRS using the 32-bit SPI-3 interface support full-duplex operation only.

C.4.1.2 Rate of operation

The SRS and GRS using the 32-bit SPI-3 interface are capable of supporting data rates of 155 Mbps to 2.5 Gbps. The granularity is dependent on the mapping of the SONET/SDH synchronous payload. The interface clock is set fast enough so as the PHY transmit FIFO does not starve in and the PHY receive FIFO does not overrun

SONET/SDH PHYs that provide an SPI-3 with 32-bit data path 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 operation mode of the PHY is expected operate in streaming, flow through or cut through mode.

C.4.1.3 32-bit SPI-3 structure

SPI interface can operate in byte level or packet level mode. In byte-level transfer, the FIFO status information is presented on a cycle by cycle basis. The PHY device indicates the transmit packet available status via the STPA or DTPA[3:0] signals. DTPA signals are provided for simpler implementation to reduce the need for addressing. Support for one or the other is optional but one is to be supported to ensure the PHY FIFO does not overrun.

In packet-level transfer, the FIFO status information applies to segments of packet data. The RS polls the status of the PHY for transmit ready status. The RS polls one of multiple available PHYs by asserting the TADR address of the PHY which response 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), and end of packet delimiter signals (TEOP and REOP). Mapping of the TDAT and RDAT signals is shown in These signals are fully defined in the OIF SPI-3 implementation agreement. Figure C.10 shows a schematic view of the SRS and GRS inputs and outputs using the 32-bit SPI-3 interface.

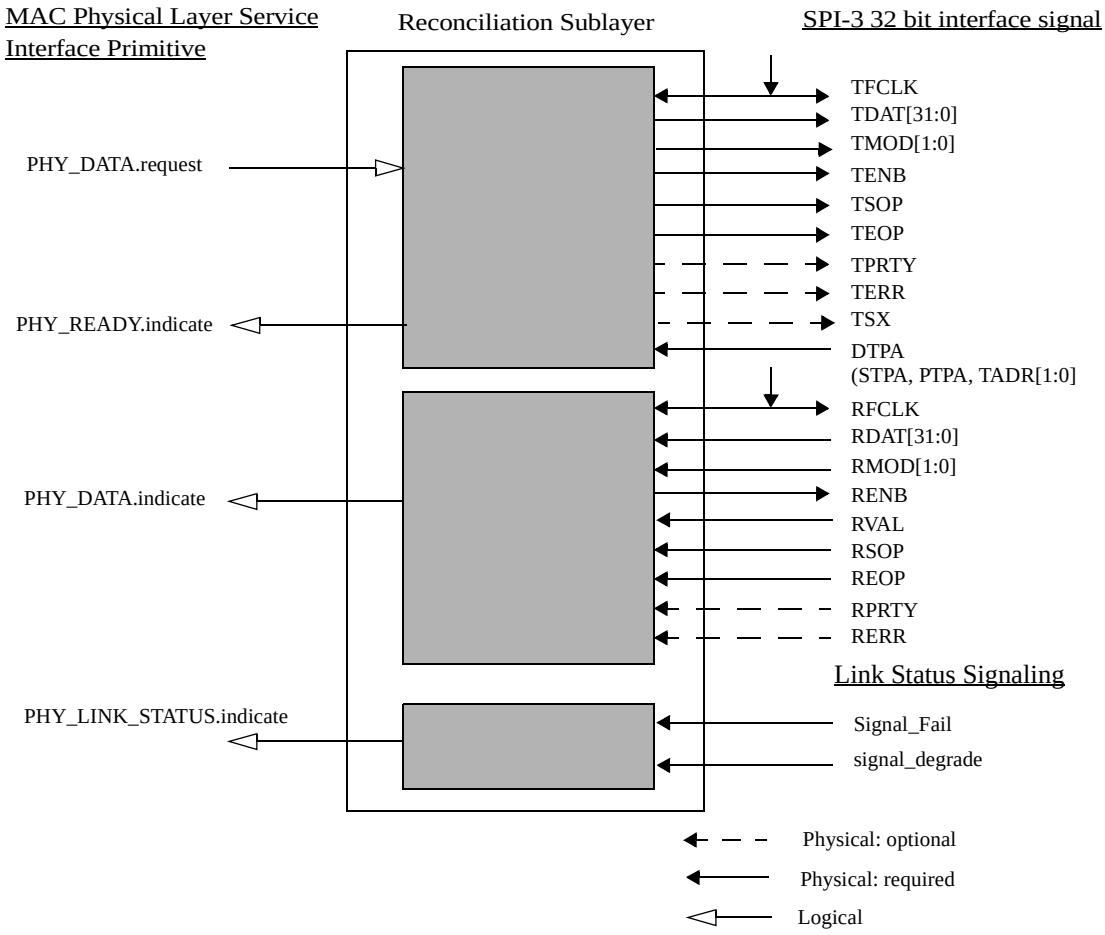


Figure C.10—SRS and GRS inputs and outputs using 32-bit SPI-3

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.

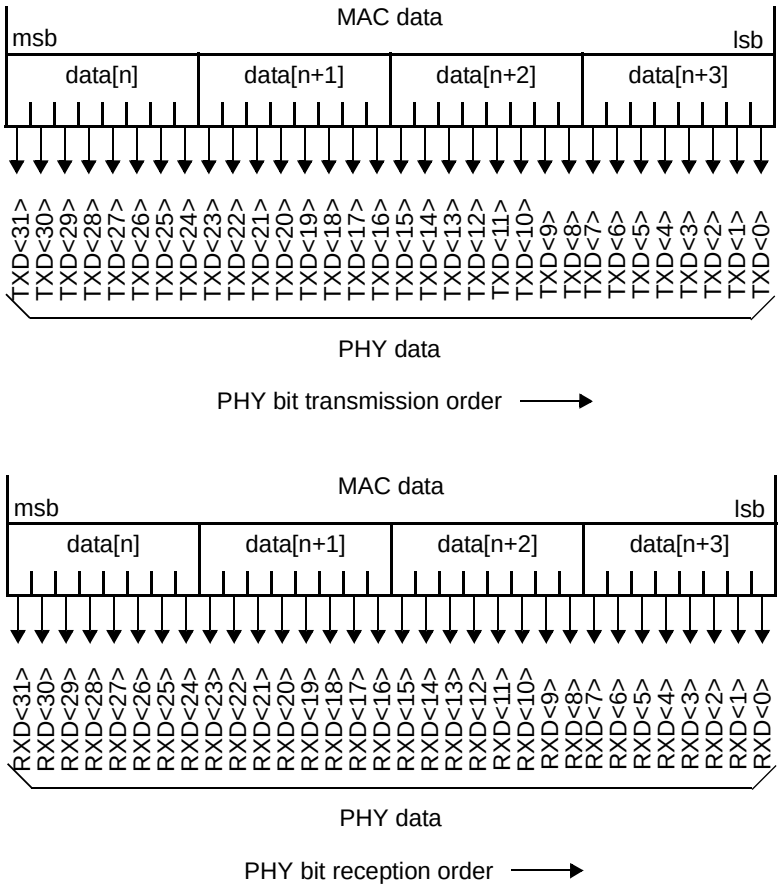


Figure C.11—32-bit SPI-3 transmit and receive signal mapping

C.4.1.3.1 TFCLK (transmit clock)

TFCLK is a continuous clock used to synchronize data transfer transactions between the RS and the PHY layer device. TFCLK may cycle at a rate up to 104 MHz. TFCLK is externally sourced to the RS and to the PHY.

C.4.1.3.2 TERR (transmit error Indicator)

The use of TERR is optional. There is no direct mapping from the PSAP primitives.

This is internal to the RS. When a packet is being transmitted and an internal error occurs then TERR is used to indicate that there is an error in the current packet. TERR should only be asserted when TEOP is asserted; it is considered valid only when TENB is asserted.

C.4.1.3.3 TENB (transmit write enable)

This is a physical interface protocol signal for flow control at the interface level. The generation of this signal is as result the reception of a PHY_DATA.request and while valid data symbols are being transmitted across the physical interface.

When TENB is high the TDAT, TMOD, TSOP, TEOP and TERR signals are invalid and should be ignored by the PHY. The TSX signal is valid and is processed by the PHY when TENB is high. When TENB is low the TDAT, TMOD, TSOP, TEOP and TERR signals are valid and are processed by the PHY. The TSX signal is ignored by the PHY when TENB is low.

C.4.1.3.4 TDAT[31:0] (transmit packet data bus)

TDAT is an 32-bit bus which carries the packet bytes that are written to the selected transmit FIFO and the in-band port address to select the desired transmit FIFO. The TDAT bus is considered valid only when TENB is asserted. Data is transmitted in big endian order on TDAT. Mapping of TDAT signals is shown in Figure C.11.

C.4.1.3.5 TMOD[1:0] (transmit word modulo)

TMOD is required only when TDAT is 32-bits. TMOD indicates the number of valid bytes of data in TDAT. The TMOD bus should always be zero except during the last word transfer of a packet on TDAT. When TEOP is asserted the number of valid packet data bytes on TDAT is decoded from TMOD as:

TMOD[1:0] = 00	TDAT[31:0] is valid
TMOD[1:0] = 01	TDAT[31:8] is valid
TMOD[1:0] = 10	TDAT[31:16] is valid
TMOD[1:0] = 11	TDAT[31:24] is valid

TMOD is considered valid only when TENB is asserted.

C.4.1.3.6 TPRTY (transmit bus parity)

TPRTY indicates the parity calculated over the TDAT bus. TPRTY is considered valid only when TENB or TSX is asserted. The parity calculation is such that odd parity is indicated on this signal. There is no direct mapping from the PSAP primitives.

C.4.1.3.7 TSX (transmit start of transfer)

The use of TSX is optional and it is interface specific. There is no direct mapping from the PSAP primitives.

TSX indicates when the in-band port address is present on the TDAT bus. When TSX is high and TENB is high, the value of TDAT is the address of the transmit FIFO to be selected. Subsequent data transfers on the TDAT bus will fill the FIFO specified by this in-band address. For single channel PHY devices the TSX signal is optional. TSX is considered valid only when TENB is not asserted.

C.4.1.3.8 TSOP (transmit start of packet)

TSOP is used to delineate the packet boundaries on the TDAT bus. When TSOP is high the start of the packet is present on the TDAT bus. TSOP is required to be present at the beginning of every packet and is considered valid only when TENB is asserted. The generation of this signal is as result of reception of a PHY_DATA.request.

C.4.1.3.9 TEOP (transmit end of packet)

TEOP is used to delineate the packet boundaries on the TDAT bus. When TEOP is high, the end of the packet is present on the TDAT bus. TEOP is required to be present at the end of every packets and is considered valid only when TENB is asserted. The generation of this signal is as result of reception of a PHY_DATA.request.

C.4.1.3.10 TADR (transmit PHY address) and PTPA (polled-PHY transmit packet available)

The use of TADR is optional. It is for multi-port PHY application. A RS communicates with one port of a PHY device. TADR is used in the packet transfer mode. There is no direct mapping from the PSAP primitives.

TADR bus is used with the PTPA signal to poll the transmit FIFO's packet available status. When TADR is sampled on the rising edge of TFCLK by the PHY the polled packet available indication PTPA is updated with the status of the channel specified by the TADR address on the following rising edge of TFCLK.

The result of the polled status PTPA is used to generate PHY_READY.indicate.

C.4.1.3.11 DTPA (direct transmit packet available)

DTPA is used in the byte transfer mode.

The result of DPTA is used to generate PHY_READY.indicate.

DTPA bus provides direct status indication for the corresponding ports in the PHY device. DTPA transitions high when a predefined minimum number of bytes is available in its transmit FIFO. Once high, the DTPA signal indicates that its corresponding transmit FIFO is not full. When DTPA transitions low it indicates that its transmit FIFO is full or near full. DTPA is updated on the rising edge of TFCLK.

C.4.1.3.12 STPA (selected transmit packet available)

The use of STPA is optional. It is for multi-port PHY application. STPA is used in the byte transfer mode.

The result of DPTA is used to generate PHY_READY.indicate.

STPA provides status indication of the selected port of a PHY device in order to avoid PHY FIFO overflow while polling is performed. The port which STPA reports is updated on the following rising edge of TFCLK after the PHY address on the TDAT is sampled by the PHY.

C.4.1.3.13 RFCLK (receive FIFO write clock)

RFCLK is a continuous clock used to synchronize data transfer transactions between the MAC and the PHY. RFCLK may cycle at a rate up to 104 MHz.

C.4.1.3.14 RVAL (receive data valid)

RVAL indicates the validity of the receive data. RVAL is low between transfers and when RSX is asserted. It is also low when the PHY pauses a transfer due to an empty receive FIFO. When a transfer is paused by holding RENB high RVAL will hold its value unchanged although no new data will be present on RDAT until the transfer resumes. When RVAL is high the RDAT, RMOD, RSOP, REOP and RERR signals are valid. When RVAL is low, the RDAT, RMOD, RSOP, REOP and RERR signals are invalid and shall be disregarded. The RSX signal is valid when RVAL is low.

C.4.1.3.15 RENB (receive read enable)

The RENB signal is used to control the flow of data from the receive FIFOs. During data transfer, RVAL shall be monitored as it will indicate if the RDAT, RPRTY, RMOD, RSOP, REOP, RERR and RSX are valid. The system may deassert RENB at anytime if it is unable to accept data from the PHY device. When RENB is sampled low by the PHY device a read is performed from the receive FIFO and the RDAT, RPRTY, RMOD, RSOP, REOP, RERR, RSX and RVAL signals are updated on the following rising edge of RFCLK.

When RENB is sampled high by the PHY device a read is not performed and the RDAT, RPRTY, RMOD, RSOP, REOP, RERR, RSX and RVAL remain unchanged on the following rising edge of RFCLK.

C.4.1.3.16 RDAT[31:0] (receive packet data bus)

RDAT is an 8-bit or 32-bit bus which carries the packet bytes that are read from the receive FIFO and the in-band port address of the selected receive FIFO. RDAT is considered valid only when RVAL is asserted. Mapping of RDAT signals is shown in Figure C.11

Valid data received on the RDAT bus from RSOP to REOP are mapped into PHY_DATA.indicate.

C.4.1.3.17 RPRTY (receive parity)

RPRTY signal indicates the odd parity calculated over the RDAT bus. There is no direct mapping to the PSAP primitives.

TPRTY indicates the parity calculated over the TDAT bus. TPRTY is considered valid only when TENB or TSX is asserted. The parity calculation is such that odd parity is indicated on this signal.

C.4.1.3.18 RMOD[1:0] (receive word modulo)

RMOD is required only when RDAT is a 32-bit bus. RMOD indicates the number of valid bytes of data in RDAT. The RMOD bus should always be zero except during the last word transfer of a packet on RDAT. When REOP is asserted the number of valid packet data bytes on RDAT is decoded from RMOD as:

RMOD[1:0] = 00 RDAT[31:0] is valid
RMOD[1:0] = 01 RDAT[31:8] is valid
RMOD[1:0] = 10 RDAT[31:16]is valid
RMOD[1:0] = 11 RDAT[31:24] is valid

RMOD is considered valid only when RVAL is asserted

C.4.1.3.19 RSOP (receive start of packet)

RSOP is used to delineate the packet boundaries on the RDAT bus. When RSOP is high, the start of the packet is present on the RDAT bus. RSOP is required to be present at the beginning of every packet and is considered valid only when RVAL is asserted.

Valid data received on the RDAT bus from RSOP to REOP are mapped into PHY_DATA.indicate.

C.4.1.3.20 REOP (receive end of packet)

REOP is used to delineate the packet boundaries on the RDAT bus. When REOP is high, the end of the packet is present on the RDAT bus. REOP is required to be present at the end of every packet and is considered valid only when RVAL is asserted.

Upon receiving an REOP, the data received from RSOP constitute a received frame. For SRS sublayer, the received frame formulates the INPUT_FRAME attribute of the PHY_DATA.indicate primitive. For a GRS sublayer, 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.indicate primitive.

C.4.1.3.21 RERR (receive error indicator)

The use of RERR is optional, as it does not interfere with the transfer of the packet.

RERR is used to indicate that the current packet is in error. RERR shall only be asserted when REOP is asserted. Conditions that can cause RERR to be set may be, but are not limit to, FIFO overflow, abort sequence detection and L1 FCS error.

C.4.1.3.22 RSX (receive start of transfer)

RSX indicates when the in-band port address is present on the RDAT bus. When RSX is high, the value of RDAT[31:0] is the address of the receive FIFO to be selected by the PHY. Subsequent data transfers on the RDAT bus will be from the FIFO specified by this in-band address. For single channel PHY devices the RSX signal is optional. For multi-port PHY devices, RSX is asserted at the beginning of each transfer. When RSX is high RVAL shall be low.

C.4.1.4 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 7. Mappings for the following primitives are defined:

— PHY_DATA.request
— PHY_DATA.indicate
— PHY_READY.indicate.

C.4.1.4.1 Mapping of PHY_DATA.request

PHY_DATA.request (OUTPUT_FRAME, LENGTH)

The OUTPUT_FRAME represents a complete RPR frame, and is conveyed to the PHY by the signals TDAT<31:0> and TMOD<1:0>.

On each TFCLK rising edge, 32 bits of data are transferred from the reconciliation sublayer to the PHY. Data is mapped to a TDAT signal in sequence (TDAT<0:7>, …, TDAT<24:31>, TDAT<0:7>), using the big endian coding as described in C.4.1.4, and with the parity information in the TPRTY signal corresponding to the value of the previous 32 bit data word. The TMOD<1:0> is always fixed to the “00” value except when the end-of-packet 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 scope of this standard. If used, this feature should be compliant with the requirements in the OIF specification.

Following transmission of the complete frame, the reconciliation sublayer generates an end-of-packet on the SPI-3 interface. The RS 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 OIF specification

The SRS and GRS do not generate preamble or interpacket gap since they are not needed for SONET/SDH PHYs.

C.4.1.4.1.1 When generated

This primitive is invoked by the MAC_datapath entity when it is performed its tasks after it had received a received a request from its client.

C.4.1.4.1.2 Effect of receipt

The receipt of this primitive shall cause the RS to generate the physical frame format required operating in GRS or SRS mode as described in C.2.1 and C.2.2 . The RS will start transferring the physical frame to the PHY in according to SPI-3 protocol: generating TSOP, TDAT, TMOD, and TEOP signals.

C.4.1.4.2 Mapping of PHY_DATA.indicate

PHY_DATA.indicate (INPUT_FRAME, STAUS, LENGTH)

The INPUT_FRAME is derived from the signals RFCLK, RVAL, RENB, RDAT<31:0>, RPRTY, RMOD<1:0>, RSOP, REOP, RERR, RSX.

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 scope of this standard. If used, this feature should be compliant with the requirements of the OIF 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 RS receives four bytes of a frame until the end of frame (when the REOP is high), where one, tow, 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 OIF SPI-3 specification.

During frame reception each RDAT signal shall be mapped in sequence into a received frame (RDAT<0:7>, …, RDAT<24:31>, RDAT<0:7>).

The STATUS parameter takes the value of OK or ERROR and is used to identify frames that are receive with error indications from the PHY, for example, due to assertion of a error signal on the PHY electrical interface. A value of OK indicates no error indication. A value of ERROR indicates that the PHY signaled an error during frame transmission to the reconciliation sublayer.

C.4.1.4.2.1 When generated

This primitive is invoked by the RS after it had received a complete frame from the physical interface. The INPUT_FRAME is derived from the signals RDAT[31:0], RMOD[1:0], as signalled 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.4.1.4.2.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 6.

C.4.1.4.3 Mapping of PHY_READY.indicate

PHY_READY.indicate (READY_STATUS)

The READY_STATUS attribute takes on the3 value of READY or NOT_READY.

C.4.1.4.3.1 When generated

Depending on single or multiple PHY mode, STPA, PTPA, or DPTA[], are indications from the PHY to the RS that the PHY is ready to receive data on the TDAT bus. Valid indication from the PHY is mapped to the PHY_READY.indicate primitive and send to RS client.

C.4.1.4.3.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 6.

C.4.2 SRS and GRS 32-bit SPI datastream

There are no electrical difference and signalling protocol between the GRS and the SRS sublayer. The only difference is the physical frame format as specified in C.2.1 and C.2.2 .

C.4.3 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 OIF SPI-3 implementation agreement.

C.4.4 Electrical specifications

The SRS and GRS using 32-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 OIF SPI-3 implementation agreement.

C.5 SRS and GRS using the SPI-4 Phase 1 interface

The SONET/SDH Reconciliation Sublayer and GFP Reconciliation Sublayer (GRS) may be implemented with an OIF SPI-4 Phase 1 interface.

C.5.1 General requirements

This clause provides a brief description of the signal for the SPI4-01.0 interface and its interaction with the MAC physical layer service interface primitive. Any discrepancy in the signal description is not intentional. The OIF-SPI4-01.0 takes precedence over the subclause below.

C.5.1.1 Summary of major concepts

a) The SRS and GRS map the signals provided at the SPI-4 Phase 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 using the SPI-4 Phase 1 interface support full-duplex operation only.

C.5.1.2 Rate of operation

The SRS and GRS using the SPI-4 Phase 1 interface are capable of supporting data rates of 622 Mbps to 10 Gbps.

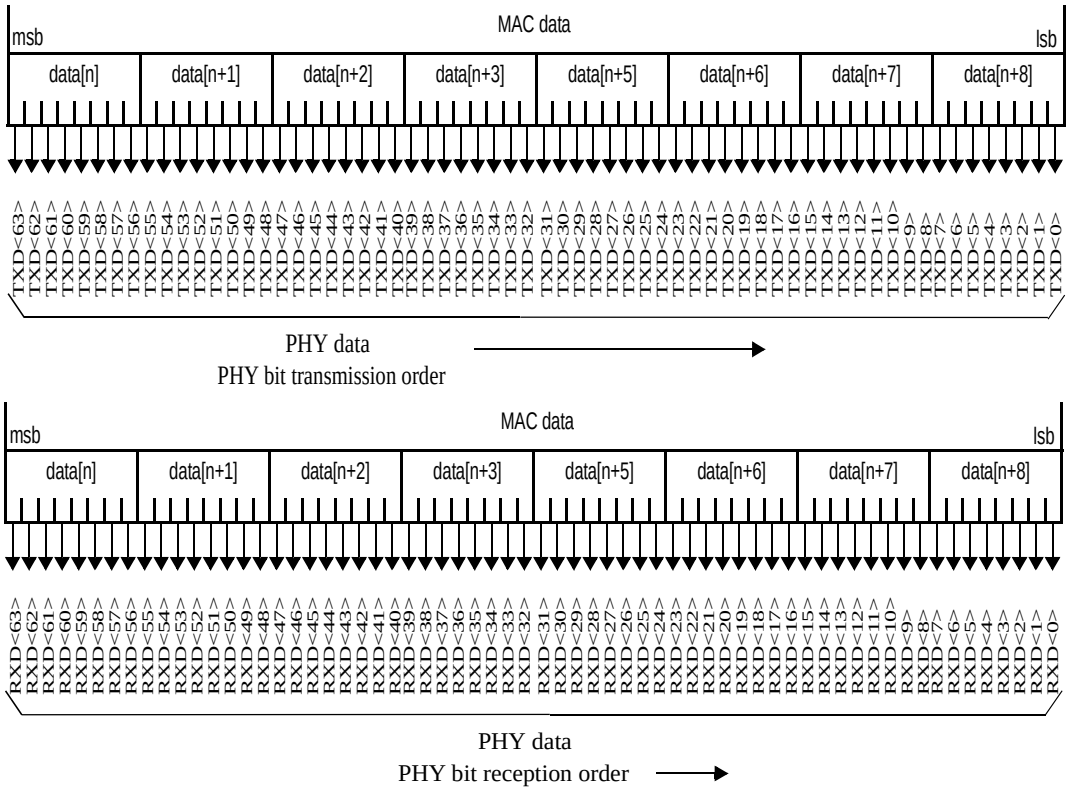


Figure C.12—64-bit SPI-4 phase 1 transmit and receive signal mapping

C.5.1.3 SPI-4 Phase 1 structure

SPI-4 phase 1 supports a single 64 bit interface and a 4x16 bit interface. The 4x16 mode operates as 4 independent 16 bit bus interface each with its full set control signals and clock. A RPR MAC is a single link layer device and it requires the PHY interface to operate a single PHY device.

The 4x16 mode is used where applicable. This clause describes the operation for single 10Gbps MAC operation.

Figure C.13 shows a schematic view of the SRS and GRS inputs and outputs using the 64-bit SPI-4 phase 1interface.

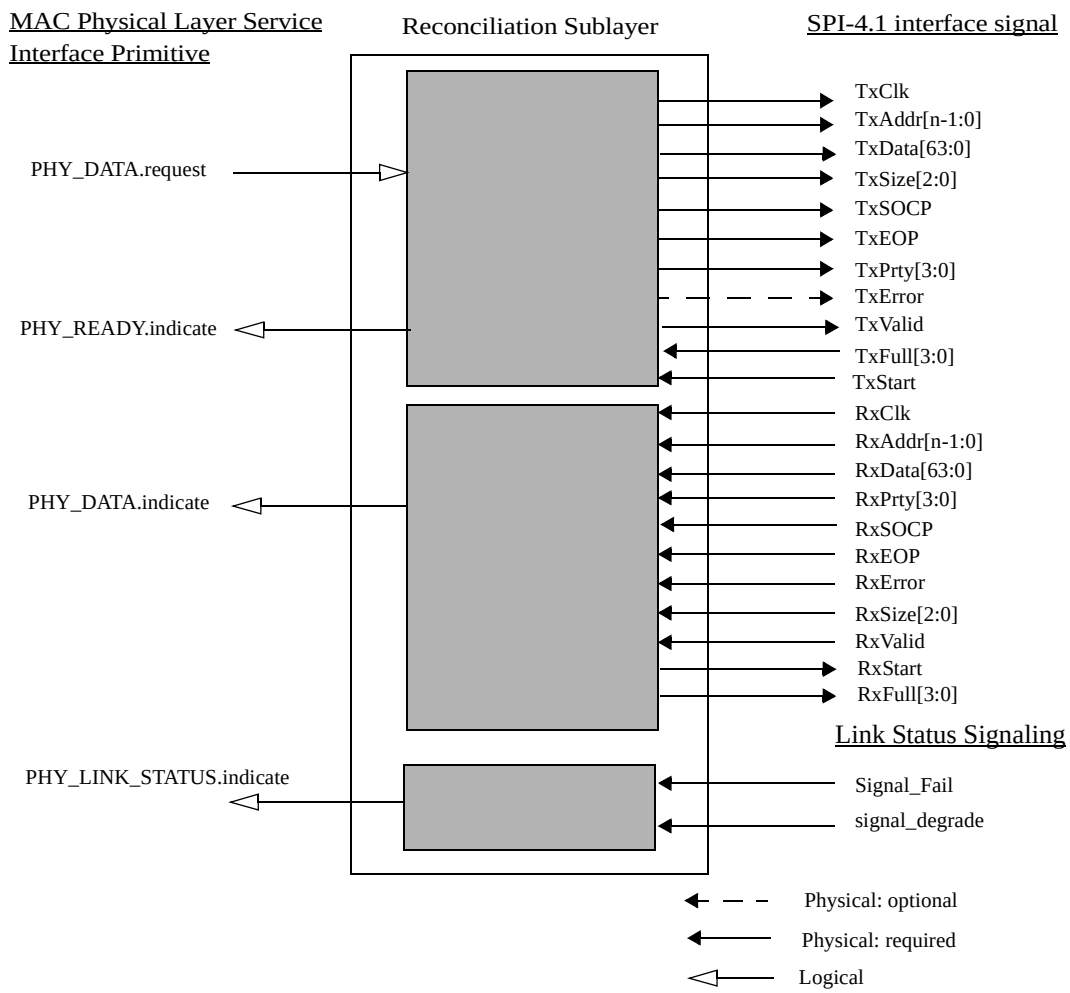


Figure C.13—SPI-4.1 single 16 bit interface Interface Signals

C.5.1.3.1 TxClk (transmit clock)

TxClk is a continuous clock used to synchronize data transfer transactions between the RS and the PHY layer device. TxClk may cycle at a rate up to 200 MHz. SPI-4.1 uses source synchronous clocking.

C.5.1.3.2 TxError (transmit error Indicator)

The use of TxError is optional. There is no direct mapping from the PSAP primitives.

When a packet is being transmitted and an internal error occurs then TxError is used to indicate that there is an error in the current packet. TxError should only be asserted when TxEOP is asserted; it is considered valid only when TxValid is asserted. The TxError does not interfere with the transfer of data in the PHY.

C.5.1.3.3 TxValid (transmit data valid)

This is a physical interface protocol signal for flow control at the interface level. The generation of this signal is as result the reception of a PHY_DATA.request and while valid data symbols are being transmitted across the physical interface.

When TxValid is deasserted the TxData, TxSize, TxSOCP, TxEOP and TxError signals are invalid and should be ignored by the PHY.

C.5.1.3.4 TxData[63:0] (transmit packet data bus)

TxData is an 64-bit bus which carries the packet bytes that are written to the selected transmit FIFO port whose address is select by the TxAddr signals. The TxData bus is considered valid only when TxValid is asserted. Data is transmitted in big endian order on TxData. Mapping of TxData signals is shown in Figure C.12.

The generation of this signal is as result of reception of a PHY_DATA.request. The OUTPUT_FRAME and LENGTH primitives are formatted into the appropriate physical frame format and each

C.5.1.3.5 TxSize[2:0] (transmit word modulo)

TxSize is required for the 64-bits TxData bus. TxSize indicates the number of valid bytes of data in TxData. The TxSize bus should always be zero except during the last word transfer of a packet on TxData. When TxEOP is asserted the number of valid packet data bytes on TxData is decoded from TxSize as:

TxSize[2:0] = 001 TxData[63:56] is valid
TxSize[2:0] = 010 TxData[63:48] is valid
TxSize[2:0] = 011 TxData[63:40] is valid
TxSize[2:0] = 100 TxData[63:32] is valid
TxSize[2:0] = 101 TxData[63:24] is valid
TxSize[2:0] = 110 TxData[6316] is valid
TxSize[2:0] = 111 TxData[63:8] is valid

TxSize is considered valid only when TxValid is asserted.

C.5.1.3.6 TxPrty[3:0] (transmit bus parity)

TxPrty indicates the parity calculated over the TxData bus. TxPrty is valid even when TxValid is deasserted. The parity calculation is such that odd parity is indicated on this signal.

TxPrty[0] = odd parity over TxData[15:0]
TxPrty[1] = odd parity over TxData[31:16]
TxPrty[2] = odd parity over TxData[47:32]
TxPrty[3] = odd parity over TxData[63:48]

The parity is used to detect interface error. It does not interfere with the data transfer.

C.5.1.3.7 TxSOP (transmit start of packet)

TxSOP is used to delineate the packet boundaries on the TxData bus. When TxSOP is high the start of the packet is present on the TxData bus. TxSOP is required to be present at the beginning of every packet and is considered valid only when TxValid is asserted.

This is a physical interface protocol signal for flow control at the interface level. The generation of this signal is as result the reception of a PHY_DATA.request and for the first valid data symbols are being transmitted across the physical interface.

C.5.1.3.8 TxEOP (transmit end of packet)

TxEOP is used to delineate the packet boundaries on the TxData bus. When TxEOP is high, the end of the packet is present on the TxData bus. TxEOP is required to be present at the end of every packets and is considered valid only when TxValid is asserted.

This is a physical interface protocol signal for flow control at the interface level. The generation of this signal is as result the reception of a PHY_DATA.request and for the last valid data symbols are being transmitted across the physical interface.

C.5.1.3.9 TxStart (transmit flow control frame start)

TxStart indicates when the start of the TxFull frame boundary. When TxStart is high The TxFull indicate PHY transmit FIFO full status for the first 4 ports. The next clock cycle indicates the FIFO status for the next 4 ports. It is time multiplexed.

This signal is synchronous to the RxClk

C.5.1.3.10 TxFull[3:0] (transmit flow control full)

TxFull is used by the PHY to signal the RS that there is no room to receive data from the RS for a port of a multiport PHY device. This signal is synchronous to the RxClk.

For single PHY support, TxFull [0] is used and the round robin scheme is disable.

The result of the TxFull status is used to generate PHY_READY.indicate.

C.5.1.3.11 RxClk (receive FIFO write clock)

RxClk is a continuous clock used to synchronize data transfer transactions between the PHY and the RS. RxClk may cycle at a rate up to 200 MHz.

C.5.1.3.12 RxValid (receive data valid)

When RxValid is deasserted the following signals are undefined: RxAddr, RxData, RxSOCP, RxEOP, RxError, and RxSize.

C.5.1.3.13 RxData[63:0] (receive packet data bus)

RxData is a 64-bit bus which carries the packet bytes that are read from the PHY. RxData is considered valid only when RxValid is asserted. Mapping of RxData signals is shown in Figure C.13

C.5.1.3.14 RxPrty (receive parity)

RxPrty indicates the parity calculated over the RxData bus. RxPrty is valid even when RxValid is deasserted. The parity calculation is such that odd parity is indicated on this signal.

RxPrty[0] = odd parity over RxData[15:0]
RxPrty[1] = odd parity over RxData[31:16]
RxPrty[2] = odd parity over RxData[47:32]
RxPrty[3] = odd parity over RxData[63:48]

C.5.1.3.15 RxSize[1:0] (receive word modulo)

RxSize is required for 64 bit RxData bus. RxSize indicates the number of valid bytes of data in RxData. The RxSize bus should always be zero except during the last word transfer of a packet on Rxdata. When RxEOP is asserted the number of valid packet data bytes on RxData is decoded from RxSize as:

RxSize[2:0] = 001	RxData[63:56] is valid
RxSize[2:0] = 010	RxData[63:48] is valid
RxSize[2:0] = 011	RxData[63:40] is valid
RxSize[2:0] = 100	RxData[63:32] is valid
RxSize[2:0] = 101	RxData[63:24] is valid
RxSize[2:0] = 110	RxData[6316] is valid
RxSize[2:0] = 111	RxData[63:8] is valid

RxSize is considered valid only when RxValid is asserted

C.5.1.3.16 RxSOCP (receive start of packet)

RxSOCP is used to delineate the packet boundaries on the RxData bus. When RxSOCP is high, the start of the packet is present on the RxData bus. RxSOCP is required to be present at the beginning of every packet and is considered valid only when RxValid is asserted.

C.5.1.3.17 RxEOP (receive end of packet)

RxEOP is used to delineate the packet boundaries on the RxData bus. When RxEOP is high, the end of the packet is present on the RxData bus. RxEOP is required to be present at the end of every packet and is considered valid only when RxValid is asserted.

C.5.1.3.18 RxError (receive error indicator)

RxError is used to indicate that the current packet is in error. RxError is asserted when RxEOP is asserted. Conditions that can cause RxError to be set may be, but are not limit to, FIFO overflow, or abort sequence.

C.5.1.4 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 7. Mappings for the following primitives are defined:

— PHY_DATA.request
— PHY_DATA.indicate
— PHY_READY.indicate.

C.5.1.4.1 Mapping of PHY_DATA.request

PHY_DATA.request (OUTPUT_FRAME, LENGTH)

The OUTPUT_FRAME is 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.

The LENGTH is used by the GRS. It is mapped to the PLI field of the core header.

C.5.1.4.2 Mapping of PHY_DATA.indicate

PHY_DATA.indicate (INPUT_FRAME, STATUS, LENGTH)

For the GRS, upon receiving a complete frame from the PHY. All valid octets received from RxSOCP and RxEOP formulates the INPUT_FRAME primitive less the GFP core header and payload header. The core header field PLI is mapped into the LENGTH primitive.

For SRS, upon receiving a complete frame from the PHY. All valid octets received from RxSOP and RxEOP formulates the INPUT_FRAME primitive.

The STATUS generates OK unless an active error signal is received on RxError.

C.5.1.4.3 Mapping of PHY_READY.indicate

PHY_READY.indicate (READY_STATUS)

The READY_STATUS attribute takes on the value of READY or NOT_READY.

C.5.1.4.3.1 When generated

Depending on single or multiple PHY mode, TxStart and TxFull are indications from the PHY to the RS that the PHY is ready to receive data on the TxData bus. Valid indication from the PHY is mapped to the PHY_READY.indicate primitive and sent to RS client.

C.5.1.4.3.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 6.

C.5.2 SRS and GRS 64-bit SPI datastream

There are no electrical difference and signaling protocol between the GRS and the SRS sublayer. The only difference is the physical frame format as specified in C.2.1 and C.2.2 .

C.5.3 Functional specifications

The SRS and GRS using SPI-4 Phase 1 interfaces shall meet the functional requirements of section 6 of the OIF SPI-4 Phase 1 implementation agreement.

C.5.4 Electrical specifications

The SRS and GRS using the SPI-4 Phase 1 interface shall meet the electrical timing requirements of sections 10, 11 and 12 of the OIF SPI-4 implementation agreement.

C.6 SRS and GRS using SPI-4 Level 2 interface

The SONET/SDH Reconciliation Sublayer and GFP Reconciliation Sublayer (GRS) may be implemented with an OIF SPI-4 Phase 2 interface.

SPI-4 is an interface for packet 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 packet'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 signal for the SPI4-02.0 interface and its interaction with the MAC physical layer service interface primitive. Any discrepancy in the signal description is not intentional. The OIF-SPI4-02.0 takes precedence over the subclause below.

C.6.1.1 Summary of major concepts

a) The SRS and GRS map the signals provided at the SPI-4 Phase 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 using the SPI-4 Phase 2 interface support full-duplex operation only.

Figure C.14 shows a schematic view of the SRS and GRS inputs and outputs using the 64-bit SPI-4 phase 2 interface.

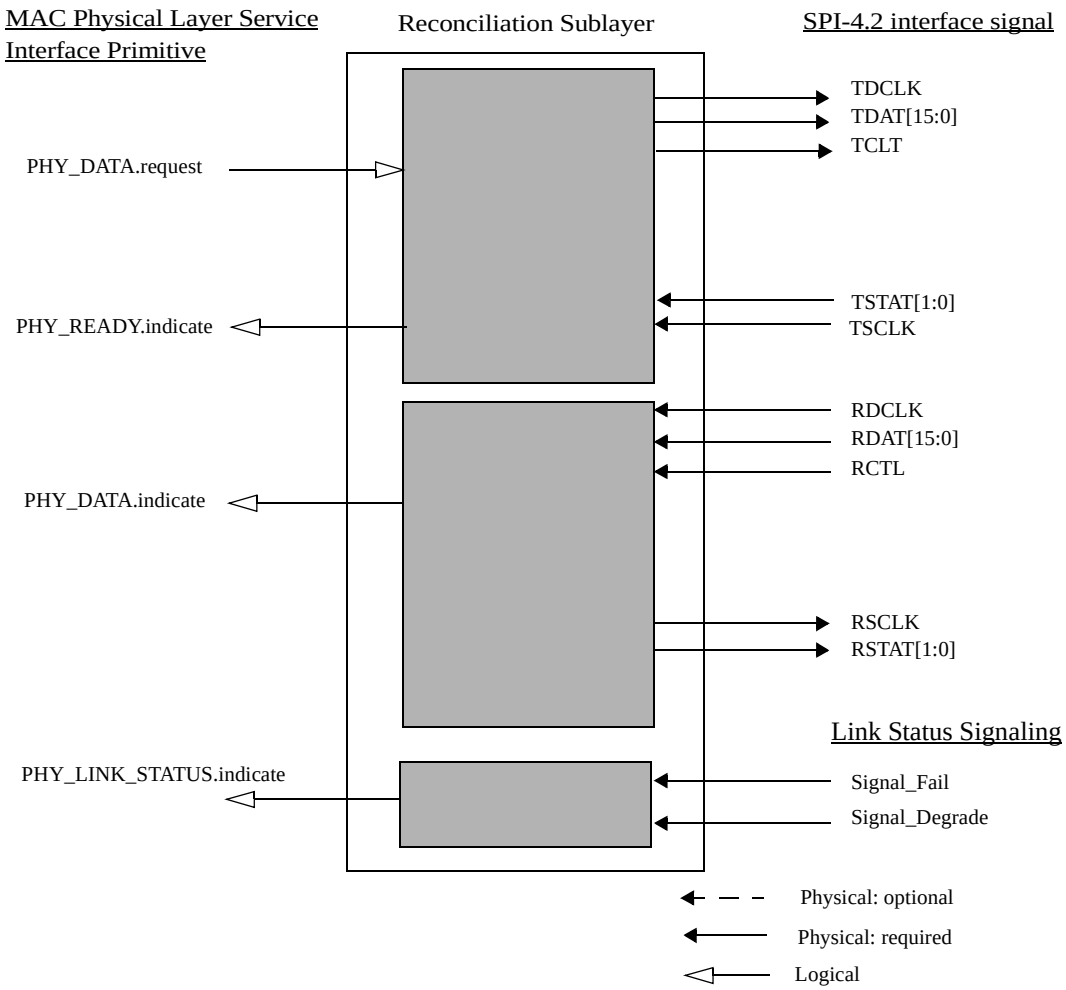


Figure C.14—SPI-4.2 Interface Signals

C.6.1.2 Rate of operation

The SRS and GRS using the SPI-4 Phase 2 interface are capable of supporting data rates of 622 Mbps to 10 Gbps.

.

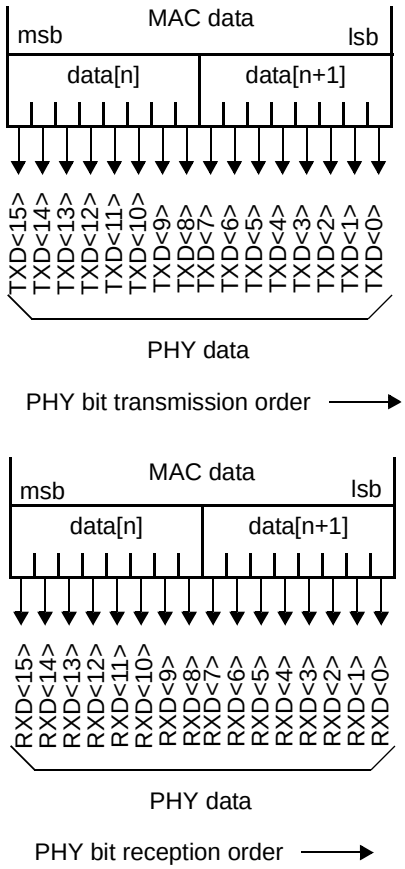


Figure C.15—SPI-4 Phase 2 transmit and receive signal mapping

C.6.1.3 SPI-4 Phase 2 structure

The SPI-4 Phase 2 interface signals are described in the following subclauses. Data and control lines are driven from the rising and falling edges of the clock (TDCLK).

C.6.1.3.1 TDCLK (transmit data clock)

TDCLK is a clock associated with TDAT 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.

C.6.1.3.2 TDAT[15:0] (transmit data)

TDAT is 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 when TCTL is high. The minimum data rate for TDAT is 622 Mb/s. Mapping of TDAT is shown in Figure C.15.

C.6.1.3.3 TCTL (transmit control)

TCTL is high when a control word is present on TDAT, otherwise it is low. TCTL is sourced by the MAC to the PHY.

C.6.1.3.4 TSCLK (transmit status clock)

TSCLK is a clock associated with TSTAT providing source-synchronous clocking. For LVTTL I/O a maximum clock rate restraint is ¼ that of the data path clock rate. LVDS I/O allows a maximum of that equal to the data path clock (double-edge clocking).

C.6.1.3.5 TSTAT[1:0] (transmit FIFO status)

TSTAT is a 2-bit bus used to carry round-robin FIFO status information, along with associated error detection and framing. The maximum data rate for TSTAT is dependent on the I/O type, either LVDS or LVTTL, and is limited to its respective TSCLK restraints. TSTAT is sourced by the PHY to the MAC. The FIFO status formats are:

TSTAT[1:0] = 11	Reserved for framing or to indicate a disabled status link.
TSTAT[1:0] = 10	SATISFIED
TSTAT[1:0] = 01	HUNGRY
TSTAT[1:0] = 00	STARVING

C.6.1.3.6 RDCLK (receive data clock)

RDCLK is a clock associated with RDAT 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.

C.6.1.3.7 RDAT[15:0] (receive data)

RDAT is a 16-bit bus which carries payload data and in-band control from the PHY to the Link Layer device. A control word is present on RDAT when RCTL is high. The minimum data rate for RDAT is 622 Mb/s. Mapping of RDAT is shown in Figure C.15

C.6.1.3.8 RCTL (receive control)

RCTL is high when a control word is present on RDAT, otherwise it is low. RCTL is sourced by the PHY to the MAC.

C.6.1.3.9 RSCLK (receive status clock)

RSCLK is a clock associated with RSTAT providing source-synchronous clocking. RSCLK is sourced by the Mac to the PHY. LVDS I/O allows a maximum of that equal to the data path clock (double-edge clocking).

C.6.1.3.10 RSTAT[1:0] (receive FIFO status)

RSTAT is a 2-bit bus used to carry round-robin FIFO status information, along with associated error detection and framing. The maximum data rate for RSTAT is dependent on the I/O type, either LVDS or LVTTL, and is limited to its respective RSCLK restraints. RSTAT is sourced by the Mac to the PHY. The FIFO status formats are:

RSTAT[1:0] = 11	Reserved for framing or to indicate a disabled status link.
RSTAT[1:0] = 10	SATISFIED
RSTAT[1:0] = 01	HUNGRY
RSTAT[1:0] = 00	STARVING

C.6.1.4 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 7. Mappings for the following primitives are defined:

— PHY_DATA.request
— PHY_DATA.indicate
— PHY_READY.indicate.

C.6.1.4.1 Mapping of PHY_DATA.request

PHY_DATA.request (OUTPUT_FRAME, LENGTH)

There are two attributes: INPUT_FRAME and LENGTH

The INPUT_FRAME is mapped from the received payload. The LENGTH applies to GRS only. It is mapped from the PLI field in the GFP core header.

C.6.1.4.1.1 When generated

This primitive is invoked by the RS after it had received a complete frame from the physical interface. The INPUT_FRAME is derived from the signals TDAT and TCTL, in accordance to SPI-4.2 specification.

C.6.1.4.1.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 6.

C.6.1.4.2 Mapping of PHY_DATA.indicate

PHY_DATA.indicate (INPUT_FRAME, STATUS, LENGTH)

PHY_DATA.indicate (INPUT_FRAME, STATUS,LENGTH)

There are three attributes: INPUT_FRAME, STATUS, and LENGTH

The INPUT_FRAME is mapped from the received payload. The LENGTH applies to GRS only. It is mapped from the PLI field in the GFP core header.

C.6.1.4.2.1 When generated

This primitive is invoked by the RS after it had received a complete frame from the physical interface. The INPUT_FRAME is derived from the signals RDAT[31:0], RSCLK, and RSTAT in accordance to SPI-4.2 specification.

C.6.1.4.2.2 Effect of receipt

The effect of receiving this primitive by the MAC sublayer is specified in Clause 6.

C.6.2 SRS and GRS SPI-4 Phase 2 datastream

There are no electrical difference and signaling protocol between the GRS and the SRS sublayer. The only difference is the physical frame format as specified in

C.6.3 Functional specifications

The SRS and GRS using SPI-4 Phase 2 interfaces shall meet the functional requirements of section 6 of the OIF SPI-4 Phase 2 implementation agreement

C.6.4 Electrical specifications

The SRS and GRS using the SPI-4 Phase 2 interface shall meet the electrical timing requirements of sections 6.4 and 6.5 of the OIF SPI-4 implementation agreement.

Annex D

(normative)

SNMP MIB definitions

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for WG review.
Draft 0.2, April 2002	Draft 0.2 for TF review, modified according to comments on D0.1.
Draft 0.3, June 2002	Draft 0.3 for WG review, modified according to comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for TF review, modified according to comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified according to comments on D1.0
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1

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.

D.2 The SNMP management framework

An introduction to the current SNMP Management Framework can be found in RFC 2570 [RFC2570].

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. A MIB conforming to the SMIv1 can be produced through the appropriate translations. The resulting translated MIB must be semantically equivalent, except where objects or events are omitted because no translation is possible (use of Counter64). 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.

D.3 Security considerations

There are number of 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.

There are a number of managed objects in the RPR MIB that may be considered to contain sensitive information.

Therefore, it may be important in some environments to control read access to these objects and possibly to even encrypt the values of these objects when sending them over the network via SNMP. Not all versions of SNMP provide features for such a secure environment.

SNMPv1 by itself is such an insecure environment. Even if the network itself is secure (for example by using IPSec), even then, there is no control as to who on the secure network is allowed to access and GET (read) the objects in this MIB.

It is recommended that the implementors consider the security features as provided by the SNMPv3 framework. Specifically, the use of the User-based Security Model RFC 2574 [RFC2574] and the View-based Access Control Model RFC 2575 [RFC2575] is recommended.

It is then a customer/user responsibility to ensure that the SNMP entity giving access to an instance of this MIB, is properly configured to give access to those objects only to those principals (users) that have legitimate rights to access them.

D.4 Structure of the MIB

This section describes the structure of the RPR MIB.

The MIB provides objects to configure and manage the RPR MAC, as defined in this standard. The structure is described in Table D.1, “Structure of the MIB,” on page 317.

D.5 Relationship to other MIBs

A system implementing the RPR MIB shall also implement (at least) the “interfaces” group defined in RFC 2863 [RFC2863].

D.5.1 Relationship to the Interfaces MIB

The Interfaces MIB [RFC2863] 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 RPR MAC interface. Each of these 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 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.

Table D.1—Structure of the MIB

rprIfTable			The rprIfTable table is used to configure parameters, which are applied to both spans of the RPR MAC.
	rprSpanTable		The rprSpanTable table contains the protection information on each span of the RPR MAC. The table is indexed by the ifIndex of the RPR interface and the span ID (East/West).
		rprTopoImageTable	The rprTopoImageTable table is a read-only table, and it contains the topology database that the RPR learned from the topology discovery protocol. A database is learned per ringlet, and it maintains the number of hops to each station on the ring, and the capability of each station (wrap support, jumbo frames support, etc.). The table is indexed by the ifIndex of the RPR interface and the ringlet and a row is reported for each station on the ringlet.
		rprProtectImageTable	The rprProtectImageTable table is a read-only table, and it contains the protection information on each interface of each station on the ring. The protection information is collected by monitoring the RPR protection messages that are sent around the ring by RPR stations. The table is indexed by the ifIndex of the RPR interface and the ringlet and a row is created for each station on the ringlet.
		rprFairnessTable	The rprFairnessTable table is used to configure RPR fairness parameters, low-pass filter coefficients and transit buffer thresholds. The table enables the configuration of different parameters to each span of the RPR MAC. The table is indexed by the ifIndex of the RPR interface and the ringlet ID.
	rprOamEchoTable		The rprOamEchoTable table is used to operate and monitor the echo request/response between two stations on the ring. The table enables the management system to define the destination of the echo request, the service class of the request/response, the ringlets on which the request should be sent, the response that should be replied, and the number of consecutive requests to generate. The table enables also setting the time to declare the timeout of the echo response. The table is indexed by the ifIndex of the RPR interface.
	Statistics		The remaining twelve tables are used to collect current, past intervals, day, total statistics and error counters from the client and span interfaces of the RPR MAC. Statistics are collected per service class, for Unicast and Multicast+Broadcast packets/octets.

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 an interface stacks for both east and west spans.

D.5.1.3 Specific Interface MIB Objects

The following table provides specific implementation guidelines for applying the interface group objects to RPR media.

Table D.2—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	Refer to [RFC2863].
ifType	The RPR ifType MUST 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	Represents the current operational speed of the interface in bits per second
ifPhysAddress	The RPR MAC address
ifAdminStatus	Write access is not required. Support for 'testing' is not required.
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	rprClientStatsInUcastClassAOctets + rprClientStatsInUcastClassBCirOctets + rprClientStatsInUcastClassBEirOctets + rprClientStatsInUcastClassCOctets + rprClientStatsInUcastClassAOctets + rprClientStatsInMcastClassBCirOctets + rprClientStatsInMcastClassBEirOctets + rprClientStatsInMcastClassCOctets
ifInUcastPkts	rprClientStatsInUcastClassAPkts + rprClientStatsInUcastClassBCirPkts + rprClientStatsInUcastClassBEirPkts + rprClientStatsInUcastClassCPkts
ifInDiscards	The number of frames from the client discarded on reception.
ifInErrors	The number of frames from the client with reception errors.
ifInUnknownProtos	Not used, replaced by this MIB module.

Table D.2—Specific Interface MIB Objects *(continued)*

Object	Guideline
ifOutOctets	rprClientStatsOutUcastClassAOctets + rprClientStatsOutUcastClassBCirOctets + rprClientStatsOutUcastClassBEirOctets + rprClientStatsOutUcastClassCOctets + rprClientStatsOutMcastClassAOctets + rprClientStatsOutMcastClassBCirOctets + rprClientStatsOutMcastClassBEirOctets + rprClientStatsOutMcastClassCOctets
ifOutUcastPkts	rprClientStatsOutUcastClassAPkts + rprClientStatsOutUcastClassBCirPkts + rprClientStatsOutUcastClassBEirPkts + rprClientStatsOutUcastClassCPkts
ifOutDiscards	The number of frames discarded on transmission to the client.
ifOutErrors	The number of frames to the client discarded due to transmission errors.
ifName	Locally-significant textual name for the interface
ifInMulticastPkts ifInBroadcastPkts	rprClientStatsInMcastClassAPkts + rprClientStatsInMcastClassBCirPkts + rprClientStatsInMcastClassBEirPkts + rprClientStatsInMcastClassCPkts
ifOutMulticastPkts ifOutBroadcastPkts	rprClientStatsOutMcastClassAPkts + rprClientStatsOutMcastClassBCirPkts + rprClientStatsOutMcastClassBEirPkts + rprClientStatsOutMcastClassCPkts
ifHCInOctets	rprClientStatsInUcastClassAOctets + rprClientStatsInUcastClassBCirOctets + rprClientStatsInUcastClassBEirOctets + rprClientStatsInUcastClassCOctets + rprClientStatsInUcastClassAOctets + rprClientStatsInMcastClassBCirOctets + rprClientStatsInMcastClassBEirOctets + rprClientStatsInMcastClassCOctets
ifHCOutOctets	rprClientStatsOutUcastClassAOctets + rprClientStatsOutUcastClassBCirOctets + rprClientStatsOutUcastClassBEirOctets + rprClientStatsOutUcastClassCOctets + rprClientStatsOutMcastClassAOctets + rprClientStatsOutMcastClassBCirOctets + rprClientStatsOutMcastClassBEirOctets + rprClientStatsOutMcastClassCOctets
ifHCInUcastPkts	rprClientStatsInUcastClassAPkts + rprClientStatsInUcastClassBCirPkts+ rprClientStatsInUcastClassBEirPkts+ rprClientStatsInUcastClassCPkts
ifHCInMulticastPkts ifHCInBroadcastPkts	rprClientStatsInMcastClassAPkts + rprClientStatsInMcastClassBCirPkts + rprClientStatsInMcastClassBEirPkts + rprClientStatsInMcastClassCPkts

Table D.2—Specific Interface MIB Objects *(continued)*

Object	Guideline
ifHCOutUcastPkts	rprClientStatsOutUcastClassAPkts + rprClientStatsOutUcastClassBCirPkts+ rprClientStatsOutUcastClassBEirPkts+ rprClientStatsOutUcastClassCPkts
ifHCOutMulticastPkts ifHCOutBroadcastPkts	rprClientStatsOutMcastClassAPkts + rprClientStatsOutMcastClassBCirPkts + rprClientStatsOutMcastClassBEirPkts + rprClientStatsOutMcastClassCPkts
ifLinkUpDownTrapEnable	Refer to [RFC2863]. Default is 'enabled'
ifHighSpeed	Represents the current operational speed in millions of bits per second.
ifPromiscuousMode	Refer to [RFC2863].
ifConnectorPresent	Always false.
ifAlias	Refer to [RFC2863].
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 re-initialization of the local management subsystem, then this object contains a zero value.
ifStackHigherLayer ifStackLowerLayer ifStackStatus	Refer to section E.5.1.1.
ifRcvAddressAddress ifRcvAddressStatus ifRcvAddressType	Refer to [RFC2863].

D.6 Definitions for the RPR MIB

```
RPR-MIB DEFINITIONS ::= BEGIN


IMPORTS

   MODULE-IDENTITY, OBJECT-TYPE, Integer32, Counter64,
   Counter32, Unsigned32
       FROM SNMPv2-SMI
   MODULE-COMPLIANCE, OBJECT-GROUP
       FROM SNMPv2-CONF
   MacAddress, TimeStamp,TEXTUAL-CONVENTION, TruthValue
       FROM SNMPv2-TC
   InterfaceIndex, InterfaceIndexOrZero
       FROM IF-MIB;

    rprMib  MODULE-IDENTITY
    LAST-UPDATED “200212081200Z”  -- 8 December 2002 12:00:00 EST
    ORGANIZATION “IEEE 802.17 Working Group”
    CONTACT-INFO “stds-802-17@ieee.org”
    DESCRIPTION
        “The resilient packet ring MIBs for IEEE 802.17”
    ::= {iso(1) std(0) iso8802(8802) ieee802dot17(17) ieee802dot17mibs(1) rprMIB(1)}


--
--   Textual Conventions used in this MIB
--

RprStatsClear ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        “This TC allows for some or all of the statistics for
         this RPR MAC to be cleared.

         clearSpecificInterval clears the interval indicated by
         rprIfStatsIntervalClear”
    SYNTAX INTEGER {
        idle                 (1),
        clearAll             (2),
        clearCurrent         (3),
        clearIntervals       (4),
        clearSpecificInterval (5)
        }

RprRinglet ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        “Indicates the ringlet of the RPR ring. Each RPR
         ring consists of two ringlets, ringlet 0 and ringlet 1.
         ringlet 0 is the transmission of the east span and the
         reception of the west span,
         ringlet 1 is the transmission of the west span and the
         reception of the east span”
    SYNTAX INTEGER {
        ringlet0    (1),
        ringlet1    (2)
        }

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)
        }

RprServiceClass ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
       "Indicates the service classes that the RPR supports."
    SYNTAX INTEGER {
         classA   (1),
         classB   (2),
         classC   (3)
        }

RprProtectionControl ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
       "Indicates the protection mode"
    SYNTAX INTEGER {
         protected   (1),
         unprotected (2)
        }

RprMacCapabilties ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
       "Indicates the capabilities that this MAC supports"
    SYNTAX BITS {
         singleChokeFairness (0),
         multiChokeFairness  (1),
         jumboFrames         (2),
         wrapProtection      (3)
        }

RprRingStationCount ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Indicates the number of stations on the RPR ring"
    SYNTAX Integer32 (0..255)

RprRingHopCount ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Indicates the number of hops to a station on the RPR ring"
    SYNTAX Integer32 (0..255)

RprHoldOffTimer ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Indicates the period in millisecond units that RPR layer
         gives to lower layer to perform protection, before it
         activate its protection mechanism."
    SYNTAX Integer32 (0..500)

RprProtectionWtr ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Indicates the period in seconds to remain in the
```

```
          protection state, after the cause of an automatic
          protection is removed. This mechanism prevents protection
          switch oscillations."
      SYNTAX Integer32 (0..1440)

RprFastTimer ::= TEXTUAL-CONVENTION
      STATUS current
      DESCRIPTION
          "Specifies the fast timer range and resolution, in 1msec units.
           Fast timer is used for both protection and topology protocols."
      SYNTAX Integer32 (1..20)

RprSlowTimer ::= TEXTUAL-CONVENTION
      STATUS current
      DESCRIPTION
          "Specifies the slow timer range and resolution, in 50msec units.
           Slow timer is used for both protection and topology protocols."
      SYNTAX Integer32 (1..200)

RprKeepaliveTimout ::= TEXTUAL-CONVENTION
      STATUS current
      DESCRIPTION
          "Indicates the timer in millisecond units to declare
           keepalive timeout."
      SYNTAX Integer32 (1..10)

RprFairnessMode ::= TEXTUAL-CONVENTION
      STATUS current
      DESCRIPTION
          "Indicates the operational mode of the fairness algorithm"
      SYNTAX INTEGER {
          aggressive   (1),
          conservative (2)
          }

RprFairnessWeight ::= TEXTUAL-CONVENTION
      STATUS current
      DESCRIPTION
          "Indicates the weight of the station span.
           used for weighted fairness."
      SYNTAX Integer32 (1..255)

RprDataRate ::= TEXTUAL-CONVENTION
      STATUS current
      DESCRIPTION
          "Indicates the rate range in Mbps units used by the
           RPR fairness algorithm."
      SYNTAX Integer32 (0..40000)

RprFairnessCoef ::= TEXTUAL-CONVENTION
      STATUS current
      DESCRIPTION
          "Indicates the coefficients range used by the
           RPR fairness algorithm.
           Values are recommended to be in power of 2 (2^N)"
      SYNTAX Integer32 (1..65536)

RprProtectionCommand ::= TEXTUAL-CONVENTION
      STATUS current
      DESCRIPTION
          "An RPR ring provides redundancy and protection from a
           failed station or a link/fiber cut through the use of
           protection modes that are automatic or user initiated.
           Automatic protection modes take effect when the ring
           detects an event, a fiber cut, or a station failure and
```

```
            remain in effect until the wait-to-restore value expires.
            As the protection requests travel around the ring, the
            protection hierarchy is applied. All stations are signalled
            on the protection messaging channel. If a station or fiber
            facility failure is detected, traffic going toward or
            coming from the failure direction is wrapped (looped)
            back to go in opposite direction on the other ring or
            dropped by the protecting interface (depend on the
            protection method). The wrap/drop takes place on the
            stations adjacent to the failure.

            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 doesn't have precedence over automatic protection,
            and therefore it can't 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."
        SYNTAX INTEGER {
            idle            (1),
            manualSwitch    (2),
            forcedSwitch    (3)
            }

    RprProtectionStatus ::= TEXTUAL-CONVENTION
        STATUS current
        DESCRIPTION
            "Indicates the current protection status of the interface
             span.
             The status values are (arranged in ascending priority
             order) :

             noRequest
             No protection request on the interface, the protection
             status is idle.

             waitToRestore
             The condition for an automatic protection was cleared and
             the interface span is engaged in a wait to restore period.

             manualSwitch
             A user initiated manual switch (via the
             rprMacProtectionCommand) on the interface. Both spans of
             the specified span performs protection.

             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
         rprMacProtectionCommand) on the interface. Both spans of
         the specified span performs protection."
    SYNTAX BITS {
        noRequest       (0),
        waitToRestore   (1),
        manualSwitch    (2),
        signalDegraded  (3),
        signalFailed    (4),
        forcedSwitch    (5)
        }

RprOamEchoRinglet ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
       "Indicates the ringlet on which the echo request/response
        is sent/replied.
        The valid values for this object are:
        (1) ringlet 0,(2) ringlet 1 (3) Default."
    SYNTAX INTEGER {
        ringlet0 (1),
        ringlet1 (2),
        default  (3)
        }

RprOamEchoTimeout ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "Indicates the timer to declare echo timeout, in
         10usec units."
    SYNTAX Integer32 (0..65535)

RprOamControl ::= TEXTUAL-CONVENTION
   STATUS current
    DESCRIPTION
       "Control of an OAM Echo action.
        The valid values for this object are:
        (1) idle (2) active (3) abort. "
    SYNTAX INTEGER {
         idle     (1),
         active   (2),
         abort    (3)
         }

RprOamEchoStatus ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
       "Status of an OAM Echo action.
        The valid values for this object are:
        (1) unknown (2) inProcess (3) error (4) success."
    SYNTAX INTEGER {
         unknown   (1),
         inProcess (2),
         error     (3),
         success   (4)
         }

--
-- Object groups
--

rprObjects            OBJECT IDENTIFIER ::= { rprMib 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 }


--
-- Conformance
--

rprConformance        OBJECT IDENTIFIER ::= { rprMib 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."
    ::= { 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."
    INDEX { rprIfIndex }
    ::= { rprIfTable 1 }

RprIfEntry ::= SEQUENCE {
    rprIfIndex                  InterfaceIndex,
    rprIfStationsOnRing         RprRingStationCount,
    rprIfReversionMode          TruthValue,
    rprIfHoldOffTimer           RprHoldOffTimer,
    rprIfProtectionWTR          RprProtectionWtr,
    rprIfFastTimer              RprFastTimer,
    rprIfSlowTimer              RprSlowTimer,
    rprIfKeepaliveTimout        RprKeepaliveTimout,
    rprIfDropBadFcsControl      TruthValue,
    rprIfFairnessMode           RprFairnessMode,
    rprIfPtqSize                Unsigned32,
    rprIfStqSize                Unsigned32,
    rprIfWrapControl            TruthValue,
    rprIfMacCapabilities        RprMacCapabilties,
    rprIfRingCapabilities       RprMacCapabilties,
    rprIfStatsClear             RprStatsClear,
    rprIfStatsIntervalClear     Integer32,
    rprIfTimeElapsed            Integer32,
    rprIfValidIntervals         Integer32
    }

rprIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ifIndex of this RPR interface."
```

```
    ::= { rprIfEntry 1 }

rprIfStationsOnRing OBJECT-TYPE
    SYNTAX        RprRingStationCount
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of stations in the RPR ring. When the interface
         is down the value is 1."
    REFERENCE
        "IEEE 802.17 Subclause 10.2.5"
    ::= { rprIfEntry 2 }

rprIfReversionMode OBJECT-TYPE
    SYNTAX        TruthValue
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "The reversion mode for flows that uses steering protection
         method.
         Revertive flows will return to their primary path after
         WTR interval expires."
    REFERENCE
        "IEEE 802.17 Subclause 11.4"
    ::= { rprIfEntry 3 }

rprIfHoldOffTimer OBJECT-TYPE
    SYNTAX        RprHoldOffTimer
    UNITS         "milliseconds"
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "The period that RPR gives to lower layer to perform
         protection, before it activate its protection mechanism."
    REFERENCE
        "IEEE 802.17 Subclause 11.4.3"
    DEFVAL { 0 }
    ::= { rprIfEntry 4 }

rprIfProtectionWTR OBJECT-TYPE
    SYNTAX        RprProtectionWtr
    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."
    REFERENCE
        "IEEE 802.17 Subclause 11.5"
    DEFVAL { 10 }
    ::= { rprIfEntry 5 }

rprIfFastTimer OBJECT-TYPE
    SYNTAX        RprFastTimer
    UNITS         "1 millisecond"
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Indicates the fast timer value in 1msec units.
         The fast timer is used for topology and protection protocols."
    REFERENCE
        "IEEE 802.17 Subclause 11.7.1"
    DEFVAL { 10 }
```

```
    ::= { rprIfEntry 6 }

rprIfSlowTimer OBJECT-TYPE
    SYNTAX        RprSlowTimer
    UNITS         "50 milliseconds"
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Indicates the slow timer value in 50msec units.
         The slow timer is used for topology and protection protocols."
    REFERENCE
        "IEEE 802.17 Subclause 10.5.1, and 11.7.1"
    DEFVAL { 2 }
    ::= { rprIfEntry 7 }

rprIfKeepaliveTimout OBJECT-TYPE
    SYNTAX        RprKeepaliveTimout
    UNITS         "milliseconds"
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Indicates the timer to declare keepalive timeout, multiples
         of milliseconds."
    REFERENCE
        "IEEE 802.17 Subclause 11.5"
    DEFVAL { 3 }
    ::= { rprIfEntry 8 }

rprIfDropBadFcsControl OBJECT-TYPE
    SYNTAX        TruthValue
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Indicates whether the RPR MAC drops packets with bad FCS
         that are destined to its client."
    REFERENCE
        "IEEE 802.17 Subclause 5.3.2.2"
    DEFVAL { false }
    ::= { rprIfEntry 9 }

rprIfFairnessMode OBJECT-TYPE
    SYNTAX        RprFairnessMode
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "The operational mode of the RPR fairness algorithm."
    REFERENCE
        "IEEE 802.17 Subclause 9.2.9, 9.2.15"
    DEFVAL { aggressive }
    ::= { rprIfEntry 10 }

rprIfPtqSize OBJECT-TYPE
    SYNTAX        Unsigned32
    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 6.5"
    ::= { rprIfEntry 11 }

rprIfStqSize OBJECT-TYPE
    SYNTAX        Unsigned32
    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 6.5"
    ::= { rprIfEntry 12 }

rprIfWrapControl OBJECT-TYPE
    SYNTAX        TruthValue
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Enables control of wrap protection through SNMP manager."
    REFERENCE
        "IEEE 802.17 Subclause 11.6.1.1"
    DEFVAL { false }
    ::= { rprIfEntry 13 }

rprIfMacCapabilities OBJECT-TYPE
    SYNTAX        RprMacCapabilties
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The capabilities supported by this RPR MAC."
    REFERENCE
        "IEEE 802.17 Subclause 10.4.2.3"
    ::= { rprIfEntry 14 }

rprIfRingCapabilities OBJECT-TYPE
    SYNTAX        RprMacCapabilties
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The summary of the ring capabilities collected through
         the topology discovery protocol."
    REFERENCE
        "IEEE 802.17 Subclause 10.4.2.3"
    ::= { rprIfEntry 15 }

rprIfStatsClear OBJECT-TYPE
    SYNTAX        RprStatsClear
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "This attribute allows for some or all of the statistics for
         this RPR MAC to be cleared.

         clearSpecificInterval clears the interval indicated by
         rprIfStatsIntervalClear"
    DEFVAL { idle }
    ::= { rprIfEntry 16 }

rprIfStatsIntervalClear OBJECT-TYPE
    SYNTAX        Integer32 (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 rprIfStatsClear to clearSpecificInterval will
         clear the interval that indicated by rprIfStatsIntervalClear"
```

```
        ::= { rprIfEntry 17 }

    rprIfTimeElapsed OBJECT-TYPE
        SYNTAX        Integer32 (0..899)
        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."
        ::= { rprIfEntry 18 }

    rprIfValidIntervals OBJECT-TYPE
        SYNTAX        Integer32 (0..96)
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of previous 15-minute measurement intervals for
             which data was collected.
             An RPR interface must be capable of supporting up to 96
             intervals.
             The value will be 96 unless the measurement was restarted
             within the last (96*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."
        ::= { rprIfEntry 19 }

    --
    -- RPR span table
    --

    rprSpanTable OBJECT-TYPE
        SYNTAX        SEQUENCE OF RprSpanEntry
        MAX-ACCESS    not-accessible
        STATUS        current
        DESCRIPTION
            "The RPR interface Span table."
        ::= { rprGeneral 2 }

    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,
        rprSpanSpeed                        Unsigned32,
        rprSpanNeighbor                     MacAddress,
        rprSpanProtectionState              TruthValue,
        rprSpanProtectionCommand            RprProtectionCommand,
        rprSpanProtectionStatus             RprProtectionStatus,
        rprSpanProtectionCount              Counter32,
        rprSpanProtectionDuration           Counter32,
        rprSpanProtectionLastActivationTime TimeStamp,
        rprSpanStatsDiscontinuityTime       TimeStamp
```

```
    }

rprSpanIfIndex  OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        “The ifIndex of this RPR interface.”
    ::= { rprSpanEntry 1 }

rprSpanId OBJECT-TYPE
    SYNTAX        RprSpan
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        “The Span for this entry.”
    REFERENCE
        “IEEE 802.17 Subclause 1.4”
    ::= { 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
        “IEEE 802.17 Subclause 1.4”
    ::= { rprSpanEntry 3 }

rprSpanSpeed OBJECT-TYPE
    SYNTAX      Unsigned32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
           “The nominal bandwidth of the interface in units of Mbps
            (1,000,000 bits per second). If this object reports a
            value of `n’ then the speed of the interface is somewhere in
            the range of `n-500,000’ to `n+499,999’.”
    REFERENCE
        “IEEE 802.17 Subclause 7.3, 7.4”
    ::= { rprSpanEntry 4 }

rprSpanProtectionState OBJECT-TYPE
    SYNTAX        TruthValue
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “Indicates if protection is currently active on this span.”
    REFERENCE
        “IEEE 802.17 Subclause 11.6.1.2”
    ::= { rprSpanEntry 5 }

rprSpanNeighbor OBJECT-TYPE
    SYNTAX        MacAddress
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The 48-bit MAC address of the neighbor.”
    REFERENCE
        “IEEE 802.17 Subclause 10.4.2.4”
    ::= { rprSpanEntry 6 }
```

```
rprSpanProtectionCommand OBJECT-TYPE
    SYNTAX        RprProtectionCommand
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        “The protection mode requested by management for the local
         station that can affect the rprMacProtectionStatus of the RPR
         station, according to the set of rules describing the RPR
         protection.

         When read this object returns the last command written or
         idle if no command has been written to this interface
         span since initialization.

         The return of the last command written does not imply that
         this command is currently in effect. This request may have
         been preempted by a higher priority event, for example a
         manual protection is preempted by a failure. In order to
         determine the current state of the interface span and the
         station it is necessary to read the objects
         rprMacProtectionStatus and rprMacProtectionActive.

         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 an interface that has no pending protection
         command, has no affect. An idle clears an active WTR state.”
    REFERENCE
        “IEEE 802.17 Subclause 11.5”
    DEFVAL { idle }
    ::= { rprSpanEntry 7 }

rprSpanProtectionStatus OBJECT-TYPE
    SYNTAX        RprProtectionStatus
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The status of protection on this span.

         There is no pending of the waitToRestore status, that is
         if the restore period has been preempted by a higher
         priority event, when the event clears the restore does not
         continue.”
    REFERENCE
        “IEEE 802.17 Subclause 11.5”
    ::= { rprSpanEntry 8 }

rprSpanProtectionCount OBJECT-TYPE
    SYNTAX        Counter32
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transitions from idle state to active
         protection state.”
    ::= { rprSpanEntry 9 }

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."
    ::= { rprSpanEntry 10 }

rprSpanProtectionLastActivationTime OBJECT-TYPE
    SYNTAX        TimeStamp
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The value of sysUpTime at the time of the last protection
         activation."
    ::= { rprSpanEntry 11 }

rprSpanStatsDiscontinuityTime OBJECT-TYPE
    SYNTAX        TimeStamp
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
       "The value of sysUpTime on the most recent occasion at
        which any one or more of this span's
        Counter64 suffered a discontinuity. If no such
        discontinuities have occurred since the last re-
        initialization of the local management subsystem, then
        this object contains a zero value."
    ::= { rprSpanEntry 12 }


--
-- 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."
    ::= { 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."
    INDEX { rprTopoImageIfIndex,
            rprTopoImageRinglet,
            rprTopoImageHops }
    ::= { rprTopoImageTable 1 }

RprTopoImageEntry ::= SEQUENCE {
    rprTopoImageIfIndex                 InterfaceIndex,
    rprTopoImageRinglet                 RprRinglet,
    rprTopoImageHops                    RprRingHopCount,
    rprTopoImageMacAddress              MacAddress,
    rprTopoImageCapabilites             RprMacCapabilties,
    rprTopoImageFairnessRinglet0Weight  RprFairnessWeight,
    rprTopoImageFairnessRinglet1Weight  RprFairnessWeight,
    rprTopoImageRinglet0Neighbor        MacAddress,
    rprTopoImageRinglet1Neighbor        MacAddress,
    rprTopoImageRinglet0ReservedRate    RprDataRate,
    rprTopoImageRinglet1ReservedRate    RprDataRate
```

```
    }

rprTopoImageIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        “The ifIndex of this RPR interface.”
    ::= { rprTopoImageEntry 1 }

rprTopoImageRinglet OBJECT-TYPE
    SYNTAX        RprRinglet
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        “The ringlet for which this row contains information.”
    ::= { rprTopoImageEntry 2 }

rprTopoImageHops OBJECT-TYPE
    SYNTAX        RprRingHopCount
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        “The number of hops to this station on this ringlet.”
    REFERENCE
        “IEEE 802.17 Subclause 10.2.6”
    ::= { rprTopoImageEntry 3 }

rprTopoImageMacAddress OBJECT-TYPE
    SYNTAX        MacAddress
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The 48-bit MAC address of the station.”
    REFERENCE
        “IEEE 802.17 Subclause 10.2.6”
    ::= { rprTopoImageEntry 4 }

rprTopoImageCapabilites OBJECT-TYPE
    SYNTAX        RprMacCapabilties
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The capabilities of this station.”
    REFERENCE
        “IEEE 802.17 Subclause 10.4.2.3”
    ::= { rprTopoImageEntry 5 }

rprTopoImageFairnessRinglet0Weight OBJECT-TYPE
    SYNTAX        RprFairnessWeight
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The weight of the station on ringlet 0.
         used for weighted fairness.”
    REFERENCE
        “IEEE 802.17 Subclause 10.4.2.1”
    ::= { rprTopoImageEntry 6 }

rprTopoImageFairnessRinglet1Weight OBJECT-TYPE
    SYNTAX        RprFairnessWeight
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The weight of the station on ringlet 1.
```

```
          used for weighted fairness."
    REFERENCE
        "IEEE 802.17 Subclause 10.4.2.1"
    ::= { rprTopoImageEntry 7 }

rprTopoImageRinglet0Neighbor OBJECT-TYPE
    SYNTAX        MacAddress
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The 48-bit MAC address of the neighbor on ringlet0 span."
    REFERENCE
        "IEEE 802.17 Subclause 10.4.2.4"
    ::= { rprTopoImageEntry 8 }

rprTopoImageRinglet1Neighbor OBJECT-TYPE
    SYNTAX        MacAddress
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The 48-bit MAC address of the neighbor on ringlet1 span."
    REFERENCE
        "IEEE 802.17 Subclause 10.4.2.4"
    ::= { rprTopoImageEntry 9 }

rprTopoImageRinglet0ReservedRate OBJECT-TYPE
    SYNTAX        RprDataRate
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "Configured rate of A0 traffic on ringlet0 in Mbps units."
    REFERENCE
        "IEEE 802.17 Subclause 10.4.2.2"
    ::= { rprTopoImageEntry 10 }

rprTopoImageRinglet1ReservedRate OBJECT-TYPE
    SYNTAX        RprDataRate
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "Configured rate of A0 traffic on ringlet1 in Mbps units."
    REFERENCE
        "IEEE 802.17 Subclause 10.4.2.2"
    ::= { rprTopoImageEntry 11 }

--
-- The RPR Protection Image table
--

rprProtectImageTable OBJECT-TYPE
    SYNTAX        SEQUENCE OF RprProtectImageEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "A table of RPR protection per RPR span."
    ::= { rprProtocols 2 }

rprProtectImageEntry OBJECT-TYPE
    SYNTAX        RprProtectImageEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "A protection information of a particular RPR interfaces
        attachment to a ringlet."
    INDEX { rprProtectImageIfIndex,
```

```
            rprProtectImageSpan,
            rprProtectImageHops }
    ::= { rprProtectImageTable 1 }

RprProtectImageEntry ::= SEQUENCE {
    rprProtectImageIfIndex              InterfaceIndex,
    rprProtectImageSpan                 RprSpan,
    rprProtectImageHops                 RprRingHopCount,
    rprProtectImageActive               TruthValue,
    rprProtectImageStatus               RprProtectionStatus,
    rprProtectImageLastChange           TimeStamp
    }

rprProtectImageIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    ::= { rprProtectImageEntry 1 }

rprProtectImageSpan OBJECT-TYPE
    SYNTAX        RprSpan
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The span for which this row contains information."
    ::= { rprProtectImageEntry 2 }

rprProtectImageHops OBJECT-TYPE
    SYNTAX        RprRingHopCount
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The number of hops to this station on this span."
    REFERENCE
        "IEEE 802.17 Subclause 10.2.6"
    ::= { rprProtectImageEntry 3 }

rprProtectImageActive OBJECT-TYPE
    SYNTAX        TruthValue
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "A boolean flag to indicate if this span of the RPR
         interface is currently engaged in active protection.

         This object summarizes the rprProtectImageStatus, it set
         to True if rprProtectImageStatus different from noRequest"
    REFERENCE
        "IEEE 802.17 Subclause 10.2.6"
    ::= { rprProtectImageEntry 4 }

rprProtectImageStatus OBJECT-TYPE
    SYNTAX        RprProtectionStatus
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The current protection mode for the interface span.

         There is no pending of the waitToRestore status, that is
         if the restore period has been preempted by a higher
         priority event, when the event clears the restore does not
         continue."
    REFERENCE
```

```
        "IEEE 802.17 Subclause 10.2.6"
    ::= { rprProtectImageEntry 5 }

rprProtectImageLastChange OBJECT-TYPE
    SYNTAX        TimeStamp
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The value of sysUpTime at the time of the last protection
         status change."
    ::= { rprProtectImageEntry 6 }


--
-- 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."
    ::= { rprProtocols 3 }

rprFairnessEntry OBJECT-TYPE
    SYNTAX        RprFairnessEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "A fairness parameters for a particular RPR interface."
    INDEX { rprFairnessIfIndex,
            rprFairnessRinglet }
    ::= { rprFairnessTable 1 }

RprFairnessEntry ::= SEQUENCE {
    rprFairnessIfIndex               InterfaceIndex,
    rprFairnessRinglet               RprRinglet,
    rprFairnessRingletWeight         RprFairnessWeight,
    rprFairnessReservedRate          RprDataRate,
    rprFairnessMaxAllowed            RprDataRate,
    rprFairnessAgeCoef               RprFairnessCoef,
    rprFairnessRampCoef              RprFairnessCoef,
    rprFairnessLpCoef                RprFairnessCoef,
    rprFairnessFullThreshold         Unsigned32,
    rprFairnessHighThreshold         Integer32,
    rprFairnessLowThreshold          Integer32
    }

rprFairnessIfIndex  OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    ::= { rprFairnessEntry 1 }

rprFairnessRinglet OBJECT-TYPE
    SYNTAX        RprRinglet
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ringlet for which this row contains information."
    ::= { rprFairnessEntry 2 }
```

```
rprFairnessRingletWeight OBJECT-TYPE
    SYNTAX        RprFairnessWeight
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        “The weight of the station, used for weighted fairness.”
    REFERENCE
        “IEEE 802.17 Subclause 9.2.53”
    DEFVAL { 1 }
    ::= { rprFairnessEntry 3 }

rprFairnessReservedRate OBJECT-TYPE
    SYNTAX        RprDataRate
    UNITS         “Mbps”
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        “Configured rate of A0 traffic in this span in Mbps units”
    REFERENCE
        “IEEE 802.17 Subclause 9.2.47”
    DEFVAL { 0 }
    ::= { rprFairnessEntry 4 }

rprFairnessMaxAllowed OBJECT-TYPE
    SYNTAX        RprDataRate
    UNITS         “Mbps”
    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 LINK_RATE.”
    REFERENCE
        “IEEE 802.17 Subclause 9.2.33”
    ::= { rprFairnessEntry 5 }

rprFairnessAgeCoef OBJECT-TYPE
    SYNTAX        RprFairnessCoef
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        “The AGECOEF used by the fairness algorithm on this span.”
    REFERENCE
        “IEEE 802.17 Subclause 9.2.8”
    DEFVAL { 4 }
    ::= { rprFairnessEntry 6 }

rprFairnessRampCoef OBJECT-TYPE
    SYNTAX        RprFairnessCoef
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        “The RAMPCOEF used by the fairness algorithm on this span.”
    REFERENCE
        “IEEE 802.17 Subclause 9.2.41”
    DEFVAL { 64 }
    ::= { rprFairnessEntry 7 }

rprFairnessLpCoef OBJECT-TYPE
    SYNTAX        RprFairnessCoef
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        “The LPCOEF used by the fairness algorithm on this span.”
    REFERENCE
```

```
        "IEEE 802.17 Subclause 9.2.27"
    DEFVAL { 64 }
    ::= { rprFairnessEntry 8 }

rprFairnessFullThreshold OBJECT-TYPE
    SYNTAX        Unsigned32
    UNITS         "MTUs"
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "The full-threshold of the STQ, used by the fairness
         algorithm on this span.

         This attribute specifies the full-threshold location
         from the top of the STQ buffer, in MTU size.

         Default value for dual transit buffers MAC should be
         1, which means (STQ_SIZE - MTU_SIZE)"
    REFERENCE
        "IEEE 802.17 Subclause 9.2.43"
    ::= { rprFairnessEntry 9 }

rprFairnessHighThreshold OBJECT-TYPE
    SYNTAX        Integer32 (0..1000)
    UNITS         "Tenth of percent"
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "The high-threshold of the STQ, used by the fairness
         algorithm on this span.

         This attribute specifies the high-threshold location
         in percentage of the STQ size.

         Default value for dual transit buffers MAC that operates
         in fairness aggressive mode should be 25% (=250) of STQ_SIZE,
         Otherwise, should be 95% (=950) of STQ_SIZE."
    REFERENCE
        "IEEE 802.17 Subclause 9.2.44"
    ::= { rprFairnessEntry 10 }

rprFairnessLowThreshold OBJECT-TYPE
    SYNTAX        Integer32 (0..1000)
    UNITS         "Tenth of percent"
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "The low-threshold of the STQ, used by the fairness
         algorithm on this span.

         This attribute specifies the low-threshold location
         in percentage of the STQ size.

         Default value for dual transit buffers MAC the operates
         in fairness aggressive mode should be 12.5% (=125) of STQ_SIZE,
         Otherwise, should be 80% (=800) of STQ_SIZE."
    REFERENCE
        "IEEE 802.17 Subclause 9.2.45"
    ::= { rprFairnessEntry 11 }


--
-- The RPR OAM Echo table
--
```

```
rprOamEchoTable OBJECT-TYPE
    SYNTAX        SEQUENCE OF RprOamEchoEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "A table of RPR OAM Echo"
    ::= { rprProtocols 4 }

rprOamEchoEntry OBJECT-TYPE
    SYNTAX        RprOamEchoEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "OAM Echo control for an RPR interface"
    INDEX { rprOamEchoIfIndex }
    ::= { rprOamEchoTable 1 }

RprOamEchoEntry ::= SEQUENCE {
    rprOamEchoIfIndex           InterfaceIndex,
    rprOamEchoDestAddress       MacAddress,
    rprOamEchoRequestRinglet    RprOamEchoRinglet,
    rprOamEchoResponseRinglet   RprOamEchoRinglet,
    rprOamEchoCos               RprServiceClass,
    rprOamEchoProtection        RprProtectionControl,
    rprOamEchoReqCount          Integer32,
    rprOamEchoTimerTimeout      RprOamEchoTimeout,
    rprOamEchoControl           RprOamControl,
    rprOamEchoRespCount         Integer32,
    rprOamEchoAvRespTime        RprOamEchoTimeout,
    rprOamEchoRespStatus        RprOamEchoStatus
    }

rprOamEchoIfIndex  OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    ::= { rprOamEchoEntry 1 }

rprOamEchoRequestRinglet OBJECT-TYPE
    SYNTAX        RprOamEchoRinglet
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
       "An indication of the ringlet on which the echo request
        should be sent. The valid values for this object are:
        (1) ringlet 0,(2) ringlet 1 (3) default."
    REFERENCE
        "IEEE 802.17 Subclause 12.3.1.1"
    ::= { rprOamEchoEntry  2 }

rprOamEchoResponseRinglet OBJECT-TYPE
    SYNTAX        RprOamEchoRinglet
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
       "An indication of the ringlet on which the echo response
        should be replied. The valid values for this object are:
        (1) ringlet 0,(2) ringlet 1 (3) default."
    REFERENCE
        "IEEE 802.17 Subclause 12.3.1.2, 12.5.4.1.1"
    ::= { rprOamEchoEntry  3 }

rprOamEchoDestAddress OBJECT-TYPE
```

```
    SYNTAX        MacAddress
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "The 48-bit MAC address of the destination station of echo
         session."
    REFERENCE
        "IEEE 802.17 Subclause 12.3.1.1"
    ::= { rprOamEchoEntry  4 }

rprOamEchoCos OBJECT-TYPE
    SYNTAX        RprServiceClass
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "The class-of-service of echo session packets"
    REFERENCE
        "IEEE 802.17 Subclause 12.3.1.1, 12.5.1"
    DEFVAL { classA }
    ::= { rprOamEchoEntry 5 }

rprOamEchoProtection OBJECT-TYPE
    SYNTAX        RprProtectionControl
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Protection method to apply on the Echo process"
    REFERENCE
        "IEEE 802.17 Subclause 12.5.4.1.2"
    DEFVAL { protected }
    ::= { rprOamEchoEntry 6 }

rprOamEchoReqCount OBJECT-TYPE
    SYNTAX        Integer32 (0..65535)
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Indicates the number of echo requests to send."
    DEFVAL { 1 }
    ::= { rprOamEchoEntry 7 }

rprOamEchoTimerTimeout OBJECT-TYPE
    SYNTAX        RprOamEchoTimeout
    UNITS         "10usec"
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Indicates the timer to declare echo timeout, in
         10usec units."
    REFERENCE
        "IEEE 802.17 Subclause 12.3.1.2"
    DEFVAL { 20 }
    ::= { rprOamEchoEntry 8 }

rprOamEchoControl OBJECT-TYPE
    SYNTAX        RprOamControl
    MAX-ACCESS    read-write
    STATUS        current
    DESCRIPTION
        "Control of the OAM Echo process."
    DEFVAL { idle }
    ::= { rprOamEchoEntry 9 }

rprOamEchoRespCount OBJECT-TYPE
    SYNTAX        Integer32 (0..65535)
```

```
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "Indicates the number of echo responses received."
        ::= { rprOamEchoEntry 10 }

    rprOamEchoAvRespTime OBJECT-TYPE
        SYNTAX        RprOamEchoTimeout
        UNITS         "10usec"
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "Average response time to receive the echo reply."
        ::= { rprOamEchoEntry 11 }

    rprOamEchoRespStatus OBJECT-TYPE
        SYNTAX        RprOamEchoStatus
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "Status of the OAM Echo process."
        ::= { rprOamEchoEntry 12 }


    --
    -- 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
            "Packets and octets statistics for the current interval for
             the RPR MAC Span interface.
             The corresponding instance of rprIfTimeElapsed 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,

        rprSpanCurrentInUcastClassAPkts      Counter64,
        rprSpanCurrentInUcastClassAOctets    Counter64,
        rprSpanCurrentInUcastClassBCirPkts   Counter64,
        rprSpanCurrentInUcastClassBCirOctets Counter64,
        rprSpanCurrentInUcastClassBEirPkts   Counter64,
        rprSpanCurrentInUcastClassBEirOctets Counter64,
        rprSpanCurrentInUcastClassCPkts      Counter64,
        rprSpanCurrentInUcastClassCOctets    Counter64,

        rprSpanCurrentInMcastClassAPkts      Counter64,
        rprSpanCurrentInMcastClassAOctets    Counter64,
```

```
    rprSpanCurrentInMcastClassBCirPkts    Counter64,
    rprSpanCurrentInMcastClassBCirOctets  Counter64,
    rprSpanCurrentInMcastClassBEirPkts    Counter64,
    rprSpanCurrentInMcastClassBEirOctets  Counter64,
    rprSpanCurrentInMcastClassCPkts       Counter64,
    rprSpanCurrentInMcastClassCOctets     Counter64,

    rprSpanCurrentOutUcastClassAPkts      Counter64,
    rprSpanCurrentOutUcastClassAOctets    Counter64,
    rprSpanCurrentOutUcastClassBCirPkts   Counter64,
    rprSpanCurrentOutUcastClassBCirOctets Counter64,
    rprSpanCurrentOutUcastClassBEirPkts   Counter64,
    rprSpanCurrentOutUcastClassBEirOctets Counter64,
    rprSpanCurrentOutUcastClassCPkts      Counter64,
    rprSpanCurrentOutUcastClassCOctets    Counter64,

    rprSpanCurrentOutMcastClassAPkts      Counter64,
    rprSpanCurrentOutMcastClassAOctets    Counter64,
    rprSpanCurrentOutMcastClassBCirPkts   Counter64,
    rprSpanCurrentOutMcastClassBCirOctets Counter64,
    rprSpanCurrentOutMcastClassBEirPkts   Counter64,
    rprSpanCurrentOutMcastClassBEirOctets Counter64,
    rprSpanCurrentOutMcastClassCPkts      Counter64,
    rprSpanCurrentOutMcastClassCOctets    Counter64
    }

rprSpanCurrentIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        “The ifIndex of this RPR interface.”
    ::= { 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 }

rprSpanCurrentInUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received Class A unicast packets in the
         current interval.”
    ::= { rprSpanCountersCurrentEntry 3 }

rprSpanCurrentInUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received Class A unicast octets in the
         current interval.”
    ::= { rprSpanCountersCurrentEntry 4 }

rprSpanCurrentInUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
```

```
    DESCRIPTION
        "The number of received in profile Class B unicast packets
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 5 }

rprSpanCurrentInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B unicast octets
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 6 }

rprSpanCurrentInUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B unicast
         packets in the current interval."
    ::= { rprSpanCountersCurrentEntry 7 }

rprSpanCurrentInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B unicast
         octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 8 }

rprSpanCurrentInUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class C unicast packets in the
         current interval."
    ::= { rprSpanCountersCurrentEntry 9 }

rprSpanCurrentInUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class C unicast octets in the
         current interval."
    ::= { rprSpanCountersCurrentEntry 10 }

rprSpanCurrentInMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class A multicast and broadcast
         packets in the current interval."
    ::= { rprSpanCountersCurrentEntry 11 }

rprSpanCurrentInMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class A multicast and broadcast
```

```
         octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 12 }

rprSpanCurrentInMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B
         multicast and broadcast packets in the current interval."
    ::= { rprSpanCountersCurrentEntry 13 }

rprSpanCurrentInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B
         multicast and broadcast octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 14 }

rprSpanCurrentInMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B
         multicast and broadcast packets in the current interval."
    ::= { rprSpanCountersCurrentEntry 15 }

rprSpanCurrentInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B
         multicast and broadcast octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 16 }

rprSpanCurrentInMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class C multicast and broadcast
         packets in the current interval."
    ::= { rprSpanCountersCurrentEntry 17 }

rprSpanCurrentInMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class C multicast and broadcast
         octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 18 }

rprSpanCurrentOutUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A unicast packets
         in the current interval."
    ::= { rprSpanCountersCurrentEntry 19 }
```

```
rprSpanCurrentOutUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted Class A unicast octets
         in the current interval.”
    ::= { rprSpanCountersCurrentEntry 20 }

rprSpanCurrentOutUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted in profile Class B unicast
         packets in the current interval.”
    ::= { rprSpanCountersCurrentEntry 21 }

rprSpanCurrentOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted in profile Class B unicast
         octets in the current interval.”
    ::= { rprSpanCountersCurrentEntry 22 }

rprSpanCurrentOutUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted out of profile Class B unicast
         packets in the current interval.”
    ::= { rprSpanCountersCurrentEntry 23 }

rprSpanCurrentOutUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted out of profile Class B unicast
         octets in the current interval.”
    ::= { rprSpanCountersCurrentEntry 24 }

rprSpanCurrentOutUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted Class C unicast packets
         in the current interval.”
    ::= { rprSpanCountersCurrentEntry 25 }

rprSpanCurrentOutUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted Class C unicast octets
         in the current interval.”
    ::= { rprSpanCountersCurrentEntry 26 }

rprSpanCurrentOutMcastClassAPkts OBJECT-TYPE
```

```
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A multicast and broadcast
         packets in the current interval."
    ::= { rprSpanCountersCurrentEntry 27 }

rprSpanCurrentOutMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A multicast and broadcast
         octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 28 }

rprSpanCurrentOutMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B
         multicast and broadcast packets in the current interval."
    ::= { rprSpanCountersCurrentEntry 29 }

rprSpanCurrentOutMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B
         multicast and broadcast octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 30 }

rprSpanCurrentOutMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted out of profile Class B
         multicast and broadcast packets in the current interval."
    ::= { rprSpanCountersCurrentEntry 31 }

rprSpanCurrentOutMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted out of profile Class B
         multicast and broadcast octets in the current interval."
    ::= { rprSpanCountersCurrentEntry 32 }

rprSpanCurrentOutMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C multicast and broadcast
         packets in the current interval."
    ::= { rprSpanCountersCurrentEntry 33 }

rprSpanCurrentOutMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
```

```
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C multicast and 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
        "Packets 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                   Integer32,
    rprSpanIntervalValidData                TruthValue,
    rprSpanIntervalTimeElapsed              Integer32,

    rprSpanIntervalInUcastClassAPkts        Counter64,
    rprSpanIntervalInUcastClassAOctets      Counter64,
    rprSpanIntervalInUcastClassBCirPkts     Counter64,
    rprSpanIntervalInUcastClassBCirOctets   Counter64,
    rprSpanIntervalInUcastClassBEirPkts     Counter64,
    rprSpanIntervalInUcastClassBEirOctets   Counter64,
    rprSpanIntervalInUcastClassCPkts        Counter64,
    rprSpanIntervalInUcastClassCOctets      Counter64,

    rprSpanIntervalInMcastClassAPkts        Counter64,
    rprSpanIntervalInMcastClassAOctets      Counter64,
    rprSpanIntervalInMcastClassBCirPkts     Counter64,
    rprSpanIntervalInMcastClassBCirOctets   Counter64,
    rprSpanIntervalInMcastClassBEirPkts     Counter64,
    rprSpanIntervalInMcastClassBEirOctets   Counter64,
    rprSpanIntervalInMcastClassCPkts        Counter64,
    rprSpanIntervalInMcastClassCOctets      Counter64,

    rprSpanIntervalOutUcastClassAPkts       Counter64,
    rprSpanIntervalOutUcastClassAOctets     Counter64,
    rprSpanIntervalOutUcastClassBCirPkts    Counter64,
    rprSpanIntervalOutUcastClassBCirOctets  Counter64,
    rprSpanIntervalOutUcastClassBEirPkts    Counter64,
    rprSpanIntervalOutUcastClassBEirOctets  Counter64,
```

```
    rprSpanIntervalOutUcastClassCPkts       Counter64,
    rprSpanIntervalOutUcastClassCOctets     Counter64,

    rprSpanIntervalOutMcastClassAPkts       Counter64,
    rprSpanIntervalOutMcastClassAOctets     Counter64,
    rprSpanIntervalOutMcastClassBCirPkts    Counter64,
    rprSpanIntervalOutMcastClassBCirOctets  Counter64,
    rprSpanIntervalOutMcastClassBEirPkts    Counter64,
    rprSpanIntervalOutMcastClassBEirOctets  Counter64,
    rprSpanIntervalOutMcastClassCPkts       Counter64,
    rprSpanIntervalOutMcastClassCOctets     Counter64
    }

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        Integer32 (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.”
    ::= { rprSpanCountersIntervalEntry 4 }

rprSpanIntervalTimeElapsed OBJECT-TYPE
    SYNTAX        Integer32
    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 }

rprSpanIntervalInUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
```

```
        STATUS        current
        DESCRIPTION
            "The number of received Class A unicast packets
             in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 6 }

    rprSpanIntervalInUcastClassAOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received Class A unicast octets
             in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 7 }

    rprSpanIntervalInUcastClassBCirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received in profile Class B unicast packets
             in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 8 }

    rprSpanIntervalInUcastClassBCirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received in profile Class B unicast octets
             in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 9 }

    rprSpanIntervalInUcastClassBEirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received out of profile Class B unicast
             packets in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 10 }

    rprSpanIntervalInUcastClassBEirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received out of profile Class B unicast
             octets in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 11 }

    rprSpanIntervalInUcastClassCPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received Class C unicast
             packets in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 12 }

    rprSpanIntervalInUcastClassCOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
```

```
        "The number of received Class C unicast
         octets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 13 }

rprSpanIntervalInMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class A multicast and broadcast
         packets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 14 }

rprSpanIntervalInMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class A multicast and broadcast
         octets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 15 }

rprSpanIntervalInMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B multicast and
         broadcast packets in a particular interval in the past
         24 hours."
    ::= { rprSpanCountersIntervalEntry 16 }

rprSpanIntervalInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B multicast and
         broadcast octets in a particular interval in the past
         24 hours."
    ::= { rprSpanCountersIntervalEntry 17 }

rprSpanIntervalInMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B multicast and
         broadcast packets in a particular interval in the past
         24 hours."
    ::= { rprSpanCountersIntervalEntry 18 }

rprSpanIntervalInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B multicast and
         broadcast octets in a particular interval in the past
         24 hours."
    ::= { rprSpanCountersIntervalEntry 19 }

rprSpanIntervalInMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
```

```
        STATUS        current
        DESCRIPTION
            "The number of received Class C multicast and broadcast
             packets in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 20 }

    rprSpanIntervalInMcastClassCOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received Class C multicast and broadcast
             octets in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 21 }

    rprSpanIntervalOutUcastClassAPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of transmitted Class A unicast packets
             in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 22 }

    rprSpanIntervalOutUcastClassAOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of transmitted Class A unicast octets
             in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 23 }

    rprSpanIntervalOutUcastClassBCirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of transmitted in profile Class B unicast packets
             in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 24 }

    rprSpanIntervalOutUcastClassBCirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of transmitted in profile Class B unicast octets
             in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 25 }

    rprSpanIntervalOutUcastClassBEirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of transmitted out of profile Class B unicast
             packets in a particular interval in the past 24 hours."
        ::= { rprSpanCountersIntervalEntry 26 }

    rprSpanIntervalOutUcastClassBEirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
```

```
        "The number of transmitted out of profile Class B unicast
         octets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 27 }

rprSpanIntervalOutUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C unicast
         packets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 28 }

rprSpanIntervalOutUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C unicast
         octets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 29 }

rprSpanIntervalOutMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A multicast and broadcast
         packets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 30 }

rprSpanIntervalOutMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A multicast and broadcast
         octets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 31 }

rprSpanIntervalOutMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B multicast
         and broadcast packets in a particular interval in the
         past 24 hours."
    ::= { rprSpanCountersIntervalEntry 32 }

rprSpanIntervalOutMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B multicast
         and broadcast octets in a particular interval in the
         past 24 hours."
    ::= { rprSpanCountersIntervalEntry 33 }

rprSpanIntervalOutMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
```

```
        "The number of transmitted out of profile Class B multicast
         and broadcast packets in a particular interval in the
         past 24 hours."
    ::= { rprSpanCountersIntervalEntry 34 }

rprSpanIntervalOutMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted out of profile Class B multicast
         and broadcast octets in a particular interval in the
         past 24 hours."
    ::= { rprSpanCountersIntervalEntry 35 }

rprSpanIntervalOutMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C multicast and broadcast
         packets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 36 }

rprSpanIntervalOutMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C multicast and broadcast
         octets in a particular interval in the past 24 hours."
    ::= { rprSpanCountersIntervalEntry 37 }

--
-- 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,
    rprSpanDayTimeElapsed            Integer32,

    rprSpanDayInUcastClassAPkts      Counter64,
    rprSpanDayInUcastClassAOctets    Counter64,
    rprSpanDayInUcastClassBCirPkts   Counter64,
```

```
    rprSpanDayInUcastClassBCirOctets  Counter64,
    rprSpanDayInUcastClassBEirPkts    Counter64,
    rprSpanDayInUcastClassBEirOctets  Counter64,
    rprSpanDayInUcastClassCPkts       Counter64,
    rprSpanDayInUcastClassCOctets     Counter64,

    rprSpanDayInMcastClassAPkts       Counter64,
    rprSpanDayInMcastClassAOctets     Counter64,
    rprSpanDayInMcastClassBCirPkts    Counter64,
    rprSpanDayInMcastClassBCirOctets  Counter64,
    rprSpanDayInMcastClassBEirPkts    Counter64,
    rprSpanDayInMcastClassBEirOctets  Counter64,
    rprSpanDayInMcastClassCPkts       Counter64,
    rprSpanDayInMcastClassCOctets     Counter64,

    rprSpanDayOutUcastClassAPkts      Counter64,
    rprSpanDayOutUcastClassAOctets    Counter64,
    rprSpanDayOutUcastClassBCirPkts   Counter64,
    rprSpanDayOutUcastClassBCirOctets Counter64,
    rprSpanDayOutUcastClassBEirPkts   Counter64,
    rprSpanDayOutUcastClassBEirOctets Counter64,
    rprSpanDayOutUcastClassCPkts      Counter64,
    rprSpanDayOutUcastClassCOctets    Counter64,

    rprSpanDayOutMcastClassAPkts      Counter64,
    rprSpanDayOutMcastClassAOctets    Counter64,
    rprSpanDayOutMcastClassBCirPkts   Counter64,
    rprSpanDayOutMcastClassBCirOctets Counter64,
    rprSpanDayOutMcastClassBEirPkts   Counter64,
    rprSpanDayOutMcastClassBEirOctets Counter64,
    rprSpanDayOutMcastClassCPkts      Counter64,
    rprSpanDayOutMcastClassCOctets    Counter64
    }

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 }

rprSpanDayTimeElapsed OBJECT-TYPE
    SYNTAX        Integer32
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      “The duration of the Day counters in seconds.
       Non-valid intervals will not be added to the Day counters.
       If, for some reason, such as an adjustment in the system’s
       time-of-day clock, the Day interval exceeds the maximum
       value, the agent will return the maximum value.”
    ::= { rprSpanCountersDayEntry 3 }

rprSpanDayInUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
```

```
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A unicast packets."
    ::= { rprSpanCountersDayEntry 4 }

rprSpanDayInUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A unicast octets."
    ::= { rprSpanCountersDayEntry 5 }

rprSpanDayInUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B unicast packets."
    ::= { rprSpanCountersDayEntry 6 }

rprSpanDayInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B unicast octets."
    ::= { rprSpanCountersDayEntry 7 }

rprSpanDayInUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B unicast packets."
    ::= { rprSpanCountersDayEntry 8 }

rprSpanDayInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B unicast octets."
    ::= { rprSpanCountersDayEntry 9 }

rprSpanDayInUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C unicast packets."
    ::= { rprSpanCountersDayEntry 10 }

rprSpanDayInUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C unicast octets."
    ::= { rprSpanCountersDayEntry 11 }

rprSpanDayInMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
```

```
    STATUS        current
    DESCRIPTION
      "The number of received Class A multicast and broadcast packets"
    ::= { rprSpanCountersDayEntry 12 }

rprSpanDayInMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A multicast and broadcast octets"
    ::= { rprSpanCountersDayEntry 13 }

rprSpanDayInMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B multicast
       and broadcast packets"
    ::= { rprSpanCountersDayEntry 14 }

rprSpanDayInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B multicast
       and broadcast octets"
    ::= { rprSpanCountersDayEntry 15 }

rprSpanDayInMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B multicast
       and broadcast packets"
    ::= { rprSpanCountersDayEntry 16 }

rprSpanDayInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B multicast
       and broadcast octets"
    ::= { rprSpanCountersDayEntry 17 }

rprSpanDayInMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C multicast and broadcast packets"
    ::= { rprSpanCountersDayEntry 18 }

rprSpanDayInMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C multicast and broadcast octets"
    ::= { rprSpanCountersDayEntry 19 }
```

```
rprSpanDayOutUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A unicast packets."
    ::= { rprSpanCountersDayEntry 20 }

rprSpanDayOutUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A unicast octets."
    ::= { rprSpanCountersDayEntry 21 }

rprSpanDayOutUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B unicast packets."
    ::= { rprSpanCountersDayEntry 22 }

rprSpanDayOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B unicast octets."
    ::= { rprSpanCountersDayEntry 23 }

rprSpanDayOutUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B unicast
       packets"
    ::= { rprSpanCountersDayEntry 24 }

rprSpanDayOutUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B unicast
       octets"
    ::= { rprSpanCountersDayEntry 25 }

rprSpanDayOutUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C unicast packets"
    ::= { rprSpanCountersDayEntry 26 }

rprSpanDayOutUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C unicast octets"
    ::= { rprSpanCountersDayEntry 27 }
```

```
rprSpanDayOutMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A multicast and broadcast
       packets."
    ::= { rprSpanCountersDayEntry 28 }

rprSpanDayOutMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A multicast and broadcast
       octets."
    ::= { rprSpanCountersDayEntry 29 }

rprSpanDayOutMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B
       multicast and broadcast packets."
    ::= { rprSpanCountersDayEntry 30 }

rprSpanDayOutMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B
       multicast and broadcast octets."
    ::= { rprSpanCountersDayEntry 31 }

rprSpanDayOutMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B
       multicast and broadcast packets."
    ::= { rprSpanCountersDayEntry 32 }

rprSpanDayOutMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B
       multicast and broadcast octets."
    ::= { rprSpanCountersDayEntry 33 }

rprSpanDayOutMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C multicast and broadcast
       packets."
    ::= { rprSpanCountersDayEntry 34 }

rprSpanDayOutMcastClassCOctets OBJECT-TYPE
```

```
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class C multicast and broadcast
           octets."
        ::= { rprSpanCountersDayEntry 35 }


    --
    -- 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."
        ::= { rprSpanCounters 4 }

    rprSpanCountersStatsEntry OBJECT-TYPE
        SYNTAX        RprSpanCountersStatsEntry
        MAX-ACCESS    not-accessible
        STATUS        current
        DESCRIPTION
          "An entry in the span stats table.

           The DiscontinuityTime for this table is defined in the
           rprSpanTable."
        INDEX { rprSpanStatsIfIndex,
                rprSpanStatsSpan }
        ::= { rprSpanCountersStatsTable 1 }

    RprSpanCountersStatsEntry ::= SEQUENCE {
        rprSpanStatsIfIndex                InterfaceIndex,
        rprSpanStatsSpan                   RprSpan,

        rprSpanStatsInUcastClassAPkts      Counter64,
        rprSpanStatsInUcastClassAOctets    Counter64,
        rprSpanStatsInUcastClassBCirPkts   Counter64,
        rprSpanStatsInUcastClassBCirOctets Counter64,
        rprSpanStatsInUcastClassBEirPkts   Counter64,
        rprSpanStatsInUcastClassBEirOctets Counter64,
        rprSpanStatsInUcastClassCPkts      Counter64,
        rprSpanStatsInUcastClassCOctets    Counter64,

        rprSpanStatsInMcastClassAPkts      Counter64,
        rprSpanStatsInMcastClassAOctets    Counter64,
        rprSpanStatsInMcastClassBCirPkts   Counter64,
        rprSpanStatsInMcastClassBCirOctets Counter64,
        rprSpanStatsInMcastClassBEirPkts   Counter64,
        rprSpanStatsInMcastClassBEirOctets Counter64,
        rprSpanStatsInMcastClassCPkts      Counter64,
        rprSpanStatsInMcastClassCOctets    Counter64,

        rprSpanStatsOutUcastClassAPkts     Counter64,
        rprSpanStatsOutUcastClassAOctets   Counter64,
        rprSpanStatsOutUcastClassBCirPkts  Counter64,
        rprSpanStatsOutUcastClassBCirOctets Counter64,
        rprSpanStatsOutUcastClassBEirPkts  Counter64,
        rprSpanStatsOutUcastClassBEirOctets Counter64,
        rprSpanStatsOutUcastClassCPkts     Counter64,
        rprSpanStatsOutUcastClassCOctets   Counter64,
```

```
    rprSpanStatsOutMcastClassAPkts      Counter64,
    rprSpanStatsOutMcastClassAOctets    Counter64,
    rprSpanStatsOutMcastClassBCirPkts   Counter64,
    rprSpanStatsOutMcastClassBCirOctets Counter64,
    rprSpanStatsOutMcastClassBEirPkts   Counter64,
    rprSpanStatsOutMcastClassBEirOctets Counter64,
    rprSpanStatsOutMcastClassCPkts      Counter64,
    rprSpanStatsOutMcastClassCOctets    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."
    ::= { rprSpanCountersStatsEntry 2 }

rprSpanStatsInUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A unicast packets."
    ::= { rprSpanCountersStatsEntry 3 }

rprSpanStatsInUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A unicast octets."
    ::= { rprSpanCountersStatsEntry 4 }

rprSpanStatsInUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B unicast packets."
    ::= { rprSpanCountersStatsEntry 5 }

rprSpanStatsInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B unicast octets."
    ::= { rprSpanCountersStatsEntry 6 }

rprSpanStatsInUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B unicast packets."
```

```
    ::= { rprSpanCountersStatsEntry 7 }

rprSpanStatsInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B unicast octets."
    ::= { rprSpanCountersStatsEntry 8 }

rprSpanStatsInUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C unicast packets."
    ::= { rprSpanCountersStatsEntry 9 }

rprSpanStatsInUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C unicast octets."
    ::= { rprSpanCountersStatsEntry 10 }

rprSpanStatsInMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A multicast and broadcast packets"
    ::= { rprSpanCountersStatsEntry 11 }

rprSpanStatsInMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A multicast and broadcast octets"
    ::= { rprSpanCountersStatsEntry 12 }

rprSpanStatsInMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B multicast
       and broadcast packets"
    ::= { rprSpanCountersStatsEntry 13 }

rprSpanStatsInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B multicast
       and broadcast octets"
    ::= { rprSpanCountersStatsEntry 14 }

rprSpanStatsInMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
```

```
      "The number of received out of profile Class B multicast
       and broadcast packets"
    ::= { rprSpanCountersStatsEntry 15 }

rprSpanStatsInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B multicast
       and broadcast octets"
    ::= { rprSpanCountersStatsEntry 16 }

rprSpanStatsInMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C multicast and broadcast packets"
    ::= { rprSpanCountersStatsEntry 17 }

rprSpanStatsInMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C multicast and broadcast octets"
    ::= { rprSpanCountersStatsEntry 18 }

rprSpanStatsOutUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A unicast packets."
    ::= { rprSpanCountersStatsEntry 19 }

rprSpanStatsOutUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A unicast octets."
    ::= { rprSpanCountersStatsEntry 20 }

rprSpanStatsOutUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B unicast packets."
    ::= { rprSpanCountersStatsEntry 21 }

rprSpanStatsOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B unicast octets."
    ::= { rprSpanCountersStatsEntry 22 }

rprSpanStatsOutUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
```

```
        DESCRIPTION
          "The number of transmitted out of profile Class B unicast
           packets"
        ::= { rprSpanCountersStatsEntry 23 }

    rprSpanStatsOutUcastClassBEirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted out of profile Class B unicast
           octets"
        ::= { rprSpanCountersStatsEntry 24 }

    rprSpanStatsOutUcastClassCPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class C unicast packets"
        ::= { rprSpanCountersStatsEntry 25 }

    rprSpanStatsOutUcastClassCOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class C unicast octets"
        ::= { rprSpanCountersStatsEntry 26 }

    rprSpanStatsOutMcastClassAPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class A multicast and broadcast
           packets."
        ::= { rprSpanCountersStatsEntry 27 }

    rprSpanStatsOutMcastClassAOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class A multicast and broadcast
           octets."
        ::= { rprSpanCountersStatsEntry 28 }

    rprSpanStatsOutMcastClassBCirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted in profile Class B
           multicast and broadcast packets."
        ::= { rprSpanCountersStatsEntry 29 }

    rprSpanStatsOutMcastClassBCirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted in profile Class B
           multicast and broadcast octets."
        ::= { rprSpanCountersStatsEntry 30 }
```

```
rprSpanStatsOutMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B
       multicast and broadcast packets."
    ::= { rprSpanCountersStatsEntry 31 }

rprSpanStatsOutMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B
       multicast and broadcast octets."
    ::= { rprSpanCountersStatsEntry 32 }

rprSpanStatsOutMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C multicast and broadcast
       packets."
    ::= { rprSpanCountersStatsEntry 33 }

rprSpanStatsOutMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C multicast and broadcast
       octets."
    ::= { rprSpanCountersStatsEntry 34 }

--
-- 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
       "Packets 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,
```

```
        rprClientCurrentInUcastClassAPkts       Counter64,
        rprClientCurrentInUcastClassAOctets     Counter64,
        rprClientCurrentInUcastClassBCirPkts    Counter64,
        rprClientCurrentInUcastClassBCirOctets  Counter64,
        rprClientCurrentInUcastClassBEirPkts    Counter64,
        rprClientCurrentInUcastClassBEirOctets  Counter64,
        rprClientCurrentInUcastClassCPkts       Counter64,
        rprClientCurrentInUcastClassCOctets     Counter64,

        rprClientCurrentInMcastClassAPkts       Counter64,
        rprClientCurrentInMcastClassAOctets     Counter64,
        rprClientCurrentInMcastClassBCirPkts    Counter64,
        rprClientCurrentInMcastClassBCirOctets  Counter64,
        rprClientCurrentInMcastClassBEirPkts    Counter64,
        rprClientCurrentInMcastClassBEirOctets  Counter64,
        rprClientCurrentInMcastClassCPkts       Counter64,
        rprClientCurrentInMcastClassCOctets     Counter64,

        rprClientCurrentOutUcastClassAPkts      Counter64,
        rprClientCurrentOutUcastClassAOctets    Counter64,
        rprClientCurrentOutUcastClassBCirPkts   Counter64,
        rprClientCurrentOutUcastClassBCirOctets Counter64,
        rprClientCurrentOutUcastClassBEirPkts   Counter64,
        rprClientCurrentOutUcastClassBEirOctets Counter64,
        rprClientCurrentOutUcastClassCPkts      Counter64,
        rprClientCurrentOutUcastClassCOctets    Counter64,

        rprClientCurrentOutMcastClassAPkts      Counter64,
        rprClientCurrentOutMcastClassAOctets    Counter64,
        rprClientCurrentOutMcastClassBCirPkts   Counter64,
        rprClientCurrentOutMcastClassBCirOctets Counter64,
        rprClientCurrentOutMcastClassBEirPkts   Counter64,
        rprClientCurrentOutMcastClassBEirOctets Counter64,
        rprClientCurrentOutMcastClassCPkts      Counter64,
        rprClientCurrentOutMcastClassCOctets    Counter64
        }

    rprClientCurrentIfIndex OBJECT-TYPE
        SYNTAX        InterfaceIndex
        MAX-ACCESS    not-accessible
        STATUS        current
        DESCRIPTION
            “The ifIndex of this RPR interface.”
        ::= { rprClientCountersCurrentEntry 1 }

    rprClientCurrentInUcastClassAPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            “The number of received Class A unicast packets in
             the current interval.”
        ::= { rprClientCountersCurrentEntry 2 }

    rprClientCurrentInUcastClassAOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            “The number of received Class A unicast octets in
             the current interval.”
        ::= { rprClientCountersCurrentEntry 3 }

    rprClientCurrentInUcastClassBCirPkts OBJECT-TYPE
        SYNTAX        Counter64
```

```
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B unicast packets in
         the current interval."
    ::= { rprClientCountersCurrentEntry 4 }

rprClientCurrentInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B unicast octets
         in the current interval."
    ::= { rprClientCountersCurrentEntry 5 }

rprClientCurrentInUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B unicast
         packets in the current interval."
    ::= { rprClientCountersCurrentEntry 6 }

rprClientCurrentInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B unicast
         octets in the current interval."
    ::= { rprClientCountersCurrentEntry 7 }

rprClientCurrentInUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class C unicast packets in the
         current interval."
    ::= { rprClientCountersCurrentEntry 8 }

rprClientCurrentInUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class C unicast octets in the
         current interval."
    ::= { rprClientCountersCurrentEntry 9 }

rprClientCurrentInMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class A multicast and broadcast
         packets in the current interval."
    ::= { rprClientCountersCurrentEntry 10 }

rprClientCurrentInMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
```

```
        DESCRIPTION
            "The number of received Class A multicast and broadcast
             octets in the current interval."
        ::= { rprClientCountersCurrentEntry 11 }

    rprClientCurrentInMcastClassBCirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received in profile Class B
             multicast and broadcast packets in the current interval."
        ::= { rprClientCountersCurrentEntry 12 }

    rprClientCurrentInMcastClassBCirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received in profile Class B
             multicast and broadcast octets in the current interval."
        ::= { rprClientCountersCurrentEntry 13 }

    rprClientCurrentInMcastClassBEirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received out of profile Class B
             multicast and broadcast packets in the current interval."
        ::= { rprClientCountersCurrentEntry 14 }

    rprClientCurrentInMcastClassBEirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received out of profile Class B
             multicast and broadcast octets in the current interval."
        ::= { rprClientCountersCurrentEntry 15 }

    rprClientCurrentInMcastClassCPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received Class C multicast and broadcast
             packets in the current interval."
        ::= { rprClientCountersCurrentEntry 16 }

    rprClientCurrentInMcastClassCOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received Class C multicast and broadcast
             octets in the current interval."
        ::= { rprClientCountersCurrentEntry 17 }

    rprClientCurrentOutUcastClassAPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of transmitted Class A unicast packets
```

```
         in the current interval."
    ::= { rprClientCountersCurrentEntry 18 }

rprClientCurrentOutUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A unicast octets
         in the current interval."
    ::= { rprClientCountersCurrentEntry 19 }

rprClientCurrentOutUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B unicast
         packets in the current interval."
    ::= { rprClientCountersCurrentEntry 20 }

rprClientCurrentOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B unicast
         octets in the current interval."
    ::= { rprClientCountersCurrentEntry 21 }

rprClientCurrentOutUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted out of profile Class B unicast
         packets in the current interval."
    ::= { rprClientCountersCurrentEntry 22 }

rprClientCurrentOutUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted out of profile Class B unicast
         octets in the current interval."
    ::= { rprClientCountersCurrentEntry 23 }

rprClientCurrentOutUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C unicast packets
         in the current interval."
    ::= { rprClientCountersCurrentEntry 24 }

rprClientCurrentOutUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C unicast octets
         in the current interval."
    ::= { rprClientCountersCurrentEntry 25 }
```

```
rprClientCurrentOutMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted Class A multicast and broadcast
         packets in the current interval.”
    ::= { rprClientCountersCurrentEntry 26 }

rprClientCurrentOutMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted Class A multicast and broadcast
         octets in the current interval.”
    ::= { rprClientCountersCurrentEntry 27 }

rprClientCurrentOutMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted in profile Class B
         multicast and broadcast packets in the current interval.”
    ::= { rprClientCountersCurrentEntry 28 }

rprClientCurrentOutMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted in profile Class B
         multicast and broadcast octets in the current interval.”
    ::= { rprClientCountersCurrentEntry 29 }

rprClientCurrentOutMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted out of profile Class B
         multicast and broadcast packets in the current interval.”
    ::= { rprClientCountersCurrentEntry 30 }

rprClientCurrentOutMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted out of profile Class B
         multicast and broadcast octets in the current interval.”
    ::= { rprClientCountersCurrentEntry 31 }

rprClientCurrentOutMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted Class C multicast and broadcast
         packets in the current interval.”
    ::= { rprClientCountersCurrentEntry 32 }

rprClientCurrentOutMcastClassCOctets OBJECT-TYPE
```

```
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C 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
      "Packets 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                 Integer32,
    rprClientIntervalValidData              TruthValue,
    rprClientIntervalTimeElapsed            Integer32,

    rprClientIntervalInUcastClassAPkts      Counter64,
    rprClientIntervalInUcastClassAOctets    Counter64,
    rprClientIntervalInUcastClassBCirPkts   Counter64,
    rprClientIntervalInUcastClassBCirOctets Counter64,
    rprClientIntervalInUcastClassBEirPkts   Counter64,
    rprClientIntervalInUcastClassBEirOctets Counter64,
    rprClientIntervalInUcastClassCPkts      Counter64,
    rprClientIntervalInUcastClassCOctets    Counter64,

    rprClientIntervalInMcastClassAPkts      Counter64,
    rprClientIntervalInMcastClassAOctets    Counter64,
    rprClientIntervalInMcastClassBCirPkts   Counter64,
    rprClientIntervalInMcastClassBCirOctets Counter64,
    rprClientIntervalInMcastClassBEirPkts   Counter64,
    rprClientIntervalInMcastClassBEirOctets Counter64,
    rprClientIntervalInMcastClassCPkts      Counter64,
    rprClientIntervalInMcastClassCOctets    Counter64,

    rprClientIntervalOutUcastClassAPkts     Counter64,
    rprClientIntervalOutUcastClassAOctets   Counter64,
    rprClientIntervalOutUcastClassBCirPkts  Counter64,
    rprClientIntervalOutUcastClassBCirOctets Counter64,
    rprClientIntervalOutUcastClassBEirPkts  Counter64,
    rprClientIntervalOutUcastClassBEirOctets Counter64,
```

```
    rprClientIntervalOutUcastClassCPkts      Counter64,
    rprClientIntervalOutUcastClassCOctets    Counter64,

    rprClientIntervalOutMcastClassAPkts      Counter64,
    rprClientIntervalOutMcastClassAOctets    Counter64,
    rprClientIntervalOutMcastClassBCirPkts   Counter64,
    rprClientIntervalOutMcastClassBCirOctets Counter64,
    rprClientIntervalOutMcastClassBEirPkts   Counter64,
    rprClientIntervalOutMcastClassBEirOctets Counter64,
    rprClientIntervalOutMcastClassCPkts      Counter64,
    rprClientIntervalOutMcastClassCOctets    Counter64
    }

rprClientIntervalIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    ::= { rprClientCountersIntervalEntry 1 }

rprClientIntervalNumber OBJECT-TYPE
    SYNTAX        Integer32 (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."
    ::= { rprClientCountersIntervalEntry 3 }

rprClientIntervalTimeElapsed OBJECT-TYPE
    SYNTAX        Integer32
    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 }

rprClientIntervalInUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class A unicast packets
         in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 5 }

rprClientIntervalInUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
```

```
    STATUS        current
    DESCRIPTION
        "The number of received Class A unicast octets
         in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 6 }

rprClientIntervalInUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B unicast packets
         in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 7 }

rprClientIntervalInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B unicast octets
         in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 8 }

rprClientIntervalInUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B unicast
         packets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 9 }

rprClientIntervalInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B unicast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 10 }

rprClientIntervalInUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class C unicast
         packets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 11 }

rprClientIntervalInUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class C unicast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 12 }

rprClientIntervalInMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
```

```
        "The number of received Class A multicast and broadcast
         packets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 13 }

rprClientIntervalInMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class A multicast and broadcast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 14 }

rprClientIntervalInMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B multicast and
         broadcast packets in a particular interval in the past
         24 hours."
    ::= { rprClientCountersIntervalEntry 15 }

rprClientIntervalInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received in profile Class B multicast and
         broadcast octets in a particular interval in the past
         24 hours."
    ::= { rprClientCountersIntervalEntry 16 }

rprClientIntervalInMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B multicast and
         broadcast packets in a particular interval in the past
         24 hours."
    ::= { rprClientCountersIntervalEntry 17 }

rprClientIntervalInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received out of profile Class B multicast and
         broadcast octets in a particular interval in the past
         24 hours."
    ::= { rprClientCountersIntervalEntry 18 }

rprClientIntervalInMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received Class C multicast and broadcast
         packets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 19 }

rprClientIntervalInMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
```

```
    STATUS        current
    DESCRIPTION
        "The number of received Class C multicast and broadcast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 20 }

rprClientIntervalOutUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A unicast packets
         in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 21 }

rprClientIntervalOutUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A unicast octets
         in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 22 }

rprClientIntervalOutUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B unicast packets
         in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 23 }

rprClientIntervalOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B unicast octets
         in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 24 }

rprClientIntervalOutUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted out of profile Class B unicast
         packets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 25 }

rprClientIntervalOutUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted out of profile Class B unicast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 26 }

rprClientIntervalOutUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
```

```
        "The number of transmitted Class C unicast
         packets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 27 }

rprClientIntervalOutUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class C unicast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 28 }

rprClientIntervalOutMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A multicast and broadcast
         packets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 29 }

rprClientIntervalOutMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted Class A multicast and broadcast
         octets in a particular interval in the past 24 hours."
    ::= { rprClientCountersIntervalEntry 30 }

rprClientIntervalOutMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B multicast
         and broadcast packets in a particular interval in the
         past 24 hours."
    ::= { rprClientCountersIntervalEntry 31 }

rprClientIntervalOutMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted in profile Class B multicast
         and broadcast octets in a particular interval in the
         past 24 hours."
    ::= { rprClientCountersIntervalEntry 32 }

rprClientIntervalOutMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of transmitted out of profile Class B multicast
         and broadcast packets in a particular interval in the
         past 24 hours."
    ::= { rprClientCountersIntervalEntry 33 }

rprClientIntervalOutMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
```

```
    DESCRIPTION
        “The number of transmitted out of profile Class B multicast
         and broadcast octets in a particular interval in the
         past 24 hours.”
    ::= { rprClientCountersIntervalEntry 34 }

rprClientIntervalOutMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted Class C multicast and broadcast
         packets in a particular interval in the past 24 hours.”
    ::= { rprClientCountersIntervalEntry 35 }

rprClientIntervalOutMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of transmitted Class C 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,
    rprClientDayTimeElapsed             Integer32,

    rprClientDayInUcastClassAPkts       Counter64,
    rprClientDayInUcastClassAOctets     Counter64,
    rprClientDayInUcastClassBCirPkts    Counter64,
    rprClientDayInUcastClassBCirOctets  Counter64,
    rprClientDayInUcastClassBEirPkts    Counter64,
    rprClientDayInUcastClassBEirOctets  Counter64,
    rprClientDayInUcastClassCPkts       Counter64,
    rprClientDayInUcastClassCOctets     Counter64,

    rprClientDayInMcastClassAPkts       Counter64,
    rprClientDayInMcastClassAOctets     Counter64,
    rprClientDayInMcastClassBCirPkts    Counter64,
    rprClientDayInMcastClassBCirOctets  Counter64,
```

```
    rprClientDayInMcastClassBEirPkts     Counter64,
    rprClientDayInMcastClassBEirOctets   Counter64,
    rprClientDayInMcastClassCPkts        Counter64,
    rprClientDayInMcastClassCOctets      Counter64,

    rprClientDayOutUcastClassAPkts       Counter64,
    rprClientDayOutUcastClassAOctets     Counter64,
    rprClientDayOutUcastClassBCirPkts    Counter64,
    rprClientDayOutUcastClassBCirOctets  Counter64,
    rprClientDayOutUcastClassBEirPkts    Counter64,
    rprClientDayOutUcastClassBEirOctets  Counter64,
    rprClientDayOutUcastClassCPkts       Counter64,
    rprClientDayOutUcastClassCOctets     Counter64,

    rprClientDayOutMcastClassAPkts       Counter64,
    rprClientDayOutMcastClassAOctets     Counter64,
    rprClientDayOutMcastClassBCirPkts    Counter64,
    rprClientDayOutMcastClassBCirOctets  Counter64,
    rprClientDayOutMcastClassBEirPkts    Counter64,
    rprClientDayOutMcastClassBEirOctets  Counter64,
    rprClientDayOutMcastClassCPkts       Counter64,
    rprClientDayOutMcastClassCOctets     Counter64
    }

rprClientDayIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    ::= { rprClientCountersDayEntry 1 }

rprClientDayTimeElapsed OBJECT-TYPE
    SYNTAX        Integer32
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The duration of the Day counters in seconds.
       Non-valid intervals will not be added to the Day counters.
       If, for some reason, such as an adjustment in the system's
       time-of-day clock, the Day interval exceeds the maximum
       value, the agent will return the maximum value."
    ::= { rprClientCountersDayEntry 2 }

rprClientDayInUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A unicast packets."
    ::= { rprClientCountersDayEntry 3 }

rprClientDayInUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A unicast octets."
    ::= { rprClientCountersDayEntry 4 }

rprClientDayInUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
```

```
      "The number of received in profile Class B unicast packets."
    ::= { rprClientCountersDayEntry 5 }

rprClientDayInUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B unicast octets."
    ::= { rprClientCountersDayEntry 6 }

rprClientDayInUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B unicast packets."
    ::= { rprClientCountersDayEntry 7 }

rprClientDayInUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B unicast octets."
    ::= { rprClientCountersDayEntry 8 }

rprClientDayInUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C unicast packets."
    ::= { rprClientCountersDayEntry 9 }

rprClientDayInUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C unicast octets."
    ::= { rprClientCountersDayEntry 10 }

rprClientDayInMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A multicast and broadcast packets"
    ::= { rprClientCountersDayEntry 11 }

rprClientDayInMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A multicast and broadcast octets"
    ::= { rprClientCountersDayEntry 12 }

rprClientDayInMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B multicast
```

```
        and broadcast packets"
    ::= { rprClientCountersDayEntry 13 }

rprClientDayInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B multicast
       and broadcast octets"
    ::= { rprClientCountersDayEntry 14 }

rprClientDayInMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B multicast
       and broadcast packets"
    ::= { rprClientCountersDayEntry 15 }

rprClientDayInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B multicast
       and broadcast octets"
    ::= { rprClientCountersDayEntry 16 }

rprClientDayInMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C multicast and broadcast packets"
    ::= { rprClientCountersDayEntry 17 }

rprClientDayInMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C multicast and broadcast octets"
    ::= { rprClientCountersDayEntry 18 }

rprClientDayOutUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A unicast packets."
    ::= { rprClientCountersDayEntry 19 }

rprClientDayOutUcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A unicast octets."
    ::= { rprClientCountersDayEntry 20 }

rprClientDayOutUcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
```

```
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B unicast packets."
    ::= { rprClientCountersDayEntry 21 }

rprClientDayOutUcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B unicast octets."
    ::= { rprClientCountersDayEntry 22 }

rprClientDayOutUcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B unicast
       packets"
    ::= { rprClientCountersDayEntry 23 }

rprClientDayOutUcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B unicast
       octets"
    ::= { rprClientCountersDayEntry 24 }

rprClientDayOutUcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C unicast packets"
    ::= { rprClientCountersDayEntry 25 }

rprClientDayOutUcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C unicast octets"
    ::= { rprClientCountersDayEntry 26 }

rprClientDayOutMcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A multicast and broadcast
       packets."
    ::= { rprClientCountersDayEntry 27 }

rprClientDayOutMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A multicast and broadcast
       octets."
    ::= { rprClientCountersDayEntry 28 }
```

```
rprClientDayOutMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B
       multicast and broadcast packets."
    ::= { rprClientCountersDayEntry 29 }

rprClientDayOutMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B
       multicast and broadcast octets."
    ::= { rprClientCountersDayEntry 30 }

rprClientDayOutMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B
       multicast and broadcast packets."
    ::= { rprClientCountersDayEntry 31 }

rprClientDayOutMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B
       multicast and broadcast octets."
    ::= { rprClientCountersDayEntry 32 }

rprClientDayOutMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C multicast and broadcast
       packets."
    ::= { rprClientCountersDayEntry 33 }

rprClientDayOutMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C multicast and broadcast
       octets."
    ::= { rprClientCountersDayEntry 34 }


--
-- 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."
```

```
    ::= { rprClientCounters 4 }

rprClientCountersStatsEntry OBJECT-TYPE
    SYNTAX        RprClientCountersStatsEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
      "An entry in the span stats table.

       ifCounterDiscontinuityTime in ifMIB is used with this table."
    INDEX { rprClientStatsIfIndex }
    ::= { rprClientCountersStatsTable 1 }

RprClientCountersStatsEntry ::= SEQUENCE {
    rprClientStatsIfIndex                 InterfaceIndex,

    rprClientStatsInUcastClassAPkts       Counter64,
    rprClientStatsInUcastClassAOctets     Counter64,
    rprClientStatsInUcastClassBCirPkts    Counter64,
    rprClientStatsInUcastClassBCirOctets  Counter64,
    rprClientStatsInUcastClassBEirPkts    Counter64,
    rprClientStatsInUcastClassBEirOctets  Counter64,
    rprClientStatsInUcastClassCPkts       Counter64,
    rprClientStatsInUcastClassCOctets     Counter64,

    rprClientStatsInMcastClassAPkts       Counter64,
    rprClientStatsInMcastClassAOctets     Counter64,
    rprClientStatsInMcastClassBCirPkts    Counter64,
    rprClientStatsInMcastClassBCirOctets  Counter64,
    rprClientStatsInMcastClassBEirPkts    Counter64,
    rprClientStatsInMcastClassBEirOctets  Counter64,
    rprClientStatsInMcastClassCPkts       Counter64,
    rprClientStatsInMcastClassCOctets     Counter64,

    rprClientStatsOutUcastClassAPkts      Counter64,
    rprClientStatsOutUcastClassAOctets    Counter64,
    rprClientStatsOutUcastClassBCirPkts   Counter64,
    rprClientStatsOutUcastClassBCirOctets Counter64,
    rprClientStatsOutUcastClassBEirPkts   Counter64,
    rprClientStatsOutUcastClassBEirOctets Counter64,
    rprClientStatsOutUcastClassCPkts      Counter64,
    rprClientStatsOutUcastClassCOctets    Counter64,

    rprClientStatsOutMcastClassAPkts      Counter64,
    rprClientStatsOutMcastClassAOctets    Counter64,
    rprClientStatsOutMcastClassBCirPkts   Counter64,
    rprClientStatsOutMcastClassBCirOctets Counter64,
    rprClientStatsOutMcastClassBEirPkts   Counter64,
    rprClientStatsOutMcastClassBEirOctets Counter64,
    rprClientStatsOutMcastClassCPkts      Counter64,
    rprClientStatsOutMcastClassCOctets    Counter64
    }

rprClientStatsIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    ::= { rprClientCountersStatsEntry 1 }

rprClientStatsInUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
```

```
        DESCRIPTION
          "The number of received Class A unicast packets."
        ::= { rprClientCountersStatsEntry 2 }

    rprClientStatsInUcastClassAOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of received Class A unicast octets."
        ::= { rprClientCountersStatsEntry 3 }

    rprClientStatsInUcastClassBCirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of received in profile Class B unicast packets."
        ::= { rprClientCountersStatsEntry 4 }

    rprClientStatsInUcastClassBCirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of received in profile Class B unicast octets."
        ::= { rprClientCountersStatsEntry 5 }

    rprClientStatsInUcastClassBEirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of received out of profile Class B unicast packets."
        ::= { rprClientCountersStatsEntry 6 }

    rprClientStatsInUcastClassBEirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of received out of profile Class B unicast octets."
        ::= { rprClientCountersStatsEntry 7 }

    rprClientStatsInUcastClassCPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of received Class C unicast packets."
        ::= { rprClientCountersStatsEntry 8 }

    rprClientStatsInUcastClassCOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of received Class C unicast octets."
        ::= { rprClientCountersStatsEntry 9 }

    rprClientStatsInMcastClassAPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
```

```
      "The number of received Class A multicast and broadcast packets"
    ::= { rprClientCountersStatsEntry 10 }

rprClientStatsInMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class A multicast and broadcast octets"
    ::= { rprClientCountersStatsEntry 11 }

rprClientStatsInMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B multicast
       and broadcast packets"
    ::= { rprClientCountersStatsEntry 12 }

rprClientStatsInMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received in profile Class B multicast
       and broadcast octets"
    ::= { rprClientCountersStatsEntry 13 }

rprClientStatsInMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B multicast
       and broadcast packets"
    ::= { rprClientCountersStatsEntry 14 }

rprClientStatsInMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received out of profile Class B multicast
       and broadcast octets"
    ::= { rprClientCountersStatsEntry 15 }

rprClientStatsInMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C multicast and broadcast packets"
    ::= { rprClientCountersStatsEntry 16 }

rprClientStatsInMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of received Class C multicast and broadcast octets"
    ::= { rprClientCountersStatsEntry 17 }

rprClientStatsOutUcastClassAPkts OBJECT-TYPE
    SYNTAX        Counter64
```

```
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class A unicast packets."
        ::= { rprClientCountersStatsEntry 18 }

    rprClientStatsOutUcastClassAOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class A unicast octets."
        ::= { rprClientCountersStatsEntry 19 }

    rprClientStatsOutUcastClassBCirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted in profile Class B unicast packets."
        ::= { rprClientCountersStatsEntry 20 }

    rprClientStatsOutUcastClassBCirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted in profile Class B unicast octets."
        ::= { rprClientCountersStatsEntry 21 }

    rprClientStatsOutUcastClassBEirPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted out of profile Class B unicast
           packets"
        ::= { rprClientCountersStatsEntry 22 }

    rprClientStatsOutUcastClassBEirOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted out of profile Class B unicast
           octets"
        ::= { rprClientCountersStatsEntry 23 }

    rprClientStatsOutUcastClassCPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class C unicast packets"
        ::= { rprClientCountersStatsEntry 24 }

    rprClientStatsOutUcastClassCOctets OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class C unicast octets"
        ::= { rprClientCountersStatsEntry 25 }

    rprClientStatsOutMcastClassAPkts OBJECT-TYPE
```

```
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A multicast and broadcast
       packets."
    ::= { rprClientCountersStatsEntry 26 }

rprClientStatsOutMcastClassAOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class A multicast and broadcast
       octets."
    ::= { rprClientCountersStatsEntry 27 }

rprClientStatsOutMcastClassBCirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B
       multicast and broadcast packets."
    ::= { rprClientCountersStatsEntry 28 }

rprClientStatsOutMcastClassBCirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted in profile Class B
       multicast and broadcast octets."
    ::= { rprClientCountersStatsEntry 29 }

rprClientStatsOutMcastClassBEirPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B
       multicast and broadcast packets."
    ::= { rprClientCountersStatsEntry 30 }

rprClientStatsOutMcastClassBEirOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted out of profile Class B
       multicast and broadcast octets."
    ::= { rprClientCountersStatsEntry 31 }

rprClientStatsOutMcastClassCPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted Class C multicast and broadcast
       packets."
    ::= { rprClientCountersStatsEntry 32 }

rprClientStatsOutMcastClassCOctets OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
```

```
        STATUS        current
        DESCRIPTION
          "The number of transmitted Class C multicast and broadcast
           octets."
        ::= { rprClientCountersStatsEntry 33 }


    --
    -- RPR error current counters
    --

    rprSpanCountersCurrentErrorTable OBJECT-TYPE
        SYNTAX        SEQUENCE OF RprSpanCountersCurrentErrorEntry
        MAX-ACCESS    not-accessible
        STATUS        current
        DESCRIPTION
            "The RPR Errors Current counters table."
        ::= { rprSpanErrorCounters 1 }

    rprSpanCountersCurrentErrorEntry OBJECT-TYPE
        SYNTAX        RprSpanCountersCurrentErrorEntry
        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 { rprSpanCurrentErrorIfIndex,
                rprSpanCurrentErrorSpan }
        ::= { rprSpanCountersCurrentErrorTable 1 }

    RprSpanCountersCurrentErrorEntry ::= SEQUENCE {
        rprSpanCurrentErrorIfIndex              InterfaceIndex,
        rprSpanCurrentErrorSpan                 RprSpan,

        rprSpanCurrentErrorTtlExpPkts           Counter64,
        rprSpanCurrentErrorTooLongPkts          Counter64,
        rprSpanCurrentErrorTooShortPkts         Counter64,
        rprSpanCurrentErrorBadHecPkts           Counter64,
        rprSpanCurrentErrorBadFcsPkts           Counter64,
        rprSpanCurrentErrorSelfSrcUcastPkts     Counter64,
        rprSpanCurrentErrorUnknownPktType       Counter64,
        rprSpanCurrentErrorPmdAbortPkts         Counter64
        }

    rprSpanCurrentErrorIfIndex OBJECT-TYPE
        SYNTAX        InterfaceIndex
        MAX-ACCESS    not-accessible
        STATUS        current
        DESCRIPTION
            "The ifIndex of this RPR interface."
        ::= { rprSpanCountersCurrentErrorEntry 1 }

    rprSpanCurrentErrorSpan 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."
        ::= { rprSpanCountersCurrentErrorEntry 2 }

    rprSpanCurrentErrorTtlExpPkts OBJECT-TYPE
```

```
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets that were dropped
         due to zero Time To Live (TTL).”
    ::= { rprSpanCountersCurrentErrorEntry 3 }

rprSpanCurrentErrorTooLongPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets that exceed the
         maximum permitted packet size.”
    ::= { rprSpanCountersCurrentErrorEntry 4 }

rprSpanCurrentErrorTooShortPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets shortest than the
         minimum permitted packet size.”
    ::= { rprSpanCountersCurrentErrorEntry 5 }

rprSpanCurrentErrorBadHecPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets with HEC error.”
    ::= { rprSpanCountersCurrentErrorEntry 6 }

rprSpanCurrentErrorBadFcsPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets with FCS error.”
    ::= { rprSpanCountersCurrentErrorEntry 7 }

rprSpanCurrentErrorSelfSrcUcastPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received unicast packets that were
         transmitted by the station itself i.e.
         the source MAC is equal to the interface MAC.”
    ::= { rprSpanCountersCurrentErrorEntry 8 }

rprSpanCurrentErrorUnknownPktType OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets with unknown or
         unsupported packet type value (PT).”
    ::= { rprSpanCountersCurrentErrorEntry 9 }

rprSpanCurrentErrorPmdAbortPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
```

```
    DESCRIPTION
      "The number of transmitted packets that were aborted by the PMD"
    ::= { rprSpanCountersCurrentErrorEntry 10 }


--
-- RPR error interval counters table
--

rprSpanCountersIntervalErrorTable OBJECT-TYPE
    SYNTAX        SEQUENCE OF RprSpanCountersIntervalErrorEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The RPR Errors Interval counters table."
    ::= { rprSpanErrorCounters 2 }

rprSpanCountersIntervalErrorEntry OBJECT-TYPE
    SYNTAX        RprSpanCountersIntervalErrorEntry
    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 { rprSpanIntervalErrorIfIndex,
            rprSpanIntervalErrorSpan,
            rprSpanIntervalErrorNumber }
    ::= { rprSpanCountersIntervalErrorTable 1 }

RprSpanCountersIntervalErrorEntry ::= SEQUENCE {
    rprSpanIntervalErrorIfIndex           InterfaceIndex,
    rprSpanIntervalErrorSpan              RprSpan,
    rprSpanIntervalErrorNumber            Integer32,
    rprSpanIntervalErrorValidData         TruthValue,
    rprSpanIntervalErrorTimeElapsed       Integer32,

    rprSpanIntervalErrorTtlExpPkts        Counter64,
    rprSpanIntervalErrorTooLongPkts       Counter64,
    rprSpanIntervalErrorTooShortPkts      Counter64,
    rprSpanIntervalErrorBadHecPkts        Counter64,
    rprSpanIntervalErrorBadFcsPkts        Counter64,
    rprSpanIntervalErrorSelfSrcUcastPkts  Counter64,
    rprSpanIntervalErrorUnknownPktType    Counter64,
    rprSpanIntervalErrorPmdAbortPkts      Counter64
   }

rprSpanIntervalErrorIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    ::= { rprSpanCountersIntervalErrorEntry 1 }

rprSpanIntervalErrorSpan 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."
    ::= { rprSpanCountersIntervalErrorEntry 2 }
```

```
rprSpanIntervalErrorNumber OBJECT-TYPE
    SYNTAX        Integer32 (1..96)
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        “A number between 1 and 96, which identifies the interval
         for which the set od 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.”
    ::= { rprSpanCountersIntervalErrorEntry 3 }

rprSpanIntervalErrorValidData OBJECT-TYPE
    SYNTAX        TruthValue
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “This variable indicates if the data for this interval is
         valid.”
    ::= { rprSpanCountersIntervalErrorEntry 4 }

rprSpanIntervalErrorTimeElapsed OBJECT-TYPE
    SYNTAX        Integer32
    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.”
    ::= { rprSpanCountersIntervalErrorEntry 5 }

rprSpanIntervalErrorTtlExpPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets that were dropped due to
         zero Time To Live (TTL) in a particular interval in the
         past 24 hours”
    ::= { rprSpanCountersIntervalErrorEntry 6 }

rprSpanIntervalErrorTooLongPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets that exceed the maximum
         permitted packet size in a particular interval in the
         past 24 hours”
    ::= { rprSpanCountersIntervalErrorEntry 7 }

rprSpanIntervalErrorTooShortPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets shortest than the
         minimum permitted packet size in a particular interval
         in the past 24 hours.”
    ::= { rprSpanCountersIntervalErrorEntry 8 }

rprSpanIntervalErrorBadHecPkts OBJECT-TYPE
    SYNTAX        Counter64
```

```
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received packets with HEC error in a
             particular interval in the past 24 hours"
        ::= { rprSpanCountersIntervalErrorEntry 9 }

    rprSpanIntervalErrorBadFcsPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received packets with FCSerror in a
             particular interval in the past 24 hours"
        ::= { rprSpanCountersIntervalErrorEntry 10 }

    rprSpanIntervalErrorSelfSrcUcastPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received unicast packets 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"
        ::= { rprSpanCountersIntervalErrorEntry 11 }

    rprSpanIntervalErrorUnknownPktType OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received unicast packets that are destined to
             the station and have unknown or unsupported packet type (PT),
             in a particular interval in the past 24 hours"
        ::= { rprSpanCountersIntervalErrorEntry 12 }

    rprSpanIntervalErrorPmdAbortPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received unicast packets that were aborted
             by the PMD layer, in a particular interval in the past
             24 hours"
        ::= { rprSpanCountersIntervalErrorEntry 13 }

    --
    -- RPR error day (24 hour summaries) counters table
    --

    rprSpanCountersDayErrorTable OBJECT-TYPE
        SYNTAX        SEQUENCE OF RprSpanCountersDayErrorEntry
        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 }

    rprSpanCountersDayErrorEntry OBJECT-TYPE
        SYNTAX        RprSpanCountersDayErrorEntry
        MAX-ACCESS    not-accessible
        STATUS        current
        DESCRIPTION
```

```
        "An entry in the RPR Errpr Day table."
    INDEX { rprSpanDayErrorIfIndex,
            rprSpanDayErrorSpan }
    ::= { rprSpanCountersDayErrorTable 1 }

RprSpanCountersDayErrorEntry ::= SEQUENCE {
    rprSpanDayErrorIfIndex              InterfaceIndex,
    rprSpanDayErrorSpan                 RprSpan,
    rprSpanDayErrorTimeElapsed          Integer32,

    rprSpanDayErrorTtlExpPkts           Counter64,
    rprSpanDayErrorTooLongPkts          Counter64,
    rprSpanDayErrorTooShortPkts         Counter64,
    rprSpanDayErrorBadHecPkts           Counter64,
    rprSpanDayErrorBadFcsPkts           Counter64,
    rprSpanDayErrorSelfSrcUcastPkts     Counter64,
    rprSpanDayErrorUnknownPktType       Counter64,
    rprSpanDayErrorPmdAbortPkts         Counter64
    }

rprSpanDayErrorIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The ifIndex of this RPR interface."
    ::= { rprSpanCountersDayErrorEntry 1 }

rprSpanDayErrorSpan 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."
    ::= { rprSpanCountersDayErrorEntry 2 }

rprSpanDayErrorTimeElapsed OBJECT-TYPE
    SYNTAX        Integer32
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The duration of the Day counters in seconds.
       Non-valid intervals will not be added to the Day counters.
       If, for some reason, such as an adjustment in the system's
       time-of-day clock, the Day interval exceeds the maximum
       value, the agent will return the maximum value."
    ::= { rprSpanCountersDayErrorEntry 3 }

rprSpanDayErrorTtlExpPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received packets that were dropped
         due to zero Time To Live (TTL)."
    ::= { rprSpanCountersDayErrorEntry 4 }

rprSpanDayErrorTooLongPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received packets that exceed the
         maximum permitted packet size."
```

```
    ::= { rprSpanCountersDayErrorEntry 5 }

rprSpanDayErrorTooShortPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received packets shortest than the
         minimum permitted packet size."
    ::= { rprSpanCountersDayErrorEntry 6 }

rprSpanDayErrorBadHecPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received packets with HEC error."
    ::= { rprSpanCountersDayErrorEntry 7 }

rprSpanDayErrorBadFcsPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received packets with FCS error."
    ::= { rprSpanCountersDayErrorEntry 8 }

rprSpanDayErrorSelfSrcUcastPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received unicast packets that were
         transmitted by the station itself i.e.
         the source MAC is equal to the interface MAC."
    ::= { rprSpanCountersDayErrorEntry 9 }

rprSpanDayErrorUnknownPktType OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        "The number of received packets with unknown or
         unsupported packet type (PT)."
    ::= { rprSpanCountersDayErrorEntry 10 }

rprSpanDayErrorPmdAbortPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
      "The number of transmitted packets that were aborted by the PMD"
    ::= { rprSpanCountersDayErrorEntry 11 }

--
-- RPR error total continuously running counters table
--

rprSpanCountersStatsErrorTable OBJECT-TYPE
    SYNTAX        SEQUENCE OF RprSpanCountersStatsErrorEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        "The RPR Errors total counters table."
    ::= { rprSpanErrorCounters 4 }
```

```
rprSpanCountersStatsErrorEntry OBJECT-TYPE
    SYNTAX        RprSpanCountersStatsErrorEntry
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        “An entry in the span error counter table.

         The DiscontinuityTime for this table is defined in the
         rprSpanTable.”
    INDEX { rprSpanStatsErrorIfIndex,
            rprSpanStatsErrorSpan }
    ::= { rprSpanCountersStatsErrorTable 1 }

RprSpanCountersStatsErrorEntry ::= SEQUENCE {
    rprSpanStatsErrorIfIndex             InterfaceIndex,
    rprSpanStatsErrorSpan                RprSpan,

    rprSpanStatsErrorTtlExpPkts          Counter64,
    rprSpanStatsErrorTooLongPkts         Counter64,
    rprSpanStatsErrorTooShortPkts        Counter64,
    rprSpanStatsErrorBadHecPkts          Counter64,
    rprSpanStatsErrorBadFcsPkts          Counter64,
    rprSpanStatsErrorSelfSrcUcastPkts    Counter64,
    rprSpanStatsErrorUnknownPktType      Counter64,
    rprSpanStatsErrorPmdAbortPkts        Counter64
    }

rprSpanStatsErrorIfIndex OBJECT-TYPE
    SYNTAX        InterfaceIndex
    MAX-ACCESS    not-accessible
    STATUS        current
    DESCRIPTION
        “The ifIndex of this RPR interface.”
    ::= { rprSpanCountersStatsErrorEntry 1 }

rprSpanStatsErrorSpan 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.”
    ::= { rprSpanCountersStatsErrorEntry 2 }

rprSpanStatsErrorTtlExpPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets that were dropped
         due to zero Time To Live (TTL).”
    ::= { rprSpanCountersStatsErrorEntry 3 }

rprSpanStatsErrorTooLongPkts OBJECT-TYPE
    SYNTAX        Counter64
    MAX-ACCESS    read-only
    STATUS        current
    DESCRIPTION
        “The number of received packets that exceed the
         maximum permitted packet size.”
    ::= { rprSpanCountersStatsErrorEntry 4 }

rprSpanStatsErrorTooShortPkts OBJECT-TYPE
    SYNTAX        Counter64
```

```
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received packets shortest than the
             minimum permitted packet size."
        ::= { rprSpanCountersStatsErrorEntry 5 }

    rprSpanStatsErrorBadHecPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received packets with HEC error."
        ::= { rprSpanCountersStatsErrorEntry 6 }

    rprSpanStatsErrorBadFcsPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received packets with FCS error."
        ::= { rprSpanCountersStatsErrorEntry 7 }

    rprSpanStatsErrorSelfSrcUcastPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received unicast packets that were
             transmitted by the station itself i.e.
             the source MAC is equal to the interface MAC."
        ::= { rprSpanCountersStatsErrorEntry 8 }

    rprSpanStatsErrorUnknownPktType OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
            "The number of received packets with unknown or
             unsupported packet type (PT)."
        ::= { rprSpanCountersStatsErrorEntry 9 }

    rprSpanStatsErrorPmdAbortPkts OBJECT-TYPE
        SYNTAX        Counter64
        MAX-ACCESS    read-only
        STATUS        current
        DESCRIPTION
          "The number of transmitted packets that were aborted by the PMD"
        ::= { rprSpanCountersStatsErrorEntry 10 }



    --
    -- conformance information
    --

    rprGroups      OBJECT IDENTIFIER ::= { rprConformance 1 }
    rprCompliances OBJECT IDENTIFIER ::= { rprConformance 2 }


    rprModuleIntervalStatsCompliance MODULE-COMPLIANCE
        STATUS  current
        DESCRIPTION
          "The compliance statement for agent that support RPR operation
           with current and intervals statistics collections."
```

```
    MODULE  -- this module
        MANDATORY-GROUPS { rprIfGroup,
                           rprSpanGroup,
                           rprTopoGroup,
                           rprProtGroup,
                           rprFairnessGroup,
                           rprOamGroup,
                           rprSpanCurrentGroup,
                           rprSpanIntervalGroup,
                           rprClientCurrentGroup,
                           rprClientIntervalGroup,
                           rprErrorCurrentGroup,
                           rprErrorIntervalGroup }

        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 15min Intervals table."

        GROUP rprSpanStatsGroup
        DESCRIPTION
          "Collection of RPR MAC Span total counters."

        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 15min Intervals table."

        GROUP rprClientStatsGroup
        DESCRIPTION
          "Collection of RPR MAC Client interface total counters."

        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 15min Intervals table."

        GROUP rprErrorStatsGroup
        DESCRIPTION
          "Collection of RPR MAC Span error total counters."

     ::= { rprCompliances 1 }


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,
                        rprTopoGroup,
                        rprProtGroup,
                        rprFairnessGroup,
                        rprOamGroup,
                        rprSpanStatsGroup,
                        rprClientStatsGroup,
                        rprErrorStatsGroup }

    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 15min
       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 15min 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 15min 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 15min 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 15min 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 15min Intervals table."
```

```
     ::= { rprCompliances 2 }

--
-- Units of conformance.
--

rprIfGroup   OBJECT-GROUP
     OBJECTS {
               rprIfStationsOnRing,
               rprIfReversionMode,
               rprIfHoldOffTimer,
               rprIfProtectionWTR,
               rprIfFastTimer,
               rprIfSlowTimer,
               rprIfKeepaliveTimout,
               rprIfDropBadFcsControl,
               rprIfFairnessMode,
               rprIfPtqSize,
               rprIfStqSize,
               rprIfWrapControl,
               rprIfMacCapabilities,
               rprIfRingCapabilities,
               rprIfStatsClear,
               rprIfStatsIntervalClear,
               rprIfTimeElapsed,
               rprIfValidIntervals
             }
     STATUS  current
     DESCRIPTION
            “Collection of objects needed for RPR MAC
             configuration.”
     ::= { rprGroups 1 }

rprSpanGroup   OBJECT-GROUP
     OBJECTS {
               rprSpanLowerLayerIfIndex,
               rprSpanSpeed,
               rprSpanNeighbor,
               rprSpanProtectionState,
               rprSpanProtectionCommand,
               rprSpanProtectionStatus,
               rprSpanProtectionCount,
               rprSpanProtectionDuration,
               rprSpanProtectionLastActivationTime,
               rprSpanStatsDiscontinuityTime
             }
     STATUS  current
     DESCRIPTION
            “Collection of objects needed for RPR Span
             configuration and monitoring.”
     ::= { rprGroups 2 }

rprTopoGroup   OBJECT-GROUP
     OBJECTS {
               rprTopoImageMacAddress,
               rprTopoImageCapabilites,
               rprTopoImageFairnessRinglet0Weight,
               rprTopoImageFairnessRinglet1Weight,
               rprTopoImageRinglet0Neighbor,
               rprTopoImageRinglet1Neighbor,
               rprTopoImageRinglet0ReservedRate,
               rprTopoImageRinglet1ReservedRate
             }
     STATUS  current
```

```
        DESCRIPTION
               "Collection of objects needed for RPR Topology
                discovery."
        ::= { rprGroups 3 }

    rprProtGroup   OBJECT-GROUP
        OBJECTS {
                 rprProtectImageActive,
                 rprProtectImageStatus,
                 rprProtectImageLastChange
               }
        STATUS  current
        DESCRIPTION
               "Collection of objects needed for RPR ring protection
                status."
        ::= { rprGroups 4 }

    rprFairnessGroup   OBJECT-GROUP
        OBJECTS {
                 rprFairnessRingletWeight,
                 rprFairnessReservedRate,
                 rprFairnessMaxAllowed,
                 rprFairnessAgeCoef,
                 rprFairnessRampCoef,
                 rprFairnessLpCoef,
                 rprFairnessFullThreshold,
                 rprFairnessHighThreshold,
                 rprFairnessLowThreshold
               }
        STATUS  current
        DESCRIPTION
               "Collection of objects needed for RPR Fairness
                configuration."
        ::= { rprGroups 5 }

    rprOamGroup   OBJECT-GROUP
        OBJECTS {
                 rprOamEchoDestAddress,
                 rprOamEchoRequestRinglet,
                 rprOamEchoResponseRinglet,
                 rprOamEchoCos,
                 rprOamEchoProtection,
                 rprOamEchoReqCount,
                 rprOamEchoTimerTimeout,
                 rprOamEchoControl,
                 rprOamEchoRespCount,
                 rprOamEchoAvRespTime,
                 rprOamEchoRespStatus
               }
        STATUS  current
        DESCRIPTION
               "Collection of objects needed for RPR OAM
                configuration."
        ::= { rprGroups 6 }

    rprSpanCurrentGroup   OBJECT-GROUP
        OBJECTS {
                 rprSpanCurrentInUcastClassAPkts,
                 rprSpanCurrentInUcastClassAOctets,
                 rprSpanCurrentInUcastClassBCirPkts,
                 rprSpanCurrentInUcastClassBCirOctets,
                 rprSpanCurrentInUcastClassBEirPkts,
                 rprSpanCurrentInUcastClassBEirOctets,
                 rprSpanCurrentInUcastClassCPkts,
                 rprSpanCurrentInUcastClassCOctets,
```

```
              rprSpanCurrentInMcastClassAPkts,
              rprSpanCurrentInMcastClassAOctets,
              rprSpanCurrentInMcastClassBCirPkts,
              rprSpanCurrentInMcastClassBCirOctets,
              rprSpanCurrentInMcastClassBEirPkts,
              rprSpanCurrentInMcastClassBEirOctets,
              rprSpanCurrentInMcastClassCPkts,
              rprSpanCurrentInMcastClassCOctets,
              rprSpanCurrentOutUcastClassAPkts,
              rprSpanCurrentOutUcastClassAOctets,
              rprSpanCurrentOutUcastClassBCirPkts,
              rprSpanCurrentOutUcastClassBCirOctets,
              rprSpanCurrentOutUcastClassBEirPkts,
              rprSpanCurrentOutUcastClassBEirOctets,
              rprSpanCurrentOutUcastClassCPkts,
              rprSpanCurrentOutUcastClassCOctets,
              rprSpanCurrentOutMcastClassAPkts,
              rprSpanCurrentOutMcastClassAOctets,
              rprSpanCurrentOutMcastClassBCirPkts,
              rprSpanCurrentOutMcastClassBCirOctets,
              rprSpanCurrentOutMcastClassBEirPkts,
              rprSpanCurrentOutMcastClassBEirOctets,
              rprSpanCurrentOutMcastClassCPkts,
              rprSpanCurrentOutMcastClassCOctets
            }
     STATUS  current
     DESCRIPTION
            "Collection of objects counting MAC span current
             statistics."
     ::= { rprGroups 7 }

rprSpanIntervalGroup   OBJECT-GROUP
     OBJECTS {
              rprSpanIntervalValidData,
              rprSpanIntervalTimeElapsed,
              rprSpanIntervalInUcastClassAPkts,
              rprSpanIntervalInUcastClassAOctets,
              rprSpanIntervalInUcastClassBCirPkts,
              rprSpanIntervalInUcastClassBCirOctets,
              rprSpanIntervalInUcastClassBEirPkts,
              rprSpanIntervalInUcastClassBEirOctets,
              rprSpanIntervalInUcastClassCPkts,
              rprSpanIntervalInUcastClassCOctets,
              rprSpanIntervalInMcastClassAPkts,
              rprSpanIntervalInMcastClassAOctets,
              rprSpanIntervalInMcastClassBCirPkts,
              rprSpanIntervalInMcastClassBCirOctets,
              rprSpanIntervalInMcastClassBEirPkts,
              rprSpanIntervalInMcastClassBEirOctets,
              rprSpanIntervalInMcastClassCPkts,
              rprSpanIntervalInMcastClassCOctets,
              rprSpanIntervalOutUcastClassAPkts,
              rprSpanIntervalOutUcastClassAOctets,
              rprSpanIntervalOutUcastClassBCirPkts,
              rprSpanIntervalOutUcastClassBCirOctets,
              rprSpanIntervalOutUcastClassBEirPkts,
              rprSpanIntervalOutUcastClassBEirOctets,
              rprSpanIntervalOutUcastClassCPkts,
              rprSpanIntervalOutUcastClassCOctets,
              rprSpanIntervalOutMcastClassAPkts,
              rprSpanIntervalOutMcastClassAOctets,
              rprSpanIntervalOutMcastClassBCirPkts,
              rprSpanIntervalOutMcastClassBCirOctets,
              rprSpanIntervalOutMcastClassBEirPkts,
              rprSpanIntervalOutMcastClassBEirOctets,
```

```
                rprSpanIntervalOutMcastClassCPkts,
                rprSpanIntervalOutMcastClassCOctets
              }
      STATUS  current
      DESCRIPTION
             "Collection of objects counting MAC span intervals
              statistics."
      ::= { rprGroups 8 }

rprSpanDayGroup   OBJECT-GROUP
      OBJECTS {
                rprSpanDayTimeElapsed,
                rprSpanDayInUcastClassAPkts,
                rprSpanDayInUcastClassAOctets,
                rprSpanDayInUcastClassBCirPkts,
                rprSpanDayInUcastClassBCirOctets,
                rprSpanDayInUcastClassBEirPkts,
                rprSpanDayInUcastClassBEirOctets,
                rprSpanDayInUcastClassCPkts,
                rprSpanDayInUcastClassCOctets,
                rprSpanDayInMcastClassAPkts,
                rprSpanDayInMcastClassAOctets,
                rprSpanDayInMcastClassBCirPkts,
                rprSpanDayInMcastClassBCirOctets,
                rprSpanDayInMcastClassBEirPkts,
                rprSpanDayInMcastClassBEirOctets,
                rprSpanDayInMcastClassCPkts,
                rprSpanDayInMcastClassCOctets,
                rprSpanDayOutUcastClassAPkts,
                rprSpanDayOutUcastClassAOctets,
                rprSpanDayOutUcastClassBCirPkts,
                rprSpanDayOutUcastClassBCirOctets,
                rprSpanDayOutUcastClassBEirPkts,
                rprSpanDayOutUcastClassBEirOctets,
                rprSpanDayOutUcastClassCPkts,
                rprSpanDayOutUcastClassCOctets,
                rprSpanDayOutMcastClassAPkts,
                rprSpanDayOutMcastClassAOctets,
                rprSpanDayOutMcastClassBCirPkts,
                rprSpanDayOutMcastClassBCirOctets,
                rprSpanDayOutMcastClassBEirPkts,
                rprSpanDayOutMcastClassBEirOctets,
                rprSpanDayOutMcastClassCPkts,
                rprSpanDayOutMcastClassCOctets
              }
      STATUS  current
      DESCRIPTION
             "Collection of objects counting MAC span 24 hours
              statistics."
      ::= { rprGroups 9 }

rprSpanStatsGroup   OBJECT-GROUP
      OBJECTS {
                rprSpanStatsInUcastClassAPkts,
                rprSpanStatsInUcastClassAOctets,
                rprSpanStatsInUcastClassBCirPkts,
                rprSpanStatsInUcastClassBCirOctets,
                rprSpanStatsInUcastClassBEirPkts,
                rprSpanStatsInUcastClassBEirOctets,
                rprSpanStatsInUcastClassCPkts,
                rprSpanStatsInUcastClassCOctets,
                rprSpanStatsInMcastClassAPkts,
                rprSpanStatsInMcastClassAOctets,
                rprSpanStatsInMcastClassBCirPkts,
                rprSpanStatsInMcastClassBCirOctets,
```

```
              rprSpanStatsInMcastClassBEirPkts,
              rprSpanStatsInMcastClassBEirOctets,
              rprSpanStatsInMcastClassCPkts,
              rprSpanStatsInMcastClassCOctets,
              rprSpanStatsOutUcastClassAPkts,
              rprSpanStatsOutUcastClassAOctets,
              rprSpanStatsOutUcastClassBCirPkts,
              rprSpanStatsOutUcastClassBCirOctets,
              rprSpanStatsOutUcastClassBEirPkts,
              rprSpanStatsOutUcastClassBEirOctets,
              rprSpanStatsOutUcastClassCPkts,
              rprSpanStatsOutUcastClassCOctets,
              rprSpanStatsOutMcastClassAPkts,
              rprSpanStatsOutMcastClassAOctets,
              rprSpanStatsOutMcastClassBCirPkts,
              rprSpanStatsOutMcastClassBCirOctets,
              rprSpanStatsOutMcastClassBEirPkts,
              rprSpanStatsOutMcastClassBEirOctets,
              rprSpanStatsOutMcastClassCPkts,
              rprSpanStatsOutMcastClassCOctets
            }
     STATUS  current
     DESCRIPTION
            "Collection of objects counting MAC span total
             statistics."
     ::= { rprGroups 10 }

rprClientCurrentGroup   OBJECT-GROUP
     OBJECTS {
              rprClientCurrentInUcastClassAPkts,
              rprClientCurrentInUcastClassAOctets,
              rprClientCurrentInUcastClassBCirPkts,
              rprClientCurrentInUcastClassBCirOctets,
              rprClientCurrentInUcastClassBEirPkts,
              rprClientCurrentInUcastClassBEirOctets,
              rprClientCurrentInUcastClassCPkts,
              rprClientCurrentInUcastClassCOctets,
              rprClientCurrentInMcastClassAPkts,
              rprClientCurrentInMcastClassAOctets,
              rprClientCurrentInMcastClassBCirPkts,
              rprClientCurrentInMcastClassBCirOctets,
              rprClientCurrentInMcastClassBEirPkts,
              rprClientCurrentInMcastClassBEirOctets,
              rprClientCurrentInMcastClassCPkts,
              rprClientCurrentInMcastClassCOctets,
              rprClientCurrentOutUcastClassAPkts,
              rprClientCurrentOutUcastClassAOctets,
              rprClientCurrentOutUcastClassBCirPkts,
              rprClientCurrentOutUcastClassBCirOctets,
              rprClientCurrentOutUcastClassBEirPkts,
              rprClientCurrentOutUcastClassBEirOctets,
              rprClientCurrentOutUcastClassCPkts,
              rprClientCurrentOutUcastClassCOctets,
              rprClientCurrentOutMcastClassAPkts,
              rprClientCurrentOutMcastClassAOctets,
              rprClientCurrentOutMcastClassBCirPkts,
              rprClientCurrentOutMcastClassBCirOctets,
              rprClientCurrentOutMcastClassBEirPkts,
              rprClientCurrentOutMcastClassBEirOctets,
              rprClientCurrentOutMcastClassCPkts,
              rprClientCurrentOutMcastClassCOctets
            }
     STATUS  current
     DESCRIPTION
            "Collection of objects counting MAC client interface
```

```
                current statistics."
        ::= { rprGroups 11 }

    rprClientIntervalGroup   OBJECT-GROUP
        OBJECTS {
                 rprClientIntervalValidData,
                 rprClientIntervalTimeElapsed,
                 rprClientIntervalInUcastClassAPkts,
                 rprClientIntervalInUcastClassAOctets,
                 rprClientIntervalInUcastClassBCirPkts,
                 rprClientIntervalInUcastClassBCirOctets,
                 rprClientIntervalInUcastClassBEirPkts,
                 rprClientIntervalInUcastClassBEirOctets,
                 rprClientIntervalInUcastClassCPkts,
                 rprClientIntervalInUcastClassCOctets,
                 rprClientIntervalInMcastClassAPkts,
                 rprClientIntervalInMcastClassAOctets,
                 rprClientIntervalInMcastClassBCirPkts,
                 rprClientIntervalInMcastClassBCirOctets,
                 rprClientIntervalInMcastClassBEirPkts,
                 rprClientIntervalInMcastClassBEirOctets,
                 rprClientIntervalInMcastClassCPkts,
                 rprClientIntervalInMcastClassCOctets,
                 rprClientIntervalOutUcastClassAPkts,
                 rprClientIntervalOutUcastClassAOctets,
                 rprClientIntervalOutUcastClassBCirPkts,
                 rprClientIntervalOutUcastClassBCirOctets,
                 rprClientIntervalOutUcastClassBEirPkts,
                 rprClientIntervalOutUcastClassBEirOctets,
                 rprClientIntervalOutUcastClassCPkts,
                 rprClientIntervalOutUcastClassCOctets,
                 rprClientIntervalOutMcastClassAPkts,
                 rprClientIntervalOutMcastClassAOctets,
                 rprClientIntervalOutMcastClassBCirPkts,
                 rprClientIntervalOutMcastClassBCirOctets,
                 rprClientIntervalOutMcastClassBEirPkts,
                 rprClientIntervalOutMcastClassBEirOctets,
                 rprClientIntervalOutMcastClassCPkts,
                 rprClientIntervalOutMcastClassCOctets
                }
        STATUS  current
        DESCRIPTION
                "Collection of objects counting MAC client interface
                 intervals statistics."
        ::= { rprGroups 12 }

    rprClientDayGroup   OBJECT-GROUP
        OBJECTS {
                 rprClientDayTimeElapsed,
                 rprClientDayInUcastClassAPkts,
                 rprClientDayInUcastClassAOctets,
                 rprClientDayInUcastClassBCirPkts,
                 rprClientDayInUcastClassBCirOctets,
                 rprClientDayInUcastClassBEirPkts,
                 rprClientDayInUcastClassBEirOctets,
                 rprClientDayInUcastClassCPkts,
                 rprClientDayInUcastClassCOctets,
                 rprClientDayInMcastClassAPkts,
                 rprClientDayInMcastClassAOctets,
                 rprClientDayInMcastClassBCirPkts,
                 rprClientDayInMcastClassBCirOctets,
                 rprClientDayInMcastClassBEirPkts,
                 rprClientDayInMcastClassBEirOctets,
                 rprClientDayInMcastClassCPkts,
                 rprClientDayInMcastClassCOctets,
```

```
              rprClientDayOutUcastClassAPkts,
              rprClientDayOutUcastClassAOctets,
              rprClientDayOutUcastClassBCirPkts,
              rprClientDayOutUcastClassBCirOctets,
              rprClientDayOutUcastClassBEirPkts,
              rprClientDayOutUcastClassBEirOctets,
              rprClientDayOutUcastClassCPkts,
              rprClientDayOutUcastClassCOctets,
              rprClientDayOutMcastClassAPkts,
              rprClientDayOutMcastClassAOctets,
              rprClientDayOutMcastClassBCirPkts,
              rprClientDayOutMcastClassBCirOctets,
              rprClientDayOutMcastClassBEirPkts,
              rprClientDayOutMcastClassBEirOctets,
              rprClientDayOutMcastClassCPkts,
              rprClientDayOutMcastClassCOctets
            }
     STATUS  current
     DESCRIPTION
            "Collection of objects counting MAC client interface
             24 hours statistics."
     ::= { rprGroups 13 }

rprClientStatsGroup   OBJECT-GROUP
     OBJECTS {
              rprClientStatsInUcastClassAPkts,
              rprClientStatsInUcastClassAOctets,
              rprClientStatsInUcastClassBCirPkts,
              rprClientStatsInUcastClassBCirOctets,
              rprClientStatsInUcastClassBEirPkts,
              rprClientStatsInUcastClassBEirOctets,
              rprClientStatsInUcastClassCPkts,
              rprClientStatsInUcastClassCOctets,
              rprClientStatsInMcastClassAPkts,
              rprClientStatsInMcastClassAOctets,
              rprClientStatsInMcastClassBCirPkts,
              rprClientStatsInMcastClassBCirOctets,
              rprClientStatsInMcastClassBEirPkts,
              rprClientStatsInMcastClassBEirOctets,
              rprClientStatsInMcastClassCPkts,
              rprClientStatsInMcastClassCOctets,
              rprClientStatsOutUcastClassAPkts,
              rprClientStatsOutUcastClassAOctets,
              rprClientStatsOutUcastClassBCirPkts,
              rprClientStatsOutUcastClassBCirOctets,
              rprClientStatsOutUcastClassBEirPkts,
              rprClientStatsOutUcastClassBEirOctets,
              rprClientStatsOutUcastClassCPkts,
              rprClientStatsOutUcastClassCOctets,
              rprClientStatsOutMcastClassAPkts,
              rprClientStatsOutMcastClassAOctets,
              rprClientStatsOutMcastClassBCirPkts,
              rprClientStatsOutMcastClassBCirOctets,
              rprClientStatsOutMcastClassBEirPkts,
              rprClientStatsOutMcastClassBEirOctets,
              rprClientStatsOutMcastClassCPkts,
              rprClientStatsOutMcastClassCOctets
            }
     STATUS  current
     DESCRIPTION
            "Collection of objects counting MAC client interface
             total statistics."
     ::= { rprGroups 14 }

rprErrorCurrentGroup   OBJECT-GROUP
```

```
        OBJECTS {
                 rprSpanCurrentErrorTtlExpPkts,
                 rprSpanCurrentErrorTooLongPkts,
                 rprSpanCurrentErrorTooShortPkts,
                 rprSpanCurrentErrorBadHecPkts,
                 rprSpanCurrentErrorBadFcsPkts,
                 rprSpanCurrentErrorSelfSrcUcastPkts,
                 rprSpanCurrentErrorUnknownPktType,
                 rprSpanCurrentErrorPmdAbortPkts
               }
        STATUS  current
        DESCRIPTION
               "Collection of objects counting MAC span error
                current statistics."
        ::= { rprGroups 15 }

    rprErrorIntervalGroup   OBJECT-GROUP
        OBJECTS {
                 rprSpanIntervalErrorValidData,
                 rprSpanIntervalErrorTimeElapsed,
                 rprSpanIntervalErrorTtlExpPkts,
                 rprSpanIntervalErrorTooLongPkts,
                 rprSpanIntervalErrorTooShortPkts,
                 rprSpanIntervalErrorBadHecPkts,
                 rprSpanIntervalErrorBadFcsPkts,
                 rprSpanIntervalErrorSelfSrcUcastPkts,
                 rprSpanIntervalErrorUnknownPktType,
                 rprSpanIntervalErrorPmdAbortPkts
               }
        STATUS  current
        DESCRIPTION
               "Collection of objects counting MAC span error
                intervals statistics."
        ::= { rprGroups 16 }

    rprErrorDayGroup   OBJECT-GROUP
        OBJECTS {
                 rprSpanDayErrorTimeElapsed,
                 rprSpanDayErrorTtlExpPkts,
                 rprSpanDayErrorTooLongPkts,
                 rprSpanDayErrorTooShortPkts,
                 rprSpanDayErrorBadHecPkts,
                 rprSpanDayErrorBadFcsPkts,
                 rprSpanDayErrorSelfSrcUcastPkts,
                 rprSpanDayErrorUnknownPktType,
                 rprSpanDayErrorPmdAbortPkts
               }
        STATUS  current
        DESCRIPTION
               "Collection of objects counting MAC span error
                24 hours statistics."
        ::= { rprGroups 17 }

    rprErrorStatsGroup   OBJECT-GROUP
        OBJECTS {
                 rprSpanStatsErrorTtlExpPkts,
                 rprSpanStatsErrorTooLongPkts,
                 rprSpanStatsErrorTooShortPkts,
                 rprSpanStatsErrorBadHecPkts,
                 rprSpanStatsErrorBadFcsPkts,
                 rprSpanStatsErrorSelfSrcUcastPkts,
                 rprSpanStatsErrorUnknownPktType,
                 rprSpanStatsErrorPmdAbortPkts
               }
        STATUS  current
```

```
    DESCRIPTION
           "Collection of objects counting MAC span error
            total statistics."
    ::= { rprGroups 18 }

END
```

Annex E

(normative)

802.1D and 802.1Q bridging conformance

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	Draft 0.2 for TF review, based on comments to Draft 0.1.
Draft 0.3, June 2002	Draft 0.3 for WG review, based on comments to Draft 0.2.
Draft 1.0, August 2002	Draft 1.0 for TF review, based on comments to Draft 0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, based on comments to Draft 1.0.
Draft 2.0, November 2002	Draft 2.0 for WG ballot, based on comments to Draft 1.1.

Editors' Notes: *To be removed prior to final publication.*

Major unresolved issues:

Anytime unicast traffic is destination stripped by one station and replicated by others to their bridging relay, there is an issue with persistent bridged network flooding.
Need proposed text for unidirectional / bidirectional flooding
Need proposed text for minimizing frame duplication and frame misordering
Need proposed text to support source stripping for bridged frames
Need proposed text for spatial reuse of bridged traffic
Destination / Source address parameters in the frame should define whether they support local/non-local end station addresses or just local end station addresses.
Include a section defining how end stations directly attached to the ring interact with bridges on the ring. Also describe end station transmit/receive rules in order to maintain compatibility with bridges.
Receive rules need to be modified such that a bridge station does not copy frames where the frame destination address does not match the station address. It only copies broadcast/multicast frames. All other frames are transited per the MAC receive rules.

Editors' Notes: *To be removed prior to final publication.*

Additional comments picked up by section editor while working on current draft:

F.1, pg 285, item 7 should be removed from the list. The flooding indicator is more of an implementation specific used to meet the requirements defined by the other list items. Reference to a flooding indicator should not be in the draft at this time. Support of a flooding indicator worked its way into the text through specific wording to the resolution of a CRG/WG approved comment without general approval of any of the BAH bridging proposals. In fact, all specific proposed BAH motions at July meeting to adopt this concept either failed or died due to lack of support. All references to the flooding indicator here should be removed in subsequent drafts until voted/approved by the WG.
Need to define the set of MAC options required to support the bridge client.
Pg 289, line 21needs to cover cases of passing frames to bridge management entity. In general need to distinguish between when frames are passed to bridge relay vs. bridge management entity. Is there some kind of MAC address table in the MAC receiver which is used to determine when frames are sent to the bridge management entity???
F.3 pg 290, line 23. Item A should say "MAC client" instead of "MAC relay entity".

E.1 Bridging overview

This section of the draft is not intended to define a new bridging specification. IEEE Std 802.1D-1990 (subsequently republished as ISO/IEC 15802-3:1998 [IEEE Std 802.1D, 1998 Edition]) specifies an architecture and protocol for the interconnection of IEEE802 LANs below the MAC service boundary. Within this context, the RPR network defines a ring topology forming a broadcast media where specific access control mechanisms are employed by the MAC in order to achieve frame delivery and spatial reuse on the ring media. 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-1998 and Std 802.1Q-1998 respectively.

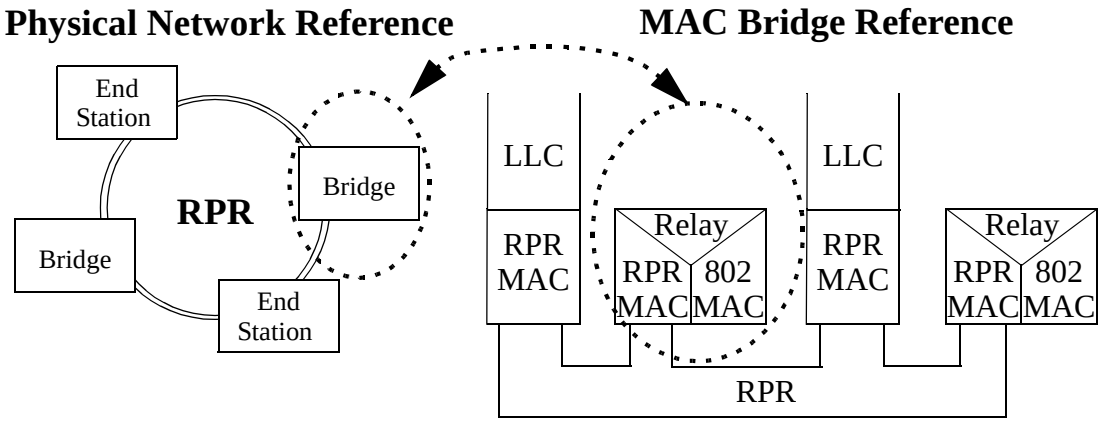


Figure E.1—Bridge architecture reference

The MAC bridging reference model for the RPR MAC is shown in Figure E.1. The bridging reference model supports an RPR network consisting of end stations and IEEE Std 802.1D-1998 (or IEEE Std 802.1Q-1998) transparent bridges, while the RPR acts as the shared broadcast media. Traffic may originate or terminate at either local or remote end stations, and may be forwarded across the RPR to other 802 networks by transparent bridges. Local end stations are end stations that directly attach to the RPR. Remote end stations are end stations which originate and terminate LAN traffic which is forwarded across the RPR via transpar-

ent bridges. Transparent bridge stations forward traffic between the RPR and their other associated LAN networks.

The RPR MAC entity appears as a single interface to the 802 bridging relay. This means the ring media and the collection of stations which attach to the ring appears to the 802 bridge relay as a single loop-free broadcast media. Since the RPR behaves as a loop-free broadcast medium, the spanning tree protocol is not required on the RPR when the ring attaches to a loop-free network. The spanning tree protocol is required over an RPR when multiple 802 networks attached to multiple RPR stations form loops through the RPR network. The RPR MAC entity provides LLC services to support the bridge protocol entity and other higher layer protocol users.

The RPR MAC ensures that a frame is delivered to the intended RPR station(s). RPR MAC procedures ensure that duplicate or misordered frames are not transferred to the 802.17 internal sublayer service (ISS). This includes scenarios where the ring is in a normal operating configuration, or frames are being wrapped or steered during a ring failure

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 MAC relay or intended MAC client at the RPR MAC service interface. The mapping function performed by the RPR MAC service interface shall conform to the interface between a MAC entity and MAC relay, and preserve the filtering services and other requirements for bridged LANs as specified in ISO/IEC 10038 (IEEE Std 802.1D-1998), and IEEE Std 802.1Q-1998. These services include:

1) Maintaining the bridge architecture;
2) Maintaining the nature of filtering services in bridged LANs;
3) Maintaining the extensions specified by IEEE Std 802.1Q-1998 to allow MAC bridges to support the definition and management of virtual LANS (VLANs);
4) Maintaining the provision of filtering services that support the dynamic definition and establishment of groups in a LAN environment, and the filtering of frames by Bridges such that frames addressed to a given group are forwarded only on those LAN segments that are required in order to reach the members of that group;
5) Supporting the registration protocol that is required in order to provide dynamic multicast filtering services;
6) Supporting management services and operations that are required in order to support administration of dynamic multicast filtering services;
7) Maintaining the provision of expedited traffic capabilities, to support the transmission of time-critical information in a LAN environment;
8) Maintaining the concept of traffic classes and the effect on the operation of the forwarding process of supporting multiple traffic classes in bridges;
9) Maintaining the spanning tree algorithm and protocol;
10) Maintaining the generic attribute registration protocol (GARP);
11) Maintaining the GARP multicast registration protocol (GMRP);

Traffic transmitted on the RPR may originate from end stations that are either local or remote to the RPR. Locally sourced traffic is sourced by local end stations. Remotely sourced traffic is sourced by remote end stations which do not directly attach to the ring, but whose traffic is bridged onto the ring by a transparent bridge. Similarly, traffic on the RPR may either be terminated by one or more local end stations, one or more remote stations (via attached bridges), or by a combination of local and remote end stations (via attached bridges). RPR traffic which is either sourced or terminated by a remote end station must pass through a bridge which is attached to the ring. The following bridged network scenarios are supported: local end station transmitting to another local end station, local end station transmitting to a remote end station (via single RPR attached bridge), remote end station transmitting to a local end station (via a single RPR attached bridge), or remote end station transmitting to a remote end station (via a pair of RPR attached bridges).

The bridging conformance model ensures the integrity of RPR attached bridges and their associated bridged networks are maintained for the various network scenarios involving bridges and end stations attached the ring. Bridge learning and forwarding procedures are compromised in scenarios where a bridge sees unicast traffic on the RPR that is destined to a specific MAC address but never sees traffic sourced from this MAC address. In this case, the particular MAC address is never learned, and anytime a bridge sees traffic destined to this MAC address is flooded. This can happen on an RPR with asymmetric traffic flows between a pair of local end stations on the ring. This also happens in the case where traffic originated from a remote end station is destined to a local end station on the ring (via a bridge) and the local end station strips the frame prior to other bridges on the ring learning the remote end station's source MAC address. When the local end station transmits back to the remote end station (via the bridge), all intermediate bridges that did not originally learn the remote end stations source MAC address will persistently flood to all their adjoining networks. The RPR MAC ensures that all bridged traffic is always flooded on the ring, such that all bridges attached to the ring are able to learn MAC addresses properly.

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 RPR to achieve bandwidth multiplication through spatial reuse. Traffic originating from a bridge is always flooded on the ring. Traffic originating from a local end station may or may not be flooded depending on whether the destination is a remote end station or a local end station. The RPR MAC shall support two operational modes in support of the basic bridging model. For bridge stations, the MAC supports *basic_bridging_mode*, in which the MAC always floods data traffic which it transmits onto the ring. End stations support *host_mode*. In *host_mode*, the MAC first checks whether the *MAC_DA* of each transmitted frame is in the topology database (local to the ring). If the MAC_DA of the transmitted frame is in the topology database, the frame is not flooded on the ring and destination stripped by the intended local end station. If the MAC_DA of the transmitted frame is not in the topology database (remote end station), the frame is flooded on the ring. The above cases only relate to frames originating from the MAC client that are being transmitted onto the ring. The above rules do not apply to frames being transmitted through the transit path.

Since local end stations typically strip frames having MAC_DA which matches their station address, a flooding indicator in the frame is used by local end stations to distinguish between frames which are to be destination stripped and frames which are flooded. Frames marked with the flooding indicator shall continue on the ring, until the flooding completes. The flooding indicator is identified in the frame by frames having a flood type of flood.

The RPR 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. While 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 alternate 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. Special procedures are required to preserve ordering with changing of network paths during protection and restoration events.

The RPR MAC conformance to the IEEE Std 802.1D-1998 and Std 802.1Q-1998 bridging standards can be achieved with the following MAC capabilities:

1) The MAC must provide an Internal Sub-Layer Service (ISS), which is used to interface with the Bridging Relay Entity. The ISS will conform to 6.4 of IEEE Std 802.1D-1998 and 7.1 of IEEE Std 802.1Q-1998, when appropriate.
2) The MAC must be able to communicate with the Bridge Protocol Entity via the LLC sub-layer, in conformance with the Bridging specifications.

3) The MAC Service will support a mode that guarantees no reordering with a given user priority for a given combination of destination address and source address to support 802 networks requiring no reordering.[1]
4) The MAC Service will support a mode that guarantees no duplication with a given user priority for a given combination of destination address and source address to support 802 networks requiring no duplication[2]
5) The MAC must be able to transmit bridged frames appropriately over the RPR shared media. This includes handling of unknown unicast, broadcast, and multicast frames.
6) The MAC transmission/reception rules shall allow end stations and bridges to coexist on the ring.
7) The MAC shall copy all frames having the flooding indicator set to its bridging relay client.

E.2 Architectural model of an IEEE Std 802.1D compliant RPR bridge

The RPR MAC conforms to the architectural model of a bridge as defined by IEEE 802.1D-1998. The component LANs are interconnected by means of MAC bridges; each port of a MAC bridge connects to a single LAN. Figure E.2 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.

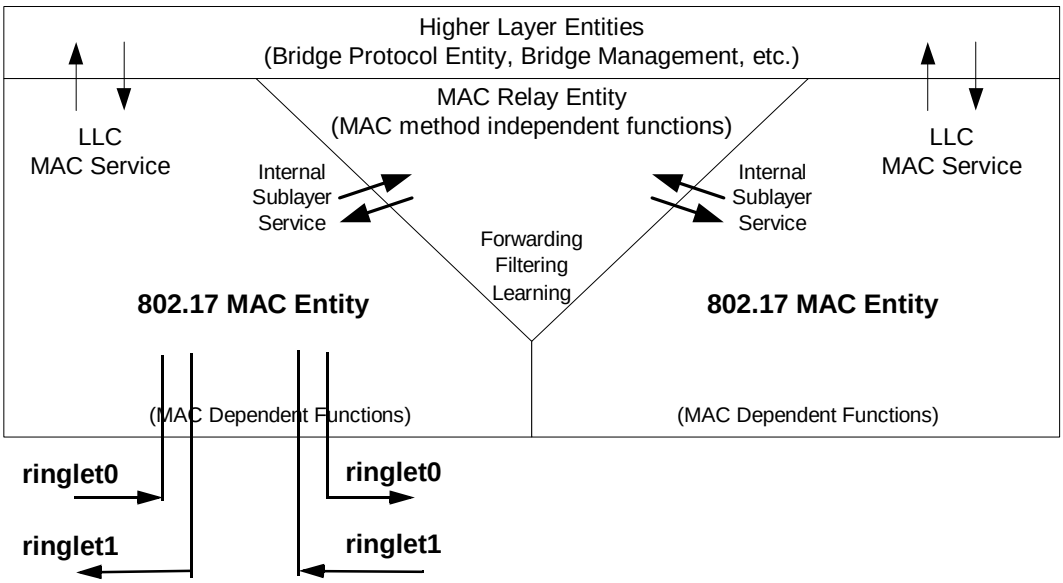


Figure E.2—Bridge architecture model

A bridge consists of:

1) A MAC relay entity that interconnects the bridge's ports;
2) At least two ports;
3) Higher layer entities, including at least a bridge protocol entity.

[1]The MAC Service (9.2 of ISO/IEC 15802-1) permits a negligible rate of reordering of frames (6.3.3 of IEEE Std 802.1D as amended by IEEE Std 802.1W:2001).

[2]The MAC Service (9.2 of ISO/IEC 15802-1) permits a negligible rate of duplication of frames (6.3.4 of IEEE Std 802.1D as amended by IEEE Std 802.1W:2001).

E.2.1 MAC relay entity

The MAC relay entity handles the MAC method independent functions of relaying frames between 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.

E.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 the MAC method dependent functions (MAC protocol and procedures).

E.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 (7.1.3 of IEEE Std 802.1D-1998) and GARP application entities including GARP participants (Clause 12 of IEEE Std 802.1D-1998), make use of logical link control procedures. These procedures are provided separately for each port, and use the MAC service provided by the individual MAC Entities.

Editors' Notes: *To be removed prior to final publication.*

The following subclause, F.3 will be incorporated into maintenance revision of 802.1D subclause 6.5 (Support of Internal Sublayer Service by specific MAC procedures - support by IEEE Std. 802.17) (and other related clauses), if an 802.1 maintenance PAR to include the 802.17 MAC sublayer service is approved.

The text associated with subclause E.3 will be incorporated into 802.17a if the PAR is approved.

E.3 RPR MAC internal sub-layer service

The ISS is provided by a MAC Entity to communicate with the MAC Relay Entity. The interface for this sub-layer is predefined in IEEE Std 802.1D-1998 and IEEE Std 802.1Q-1998. The RPR MAC will adhere to these specifications.

E.3.1 RPR MAC support of internal sub-layer service

The RPR MAC access method is specified in the draft. Clause 8 specifies the MAC frame structure, and Clause 6 specifies the MAC access method.

On receipt of a MA_UNITDATA.request primitive, the local MAC Entity performs Transmit Data Encapsulation, assembling a MAC frame using the parameters supplied as specified below.

On receipt of a MAC frame by Receive Medium Access Management, the frame is passed to the Reconciliation sub-layer which disassembles the MAC frame into parameters, as specified below, that are supplied with the MA_UNITDATA.indication primitive.

The **frame_type** parameter takes only the value *user_data_frame* and shall be encoded in the FT field of the header field.

The **mac_action** parameter takes only the value *request_with_no_response* and is not explicitly encoded in MAC frames.

The **destination_address** parameter is either the address of an individual MAC entity (end station) or a group of MAC entities. The destination_address parameter is the MAC address of the intended destination entity, independent of whether the entity is local or non-local (bridged) to the ring.

The **source_address** parameter is the individual address of the source MAC entity (end station). The source_address parameter is the MAC address of the originating entity, independent of whether the source entity is local or non-local (bridged) to the ring.

The **mac_service_data_unit** parameter is the service user data and shall be encoded in the data field.

The **user_priority** parameter provided in a data request primitive shall be encoded in the service class bits of the RPR control field of the MAC frame in accordance with user priority request and MAC service class columns of Table E.2. The user_priority parameter provided in the data indication primitive takes the value of the service class bits of the RPR control field of the MAC frame in accordance with the user priority indication and MAC service class columns of Table E.1.

Editors' Notes: *To be removed prior to final publication.*

The following MAC user and access priority to service class mapping tables should be located in Clause 5 or Clause 6.

The **access_priority** parameter in an M_UNITDATA.request primitive is mapped to the service class bits of the RPR control field of the MAC frame in accordance with the access priority request and MAC service class columns shown in Table E.2. The mapping of user_priority to outbound access_priority is achieved via fixed, MAC method-specific mappings. The access_priority parameter in an M_UNITDATA.request primitive (6.4 of 802.1D) shall be determined from the user_priority in accordance with the values shown in Table E.2 for the MAC method that will receive the data request. The values shown in Table E.2 are not modifiable by management or other means.

The **ringletID** parameter is an optional request parameter selecting the ringlet on which the frame is to be transmitted. If the **ringletID** parameter is omitted, the MAC shall use its default algorithm in 6.3 to determine which ringlet to use. The **ringletID** parameter is encoded in the MAC frame as defined in 8.2.2.1. The **ringletID** parameter is also provided in the indication primitive indicating which ringlet the frame was received from as defined in 8.2.2.1. The **ringletID** receive parameter may be ignored by the MAC client.

The **MACProtection** parameter is an optional request parameter requesting protection service for the transmitted frame as defined in 5.3.1. If the **MACProtection** parameter is omitted, the MAC shall provide protection for the frame as defined in 5.3.1.

The **markFE** parameter is an optional request parameter requesting classB priority frames to be treated as fairness eligible as defined in Clause 6. If the **markFE** parameter is omitted, the MAC shall apply default frame handling as defined in Clause 6.

The **receptionStatus** parameter is an optional indication parameter providing MAC frame reception status to the MAC client entity as defined in 5.3.2. The **receptionStatus** parameter may be ignored by the MAC client.

The **fairnessEligible** parameter is an optional indication parameter indicating the setting of the fairnessEligible bit in the frame header as described in 8.2.2.2. The **fairnessEligible** parameter may be ignored by the MAC client.

The **frame_check_sequence** parameter is encoded in the FCS field of the MAC frame. The FCS is computed as a 32-bit CRC beginning with the first byte following the header checksum to the end of the frame as described in 8.3.9. If a MA_UNITDATA.request primitive is not accompanied by this parameter, it is calculated in accordance with 8.2.10 of this draft.

Figure E.3 shows the mapping of the MA-UNITDATA.request primitive parameters to the RPR MAC frame fields, and the mapping of the RPR MAC frame fields to the MA-UNITDATA.indication primitive parameters.

Table E.1—MAC service class to User Priority indication mapping

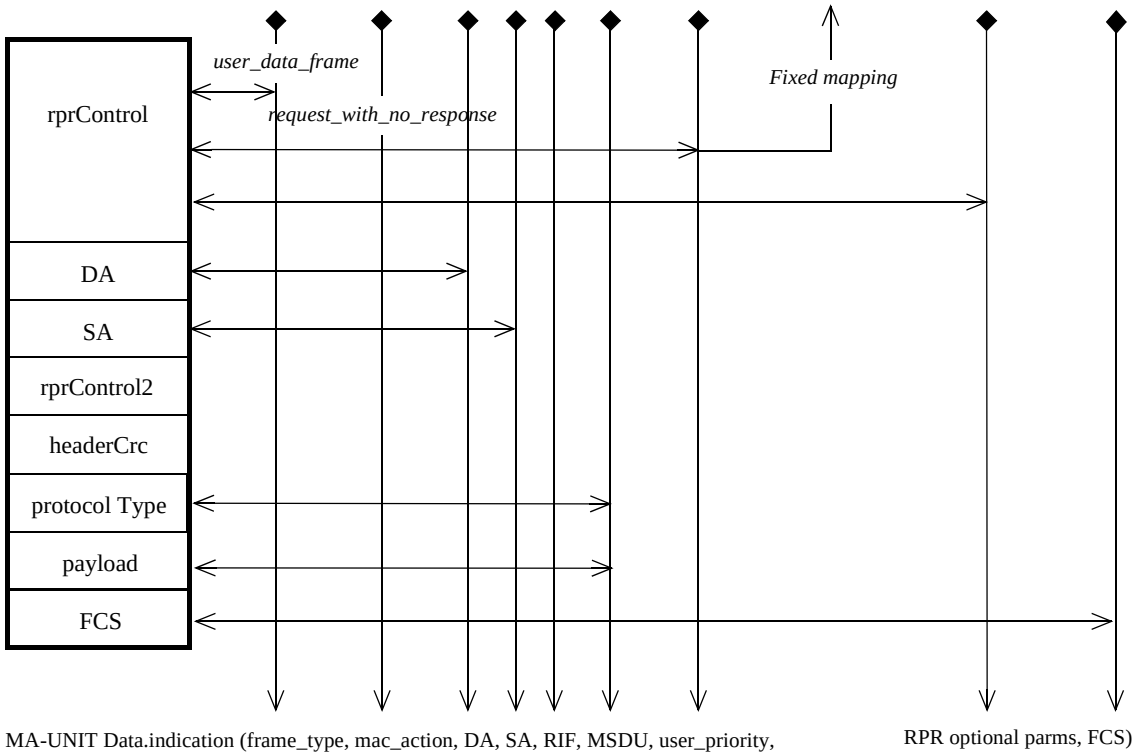
MAC Service Class	User Priority (M_UNITDATA. indication)
classC	0
classB	4
classA	6

Table E.2—Outbound access priorities

User Priority	Outbound Access Priority per MAC method									
	802.3	802.17	8802-4	8802-5 (default)	8802-5 (alternate)	8802-6	802.9a	8802.11	8802-12	FDDI
0	0	classC	0	0	4	0	0	0	0	0
1	0	classC	1	1	4	1	0	0	0	1
2	0	classC	2	2	4	2	0	0	0	2
3	0	classC	3	3	4	3	0	0	0	3
4	0	classB	4	4	4	4	0	0	4	4
5	0	classB	5	5	5	5	0	0	4	5
6	0	classA	6	6	6	6	0	0	4	6
7	0	classA	7	6	6	7	0	0	4	6

Editors' Notes: *To be removed prior to final publication.*

In the Table E.1 and Table E.2, we need to check whether any user priority should be mapped to classA.



Note - “RPR optional parms” in above primitives expand to the following signals:
RPR optional request parameters - (ringletID, MACprotection, markFE)
RPR optional indication parameters - (ringletID, receptionStatus, fairnessEligible)

Figure E.3—Mapping of MAC service primitives

E.3.2 RPR MAC support of enhanced internal sub-layer service

An Enhanced Internal Sub-Layer Service (E-ISS) is derived from the Internal Sub-Layer Service (ISS, defined in ISO/IEC 15802-3, 6.4) by augmenting that specification with elements necessary to the operation of the tagging and un-tagging functions of the MAC bridge. The E-ISS provided by the MAC will conform to 7.1 of IEEE Std 802.1Q-1998.

Figure E.4 and Figure E.5 shows the mapping of the EM-UNITDATA.request primitive parameters to the MAC frame fields, and the mapping of the MAC frame fields to the EM-UNITDATA.indication primitive parameters.

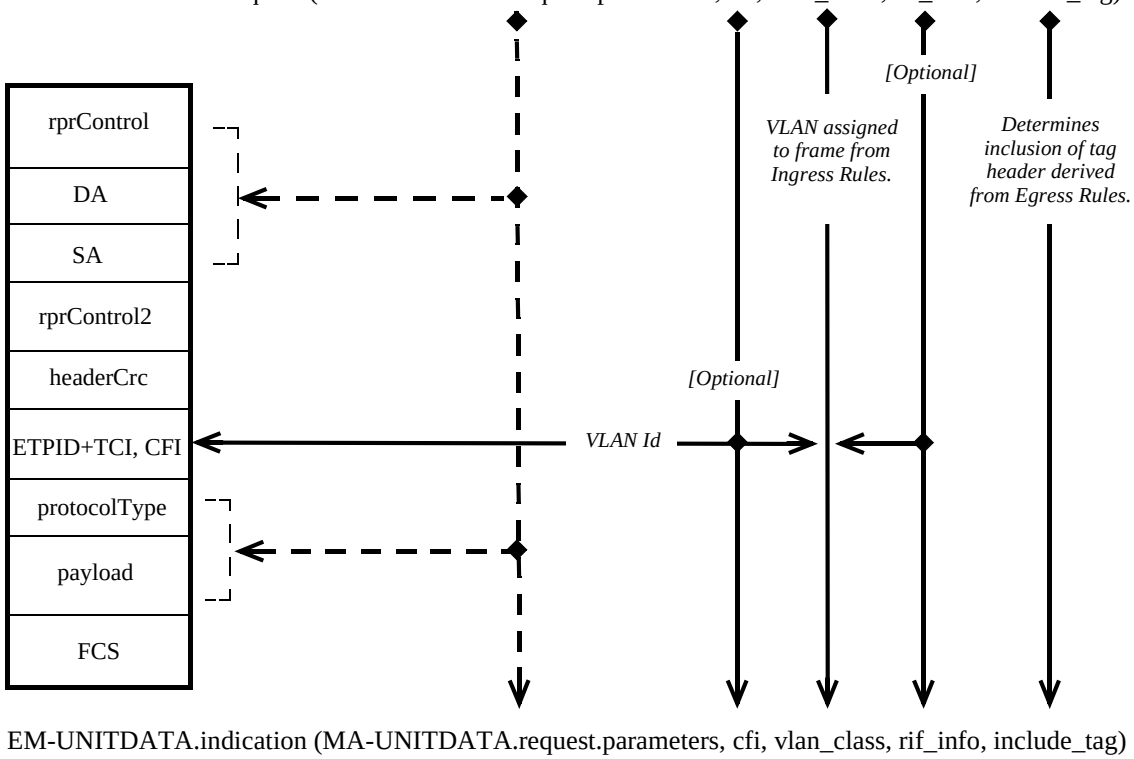


Figure E.4—Mapping of enhanced MAC service primitives

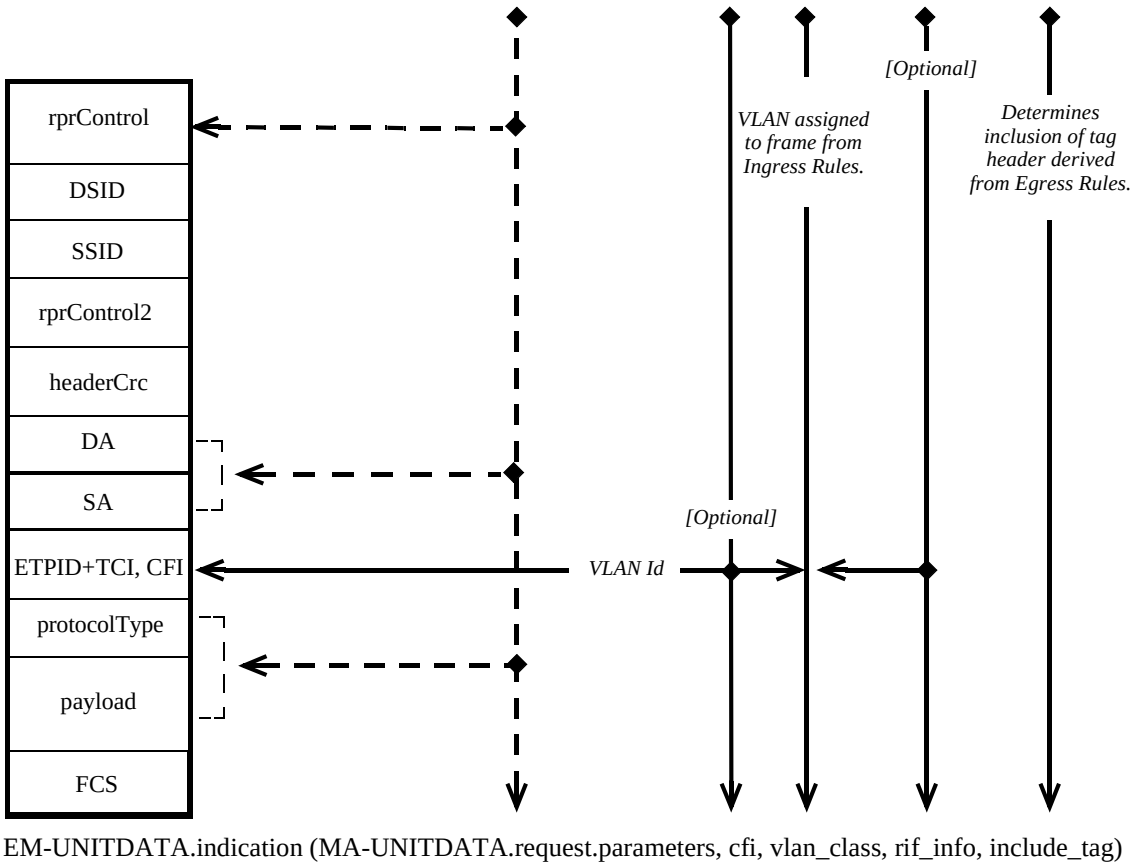


Figure E.5—Mapping of enhanced MAC service primitives using extended frame format

Editors' Notes: *To be removed prior to final publication.*

Figure E.4 and Figure E.5 illustrates two RPR frame formats that include a Q-tag. The WG needs to decide which format will be supported. I think that the mapping shown in Figure E.5 should be accepted primarily because the pre headerCRC field is fixed. Also, need to include an extended frame mapping of strict mode bridged frames.

E.4 Bridge protocol entity interactions

The RPR MAC shall provide a MAC sub-layer that conforms to 2.2 of IEEE Std 802.2-1998.

E.5 RPR MAC reception of data frames

The RPR MAC reception rules are dependent on whether the MAC is configured for host mode or bridge mode. An RPR MAC with a host client is configured for host mode. An RPR MAC with a bridge relay client is configured for bridge mode. The primary difference in reception rules between host mode and bridge

mode, is that in bridge mode all frames indicating *flood_type* of flood are transferred to the MAC client. In host mode, only those frames matching the host MAC address, broadcast address, and provisioned set of multicast addresses are transferred to the MAC client. The host mode option provides a MAC filtering function inhibiting unicast frames destined to other stations from being received and needing to be processed by the host client.

Note - Host/bridge MAC modes are internal configurations of the MAC and not user provisionable via LME.

More detailed description of general reception processing including reception of control and fairness frames is defined in 6.8.

Receive frames are either:

a) Copied/Stripped. The frame is stripped from the Ring and passed to the MAC client (bridge relay or bridge protocol entity).
b) Discarded. The frame is stripped from the Ring and not passed to the MAC client. The frame is discarded.
c) Transit. The frame is passed through the transit path and transmitted on the outgoing ringlet. The frame is not passed to the MAC client (bridge relay or bridge protocol entity).
d) Copied. The frame is passed to the MAC client (bridge relay or bridge protocol entity), and also passed through the transit path and transmitted on the outgoing ringlet.

Refer to Clause 6 for a complete description of the RPR MAC transit path.

Editors' Notes: *To be removed prior to final publication.*

Possible MAC features to add/consider in conjunction with host/bridge MAC modes: flooded frame statistics? report MAC is in this mode?, granularity of the modes

E.5.1 Host Mode data frame reception rules

The MAC shall support the following set of reception rules when supporting a host MAC client:[3]

1) Received frames having a *flood_type* = *flood*, and the DA of the received frame is the RPR station's MAC address (or broadcast or multicast address), the frame is copied from the ring and is not destination stripped from the ring by the station. The frame is to remain on the ring until the flooding procedures complete. The frame will be stripped upon completion of the flooding procedure based on the flood stripping rules.
2) Received frames having a *flood_type* = *flood*, and the DA of the received frame is **not** the RPR station's MAC address (or broadcast or multicast address), the frame is not copied from the ring, and remains on the ring until the flooding procedures complete. The frame will be stripped upon completion of the flooding procedure based on the flood stripping rules.
3) Received frames having a *flood_type* = *no_flood*, and the DA of the received frame is the RPR station's MAC address, the frame is copied/stripped from the ring. The frame must be destination stripped by the station.
4) Received frames having a *flood_type* = *no_flood*, and the DA of the received frame is **not** the RPR station's MAC address (or multicast/broadcast address), the frame is neither copied or stripped from the ring. The frame shall continue on the ring until it is destination stripped by a station whose MAC address matches the frame's DA or is discarded upon expiration of TTL.

[3]Note - Flood_type of *unidirectional_flood* or *bidirection_flood* indicates a *flood*. Flood_type is *no_flood* if it is neither *undirectional_flood* or *bidirectional_flood*.

E.5.2 Bridge Mode data frame reception rules

The MAC shall support the following set of reception rules when supporting a bridge MAC client:

1) Received frames having a *flood_type = flood*, the frame is always copied from the ring to its bridge relay client. The frame is also stripped once the flooding procedure completes.
2) Received frames having a *flood_type = flood*, and the DA of the received frame is that of the RPR station, the frame is copied/stripped and passed to the MAC client.
3) Received frames having a *flood_type = no_flood*, and the DA of the received frame is not that of the RPR station, the frame is not copied or passed to the MAC client. The frame only passes through the transit path of the MAC.

E.6 RPR MAC transmission of data frames

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 E.3 and Figure E.4 for the mappings.

The RPR MAC transmission rules are dependent on whether the MAC is configured for host mode or bridge mode. An RPR MAC with a host client is configured for host mode. An RPR MAC with a bridge relay client is configured for bridge mode. The MAC sets *flood_type* in the RPR frame for data frames based on the associated set of host or bridge mode transmission rules. The *flood_type* is ignored for control and fairness frames.

Note - Host/bridge MAC modes are internal configurations of the MAC and not user provisionable via LME.

More detailed description of general transmission processing including transmission of control and fairness frames is defined in 6.9.

1) Bridged frames with a Multicast and Broadcast destination address are flooded over the RPR.
2) Bridged frames with a destination address of a station on the RPR (i.e., a known destination address) is forwarded to the station using internal MAC topology and steering tables.
3) Bridged frames with an unknown destination address (e.g., a destination address not matching a RPR station address) are flooded over the RPR.

E.6.1 Host Mode data frame transmission rules

The MAC shall support the following set of transmission rules when supporting a host MAC client:

1) If DA is in the topology database, the frame is transmitted with the *flood_type = flood*.

2) If DA is not in the topology database, the frame is transmitted with the *flood_type = no_flood*.

E.6.2 Bridge Mode data frame transmission rules

The MAC shall support the following set of transmission rules when supporting a bridge MAC client:

1) The MAC shall always transmit frames with *flood_type = flood* for all data frames transmitted on the ring.

E.7 Flooding frame over RPR

Transparent bridging requires a form of one-to-others distribution called flooding. Flooding frames over an RPR requires extra information above and beyond the *sourceMACAddress* and *destinationMACAddress*. The additional information assist in scoping the range of the flooding distribution and suppressing undesirable duplicates that might otherwise be generated.

Bridges flood remote frames for delivery to all bridge clients, as illustrated in the left side of Figure E.6. Flooding involves transmissions between a single source station and all other stations.

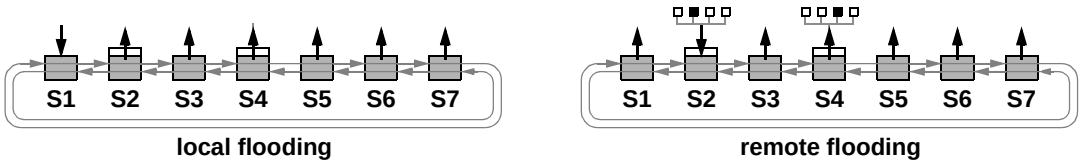


Figure E.6—Basic bridge flooding

Basic-bridging stations maintain simplicity by always flooding, as illustrated in Figure E.6. Although no spatial reuse is possible, this avoids overheads associated with maintaining and utilizing RPR forwarding tables.

Enhanced-bridging stations improve efficiencies by maintaining and utilizing RPR forwarding tables, so that remote frames can also be unicast, as illustrated in Figure E.7. The learn improves link utilization, due to the frame's unicast (not flooded) and shortest-path nature.

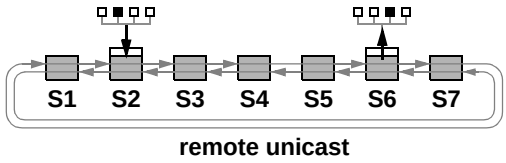


Figure E.7—Enhanced bridging unicasts

Ordering constraints mandate that flooded and related remote-unicast transfers flow over the same path. The term related refers to frames with an identical set of *{sourceMacAddress/destinationMacAddress, VLAN}* identifiers. Flowing over the same path is necessary to maintain ordering, without invoking an inefficient flush between floods and related remote-unicast transfers.

For unidirectional flooding, the potential performance impact of this ordering constraint can be severe, in that the worst case path-length nearly doubles over that associated with bidirectional flooding. To avoid that potential performance impact, enhanced bridges are expected to support bidirectional flooding. Alternatively, they can also support flushing before a frame transmission direction change.

Annex F

(normative)

CRC calculations

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	Revision for RPR TF review.
Draft 0.3, June 2002	Revised draft for WG review.
Draft 1.0, August 2002	Revised draft for TF review.
Draft 1.1, October 2002	Revised draft for WG review.
Draft 2.0, December 2002	Revised draft for TF review and WG ballot.

F.1 Cyclic redundancy check 16-bit (CRC16) algorithmic definition

There is a 16-bit check symbol at the end of the packet header. The CRC uses the generator polynomial of equation F.1. Which is performed on the most significant bits first:

$$x^{16}+x^{12}+x^{5}+1 \quad \text{(F.1)}$$

F.1.1 Serial CRC16 calculation

Editors' Notes: *To be removed prior to final publication.*

The bit-selection order was unambiguous in D1.0, so the editor selected an MSB-to-LSB ordering, thought to be more consistent with SONET usage. Readers are encouraged to reflect and comment on this ordering.

The serial implementation of the CRC-16 polynomial, as applied to the most- through least-significant bits, is specified by the C-code calculations of Table F.1 and the hardware implementation illustrated in Figure F.1. The CRC calculation are conceptually computed in a byte sequential fashion; for each byte, the checksum is updated 8 times, once for each bit. When computing the checksum on a per-bit basis, the first-through-last of the data bits are the most-through-least significant bits of the byte respectively.

Table F.1—Serial CRC-16 computations

```
// c00-through-c15 are the most- through least-significant bits of check.
// 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;
```

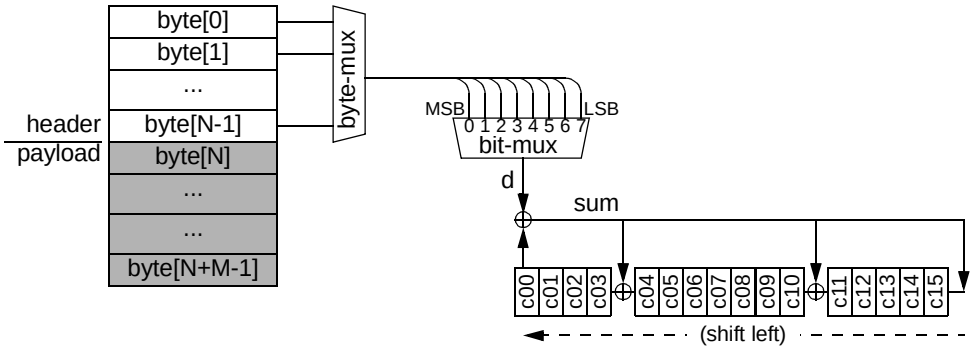


Figure F.1—Serial Crc16 reference model

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 below:

a) Startup. The CRC16 calculations start with an all-zeros *crcSum* value.
b) Completion. The computed CRC16 value is complemented before being appended to the packet.

F.1.2 Exchanged CRC16 calculations

The RPR CRC16 is based on the assumption that bits are sent in a least- through most-significant bit ordering, as described in the remainder of this subclause.

The generation and checking of 16-bit CRC values, optimized for bit-reversed transmission, is illustrated in Figure F.2. The bit-swap operations involve no circuitry and can be incorporated within the combining-EXOR circuitry.

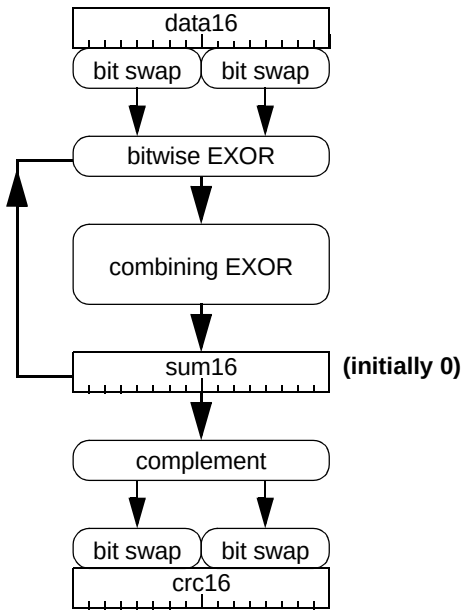


Figure F.2—Exchanged CRC16 calculations

The CRC-generation code of F.1.1 can be called to generate CRC-computation tables, using the C program documented in Annex H. This program supports the creation of tables for performing parallel CRC checks, where 1, 2, 4, 8 data bits are processed in parallel. Computer generation of the CRC-table text, rather than their values, minimized the possibility of introducing errors in the documentation process.

Editors' Notes: *To be removed prior to final publication.*

The referenced C code for CRC16 has not yet been produced.

F.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 F.1. Which is performed on the most significant bits first:

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1 \quad \text{(F.1)}$$

F.2.1 Serial CRC32 calculation

The serial implementation of the CRC-32 polynomial, as applied to the most- through least-significant bits, is specified by the C-code calculations of Table F.2 and the hardware implementation illustrated in Figure F.3.

Table F.2—Serial CRC-32 computations

```
// c00-through-c31 are the most- through least-significant bits of check.
// 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;         d23= c24^sum;
c24= c25^sum;     c25= c26;         c26= c27^sum;     c27= c28^sum;
c28= c29;         c29= c30^sum;     c30= c31^sum;     c31= sum;
```

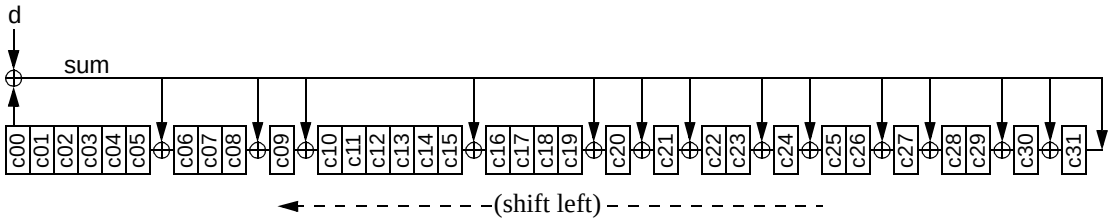


Figure F.3—Serial Crc32 reference model

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:

a) Startup. The CRC calculations start with an all-ones crcSum value.
b) Completion. The computed CRC value is complemented before being appended to the packet.

F.2.2 Exchanged ExorSum calculations

The RPR CRC is based on the assumption that bits are sent in a least- through most-significant bit ordering, as described in the remainder of this subclause.

The generation and checking of 32-bit CRC values, optimized for bit-reversed transmission, is illustrated in Figure F.4. The bit-swap operations involve no circuitry and can be incorporated within the combining-EXOR circuitry.

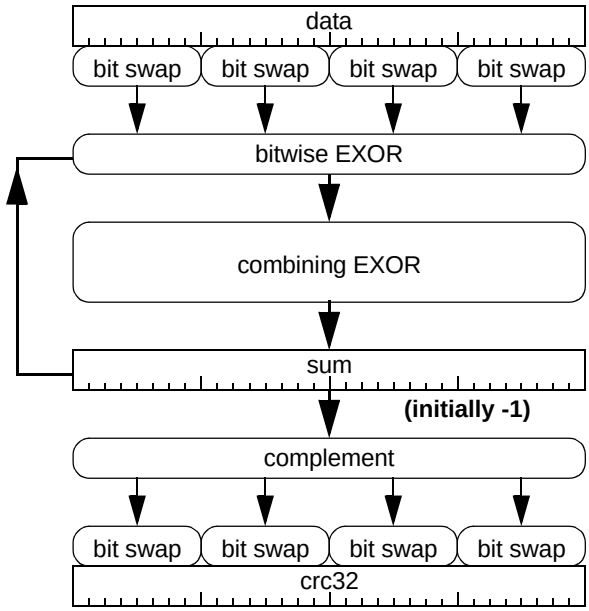


Figure F.4—Exchanged EXORSum calculations

The CRC-generation code of F.2.1 can be called to generate CRC-computation tables, using the C program documented in Annex H. This program supports the creation of tables for performing parallel CRC checks, where 1, 2, 4, 8, 16, or 32 data bits are processed in parallel. Computer generation of the CRC-table text, rather than their values, minimized the possibility of introducing errors in the documentation process.

F.2.3 Data CRC stomping

Cut-through frame processing allows frame payload retransmissions to begin before the frame's CRC has been verified. Store-and-forward processing may confirm a valid header-CRC but detect an invalid data-CRC. In both cases, the verified header continues circulating and the payload is 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 F.5.

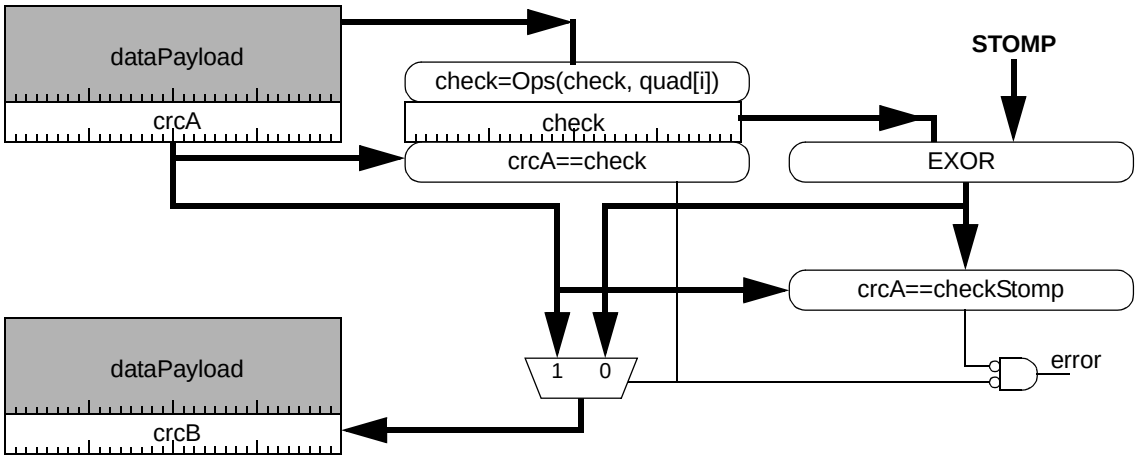


Figure F.5—Data CRC stomping

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 $FFFFFFFF_{16}$ 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 un-errored frame.

F.2.4 Protected time-to-live adjustments

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 *timeToLive* field involves adjusting the following CRC field, as illustrated in Figure F.6.

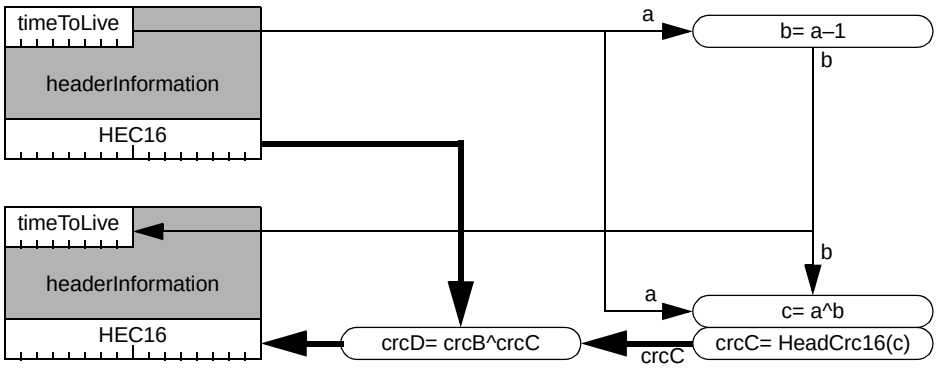


Figure F.6—Protected time-to-live adjustments

An incremental update of the CRC shall be used to maintain CRC coverage when the *timeToLive* field is adjusted. This involves computing the new *timeToLive* field value *b* and the difference *c* between new and old *timeToLive* values. The difference value *c* generates an incremental CRC value *crcC*, which is EXORed with the old *crcB* value to generate the new *crcD* value. The data is never left unprotected: an error in *crcB* or the computation of internal CRC values will (nearly) always be reflected as an error in check value *crcD*.

F.3 Example CRC calculations

Editors' Notes: *To be removed prior to final publication.*

Per comment 681 against D1.0, this (informative) subclause will present an example data frame and show the resultant headerCRC and payloadCRC values.

Annex G

(informative)

C-code illustrations

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for WG review.
Draft 0.2, April 2002	Draft 0.2 for TF review.
Draft 0.3, June 2002	Draft 0.3 for WG review, modified per comments on D0.2.
Draft 1.0, August 2002	Draft 1.0 for TF review, modified per comments on D0.3.
Draft 1.1, October 2002	Draft 1.1 for WG review, modified per comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified per comments on D1.1.

This Annex provides code examples that illustrate the computation of the CRC, the MAC data path, and the RPR fairness algorithm. The code in this Annex is purely for informational purposes, and should not be construed as mandating any particular implementation. In the event of a conflict between the contents of this Annex and another normative portion of this standard, the other normative portion shall take precedence.

Editors' Notes: *To be removed prior to final publication.*

Per comment 338 against D1.1, this annex will be updated at some point in the future to reflect the current state of the standard, especially with regards to Clause 9. Help in doing so is solicited.

The syntax used for the following code examples conforms to ISO/IEC 9899:1999.

Editors' Notes: *To be removed prior to final publication.*

The web coordinator will be requested to post the most recent C-code, as this is provided.
While the target of two weeks per code update will be attempted, no exact schedule can be guaranteed.

```
// ********************************************************************************************************************************************
// ********************************************************************************************************************************************
// ***************************************** Coding Style - V0.30 dated: 2002Jun01 ************************************************************
// ********************************************************************************************************************************************
// ********************************************************************************************************************************************
//                                                                                                                  1         1         1         1
//       1         2         3         4         5         6         7         8         9         0         1         2         3         4
// 3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901
//
// ********************************************************************************************************************************************
// ********************************************************** COMMENTING **********************************************************************
// ********************************************************************************************************************************************
// The "//" commenting convention shall be used.
// An in-line comment shall be properly indented.
// 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;
//
// ********************************************************************************************************************************************
// ************************************************** NAMING CONVENTIONS **********************************************************************
// ********************************************************************************************************************************************
// The code shall use these conventions for symbol name formats:
//   MY_DEFINE    - All caps and underscores between words
//                  Used when naming defines and enum values
//
//   MyType       - First letter is upper case; each run-together word is capitalized
//   MyRoutine()  - Used when naming typedefs and subroutine names
//
//   myData       - First letter is lower case; each run-together word is capitalized
//   myField      - Used when naming variable names, and data-structure fields.
//
// Don’t use abbreviations that are unclear to others and save only a few keystrokes.
// Examples follow:
//
// | #define MY_DEFINES
// | enum values {
// |  STATUS_GOOD,
// |  STATUS_POOR,
// |  STATUS_FATAL
// | };
//
// | typedef struct {
// |   int hiPart;
// |   int loPart;
// | } PairOfInts;
//
// | uInt4 myData;
// | sInt4 MyRoutine();
```

```
// ********************************************************************************************************************************
// ************************************************** INDENTATION STYLE ***********************************************************
// ********************************************************************************************************************************
// Each level of code uses a 4-space indentation.
// Initial "{" braces shall appear at end of line. Final "}" braces shall appear at start of line.
//
// If only a single statement is enclosed, then curly braces should not be used. In "if/else" sequences, curly braces
// should be used unless both blocks are single statements. Indentation for common structures is illustrated below.

// | if (condition)
// |   statement1;
// |
// | if (condition)
// |   statement1;
// | else
// |   statement2;
// |
// | if (condition) {
// |   statement1;
// |   statement2;
// | }
// |
// | if (condition) {
// |     statement1;
// |     statement2;
// | } else {
// |     statement3;
// | }
// |
// | if (condition) {
// |     statement1;
// | } else {
// |     statement2;
// |     statement3;
// | }

// | switch (variable) {
// | case VAL2:
// |     statement1;
// |     break;
// | default:
// |     statement2;
// | }

// | while (condition)
// |   statement1;
// |
// | while (condition) {
// |   statement1;
// |   statement2;
// | }
```

```
// | for (n= 0; n < N; n += 1)
// |   statement1;
// |
// | for (n= 0; n < N; n += 1) {
// |   statement1;
// |   statement2;
// | }


// ********************************************************************************************************************************
// ********************************************** C-code programming constraints **************************************************
// ********************************************************************************************************************************
// 1) For clarity, avoid using the following C constructs:
//     x++, y--, --x, and ++y.
//    Post increment or decrement are phrased as x+= 1 or y-= 1.
// 2) For correctness, never suppress compilation warnings. Following guidelines help eliminate spurious lint warnings
//

// ********************************************************************************************************************************
// ************************************************ Inhibiting warning messages ***************************************************
// ********************************************************************************************************************************
// To minimize the number of false lint messages (on a Sun compiler), lint directives were used. It's assumed that a preprocessor
// can change these directives to work with other compilers. The lint directives (which are included as comments before the
// offending line) include CONSTCOND and FALLTHRU which are used as illustrated below:
//
//  | // CONSTCOND
//  | lineWhichUses(!CONSTANT);
//  |
//  | case X:
//  |   initialCode();
//  |   // FALLTHRU
//  | case Y:
//  |   commonSwitchCode();
//  |   break;

//
// Special DEFAULT_ONLY and DEFAULT_LAST macros are used within switch statements, when all of the enumerated values are intended
// to be used. The DEFAULT_ONLY is used when all enumerated values are used. It is typically followed by an ERROR() statement, or an
// ERROR1(arg) statement, to catch run-time errors. The "arg" expression inside of the ERROR1(arg) macro is used to inhibit gcc
// warnings, but ensuring that variables are set before being used, as illustrated below:
//
// | switch(value) {
// | case X:
// |   add = 4;
// |   break;
// | case Y:
// |   add = 5;
// |     break;
// | DEFAULT_ONLY
// |     ERROR1(add = 4);
// | }
// | sum += add;
```

```
// The DEAFULT_LAST macro is used when some of the enumerated value are not used. Like DEFAULT_ONLY, this macro is either
// defaulted to a null statements (for compile-time checking of enumerated value usage) or to a "default:" statement,
// for run-time checking of enumerated values. An example is shown below"
//
// | switch(value) {
// | case X:
// |   ProcessX();
// |     break;
// | case Y:
// |     ProcessY();
// |   break;
// | case Z:
// |   DEFAULT_LAST
// |   ERROR();
// | }

// Depending on the option which is selected, null routines are sometimes included in the compilation, but never called. To eliminate
// lint/gcc warnings, the null routine includes a EXECUTE_ERROR2(a1,a2) macro which uses all of the calling arguments, as
// illustrated below:
//
// | #if DOP_XXXX_NONE
// | uInt2
// | StubRoutine(CommonParameters *a1, int a2)
// | {
// |     EXECUTE_ERROR2(a1, a2);
// |     return((uInt2)0);
// | }
// | #endif
```

```
// Sample routines are included to illustrate how vendor-dependent routines may be constructed. In some cases, the sample routine
// does not use all of its arguments. For such routines, the IS_USED(a) macro is used to eliminate "argument not used" warnings,
// as illustrated below:
//
// | int
// | AddressMatchSample(MemParameters *memPtr, uInt2 this, uInt2 that)
// | {
// |     assert(memPtr != NULL);
// |     IS_USED(that);
// |     return(memPtr->common.nodeId == this);
// | }
//
// ********************************************************************************************************************************
// ********************************************** indent argument values *************(**************************************
// ********************************************************************************************************************************
// The code is formatted using indent (GNU version 1.2) with the following flags set:
//   indent -l132 -c47 -cd35 -i2 -ci1 -di1 -nbad -nbap -ncs -npcs -nfc1 -br -sc -ce -ncdb -nlp
//
// -l132   maximum length of a line is 132 characters
// -c47   code comments start at line position 47
// -cd35  declaration comments start at line position 35
// -i2    indentation level is 2 spaces
// -ci1   continuation lines are indented 1 space
// -di1   put identifiers in first available position after type
// -nbad  no blank line is forced after a declaration
// -nbap  no blank line is forced after a procedure body
// -ncs   no space after a cast operator
// -npcs  no space after a procedure name and '('
// -nfc1  do not format comments in the first column
// -br    put '{' on line with if
// -sc    put '*' at left of comments
// -ce    put else on same line as '}'
// -ncdb  no comment delimiters on blank lines
// -nlp   do not line up continued lines at parentheses
//
// ********************************************************************************************************************************
// ************************************************** compiler tolerance ********************************************************
// ********************************************************************************************************************************
//
// The code is designed to support compilers that are "almost" ANSI compliant, by avoiding features that have been found to cause
// problems:
//
//  1) assert(). ANSI C defines this to behave like a function, returning the void 0 value. Since many compilers use a different
//     convention in their assert.h file, this C code provides its own assert() definition
//     (unless the GNU compiler is being used, which gets this right).
//
//  2) External function declarations. When declaring a function, either within the header or the routine itself, use the
//     typedef names, as illustrated below:
//
// | void
// | DoLocks(Quads2(*LockCompute) (Quads2, Quads2, Quads2), int sub,
// |   int size, Byte * valPtr, Byte * argPtr, Byte * oldPtr, Byte * memPtr)
// | {
// |    // Other DoLocks() code would be located here
// | }
```

```
//
//  3) When the function prototype is used to specify the format
//     within a data structure, then the typedef name should be avoided,
//     as illustrated below:
//
// | typedef struct _ChipInitInfo {
// |
// |   // Other structure entry definitions
// |
// |   void (*VdSetup) (struct _ChipInitInfo *);
// |
// | } ChipInitInfo;

// ************************************************************************************************************************
// ************************************************* Compile-time warnings ************************************************
// ************************************************************************************************************************
// Testing for compile-time errors is performed using gcc, using the command
// string given below. Note that -O2 option is required to find the
// "uninitialized" warnings.
//
//     gcc -O2 -Wall -Wshadow -Wpointer-arith -Wcast-qual -Wcast-align -Wmissing-prototypes -c code.c
//
// Symbol length limitations:
// To test that only 31 significant characters are required for
// internal identifiers, gcc is used with the following options:
//
//     gcc -fsyntax-only -Wid-clash-31 ourCode.c
//
//
// ************************************************************************************************************************
// ************************************************************************************************************************
// ************************************************* 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>
#ifdef __GNUC__
#include <assert.h>
#else
#ifdef 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
```

```
// A null definition for gcc checks; all enum values must be used
#ifdef GCC_CHECK
#define DEFAULT_ONLY                            // NOTREACHED
#define DEFAULT_LAST
#else
#define DEFAULT_ONLY  default:
#define DEFAULT_LAST  default:
#endif

#ifdef PURE_ANSI
int rand(void);                    // ANSI standard random function
#define RAND16()    (uInt2)(((rand()>>4)&0xff)|((rand()<<4)&0xff00))
#define RAND32()    (uInt4)(RAND16()<<16 | RAND16())
#else
long random(void);                 // Unix random() function
#define RAND16()    (uInt2)random()
#define RAND32()    (uInt4)random()
#endif

// This line should never be executed, print message and abort
#define ERROR() ((void)((void)fprintf(stderr, "ERROR() executed: file \"%s\",line %d\n",__FILE__,__LINE__), (void)abort(), 0))

// ************************************************************************************************************************************
// ********************************************** User Modifiable Constants ***********************************************************
// ************************************************************************************************************************************

// This constant allows the user to test whether reserved fields are really ignored.
// If DEBUG is 1, use random-valued reserved fields, to validate they are properly ignored.
// If DEBUG is 0, use zero-valued reserved fields, to conform to the specification.
#define DEBUG 0                            // Tests ignored fields

// ************************************************************************************************************************************
// ********************************************** Other header definitions ************************************************************
// ************************************************************************************************************************************
// The IS_USED() macros eliminates lint/gcc warnings within uncalled stub routines
#define IS_USED1(a)  ((a)=(a))
#define IS_USED2(a,b)  (IS_USED1(a), IS_USED1(b))
#define IS_USED3(a,b,c)  (IS_USED2(a, b), IS_USED1(c))
#define IS_USED4(a,b,c,d)  (IS_USED2(a, b), IS_USED2(c, d))
#define IS_USED5(a,b,c,d,e)  (IS_USED4(a,b,c,d), IS_USED1(e))
#define IS_USED6(a,b,c,d,e,f)  (IS_USED4(a,b,c,d), IS_USED2(e,f))
#define IS_USED7(a,b,c,d,e,f,g)  (IS_USED4(a,b,c,d), IS_USED3(e,f,g))
#define IS_USED8(a,b,c,d,e,f,g,h)  (IS_USED4(a,b,c,d), IS_USED4(e,f,g,h))
#define IS_USED9(a,b,c,d,e,f,g,h,i)  (IS_USED8(a,b,c,d,e,f,g,h), IS_USED1(i))
#define IS_USED10(a,b,c,d,e,f,g,h,i,j)  (IS_USED8(a,b,c,d,e,f,g,h), IS_USED2(i,j))
#define IS_USED11(a,b,c,d,e,f,g,h,i,j,k)  (IS_USED8(a,b,c,d,e,f,g,h), IS_USED3(i,j,k))
#define IS_USED12(a,b,c,d,e,f,g,h,i,j,k,l)  (IS_USED8(a,b,c,d,e,f,g,h), IS_USED4(i,j,k,l))
#define IS_USED13(a,b,c,d,e,f,g,h,i,j,k,l,m)  (IS_USED8(a,b,c,d,e,f,g,h), IS_USED5(i,j,k,l,m))
#define IS_USED14(a,b,c,d,e,f,g,h,i,j,k,l,m,n)  (IS_USED8(a,b,c,d,e,f,g,h), IS_USED6(i,j,k,l,m,n))
#define IS_USED15(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o)  (IS_USED8(a,b,c,d,e,f,g,h), IS_USED7(i,j,k,l,m,n,o))
```

```
#define EXECUTE_ERROR1(a1) ( IS_USED1(a1), ERROR() )
#define EXECUTE_ERROR2(a1,a2) ( IS_USED2(a1,a2), ERROR() )
#define EXECUTE_ERROR3(a1,a2,a3) ( IS_USED3(a1,a2,a3), ERROR() )
#define EXECUTE_ERROR4(a1,a2,a3,a4) ( IS_USED4(a1,a2,a3,a4), ERROR() )
#define EXECUTE_ERROR5(a1,a2,a3,a4,a5) ( IS_USED5(a1,a2,a3,a4,a5), ERROR() )
#define EXECUTE_ERROR6(a1,a2,a3,a4,a5,a6) ( IS_USED6(a1,a2,a3,a4,a5,a6), ERROR() )
#define EXECUTE_ERROR7(a1,a2,a3,a4,a5,a6,a77) ( IS_USED7(a1,a2,a3,a4,a5,a6,a7), ERROR() )
#define EXECUTE_ERROR8(a1,a2,a3,a4,a5,a6,a7,a8) ( IS_USED8(a1,a2,a3,a4,a5,a6,a7,a8), ERROR() )
#define EXECUTE_ERROR9(a1,a2,a3,a4,a5,a6,a7,a8,a99) ( IS_USED9(a1,a2,a3,a4,a5,a6,a7,a8,a9), ERROR() )
#define EXECUTE_ERROR10(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10) ( IS_USED10(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10), ERROR() )
#define EXECUTE_ERROR11(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11) ( IS_USED11(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11), ERROR() )

// RESERVED_SET() is used to keep lint quiet
#define RESERVED_SET()  ((int)(DEBUG ? RAND16() : 0))

typedef unsigned char uInt1;      // assuming 'char' is an 8-bit value
typedef unsigned short uInt2;     // assuming 'short' is a 16-bit value
typedef unsigned long uInt4;      // assuming 'long' is a 32-bit value
typedef unsigned long long uInt8; // assuming 'long long' is a 64-bit value
typedef signed char sInt1;        // assuming 'char' is an 8-bit value
typedef signed long sInt2;        // assuming 'short' is a 16-bit value
typedef signed long sInt4;        // assuming 'long' is a 32-bit value
typedef signed long long sInt8;   // assuming 'long long' is a 64-bit value


// ********************************************************************************************************************************
// ********************************************************************************************************************************
// ********************************************** CRC-32 Generation Code **********************************************************
// ********************************************************************************************************************************
// ********************************************************************************************************************************
void PrintTable(int, int);
int ValidateCrc(int, uInt4 *, int);
uInt4 GenerateCrc(int, uInt4 *, int);
uInt4 CalculateCrc(int, uInt4 *, int);
uInt4 CrcBits(uInt4, uInt4, int);
uInt4 BitReverse(uInt4);
void Error(char *);

#define MSB32  ((unsigned)1<<31)
#define ONES32 0XFFFFFFFF
#define CRC_COMPUTE ((uInt4)0X04C11DB7)
#define CRC_RESULTS ((uInt4)0XC704DD7B)

int
main(int argc, char **argv)
{
    int i;
    int size = 32, reverse = 1, table = 1;
    int setRev = 0, setHow = 0;
    char *argPtr;
```

```
    // 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;

        default:
            Error("Arguments: -n -r -t[1,2,4,8,16,32] -c\n");
            break;
        }
    }
    assert(table);
    PrintTable(reverse, size);
    return (0);
}

void
Error(char *string)
{
    printf(string);
    exit(1);
}
```

```
char keys[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'j', 'k', 'm', 'n', 'p', 'r', 's', 't' };
void
PrintTable(int reverse, int size)
{
    uInt4 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, (uInt4) 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 the CRC in packet containing "sizeInQuads" uInt4 data values
uInt4
GenerateCrc(int reverse, uInt4 * inputs, int sizeInQuads)
{
    uInt4 crcSum;

    assert(sizeInQuads >= 1);                // Packet size including CRC

    crcSum = CalculateCrc(reverse, inputs, sizeInQuads - 1);
    // compute CRC on just the data
    return (~crcSum);
}

// Validate the CRC for a packet containing "size" uInt4 data values
int
ValidateCrc(int reverse, uInt4 * inputs, int sizeInQuads)
{
    uInt4 crcSum, check;

    assert(sizeInQuads >= 1);                // Packet size including CRC
```

```
    crcSum = CalculateCrc(reverse, inputs, sizeInQuads);
    // compute CRC on the data and the received CRC
    check = reverse ? BitReverse(crcSum) : crcSum;
    return (check != CRC_RESULTS);
}

// The GenerateCrc() function points to protected values,
// it checks these values and return a final 32 - bit result
uInt4
CalculateCrc(int reverse, uInt4 * inputs, int sizeInQuads)
{
    uInt4 inQuad, crcSum, sum;
    int i;

    // The crcSum value is initialized to all ones
    crcSum = (uInt4) 0XFFFFFFFF;

    // Process each of the uInt4s covered by the CRC value
    for (i = 0; i < sizeInQuads; i += 1) {
        inQuad = reverse ? inputs[i] : BitReverse(inputs[i]);
        crcSum = CrcBits(crcSum, inQuad, 32);
    }
    sum = reverse ? BitReverse(crcSum) : crcSum;
    return (sum);
}

uInt4
CrcBits(uInt4 last, uInt4 input, int size)
{
    uInt4 crcSum, newMask;
    int i, oldBit, newBit, sumBit;

    // Process each of the bits within the input uInt4 value
    crcSum = last;
    for (newMask = MSB32, i = 0; i < size; newMask >>= 1, i += 1) {
        newBit = ((input & newMask) != 0);    // The next input bit
        oldBit = ((crcSum & MSB32) != 0);     // and MSB of crcSum
        sumBit = oldBit ^ newBit;             // are EXOR'd together

        // Shift the old crcSum left and exclusive - OR the new newBit values
        crcSum = ((crcSum << 1) & ONES32) ^ (sumBit ? CRC_COMPUTE : 0);
    }
    return (crcSum);
}

// Reverse the order of bits within bytes
uInt4
BitReverse(uInt4 old)
{
    uInt4 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);
}
```

```
// ********************************************************************************************************************************************
// ************************************************** CRC_32_EXCHANGED_BY_32 ******************************************************************
// ********************************************************************************************************************************************
#ifdef CRC_32_EXCHANGED_BY_32
// C00-through-c31 are the most- through least-significant bits of check.
// d00-through-d31 are the most- through 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 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^  p^r^        C^      G^    K^M^        T;
c01=         e^f^  h^  k^m^n^  r^s^  A^    D^      H^    M^N         ;
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^      s^    B^C^    F^              R    ;
c14= a^  c^d^  f^  h^j^  m^n^      t^  B^  D^E^  G^  J            ;
c15=     c^d^  f^  h^j^k^  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^  M^N^  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^    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^          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^  J^K^M^      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^  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^  M^N^  R^  T;
#endif  // CRC_32_EXCHANGED_BY_32
```

```
// ********************************************************************************************************************************************
// ************************************************** CRC_32_EXCHANGED_BY_16 ******************************************************************
// ********************************************************************************************************************************************
#ifdef CRC_32_EXCHANGED_BY_16
// C00-through-c31 are the most- through least-significant bits of check.
// d00-through-d15 are the most- through 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 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^g^  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                           ;
#endif  // CRC_32_EXCHANGED_BY_16
```

```
// ********************************************************************************************************************************************
// ************************************************ CRC_32_EXCHANGED_BY_8 ***********************************************************
// ********************************************************************************************************************************************
#ifdef CRC_32_EXCHANGED_BY_8
// c00-through-c31 are the most- through least-significant bits of check.
// d00-through-d07 are the most- through 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;       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
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^                              E                     ;
c13=           f^                              F                   ;
c14= a^                                          G                 ;
c15= a^b^        g^                                H               ;
c16= a^b^  d^e^                                      J             ;
c17=   b^c^  e^f^                                      K           ;
c18= a^  c^d^  f^                                        M         ;
c19= a^b^  d^e^                                            N       ;
c20= a^b^c^  e^f^g^                                          P     ;
c21=   b^c^d^  f^g^h^                                          R   ;
c22=     c^d^e^  g^h^                                            S ;
c23=       d^e^f^  h^                                              T;
c24= a^          g                                                 ;
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                                              ;
#endif  // CRC_32_EXCHANGED_BY_8


// ********************************************************************************************************************************************
// ************************************************ Old Darwin code reference *******************************************************
// **** This old code has extensive problems, but is being maintained (as a reference) during replacement code construction. ****
// ********************************************************************************************************************************************
#define OLD_CODE 0
```

```
#if OLD_CODE
#define FALSE 0
#define TRUE  1

#define LP_MU_ 1                            // Added by the editor to suppress gcc errors
#define HOPS 1                              // Added by the editor to suppress gcc errors
#define HUH 1                               // Added by the editor to suppress gcc errors

typedef uInt1 Boolean;
typedef struct
{
    uInt4 loTbDepth;              // Low priority transit buffer depth
    uInt4 addRate;                // Count of low priority (LP) and excess medium priority eMP bytes transmitted by client
    uInt4 addRateCongestion;      // Count of LP and eMP bytes transmitted by client and destined beyond the congestion point
    uInt4 lpAddRate;              // addRate run through a low pass filter
    uInt4 nlpAddRate;             // lpAddRate/WEIGHT

    uInt4 lpAddRateContestion;    // addRateCongestion run through a low pass filter
    uInt4 nlpAddRateContestion;   // lpAddRateContestion / WEIGHT
    Boolean addRateOk;            // flag indicating that client is allowed to transmit
    uInt4 allowRateCongestion;    // The fair amount each node is allowed to transmit beyond the congestion point
    uInt4 fwdRate;                // Count of LP+eMP bytes forwarded from the LP transit buffer
    uInt4 lpFwdRate;              // fwdRate run through low pass filter
    uInt4 rate_iMP;               // Count of iMP bytes forwarded and added
    uInt4 lp_rate_iMP;            // Rate_iMP run through low pass filter
    uInt4 lpAllowCongestion;      // allowRateCongestion run through low pass filter
    Boolean congested;            // Station congestion: transit buffer occupancy, link utilization or HoL timer expiration
    uInt4 accessDelay;

    uInt2 rcvdAdvertisedRate;     // The fair rate(# of bytes in a decay interval)received from the downstream neighbor
    uInt2 advertisedRate;         // The fair rate (# of bytes in a decay interval) passed along to the upstream neighbor
    sInt2 tokenBytes;            // Bytes in RPR-fa dynamic shaper/policer for eMP&LP traffic beyond congestion point
    sInt2 tokenBytesLow;         // Bytes in RPR-fa shaper/policer for low priority
    sInt2 tokenBytesMed;         // Bytes in RPR-fa shaper/policer for medium (iMP+eMP) priority
    sInt2 tokenBytesHigh;        // Bytes in RPR-fa shaper/policer for high priority
    uInt2 numberActiveSources;    // Number of active sources
    Boolean addPacketEnqueuedHigh;
    Boolean addPacketEnqueued_low;
```

```
    int fairnessOn;              // Added by the editor to eliminate gcc errors
    int mode;                    // Added by the editor to eliminate gcc errors
    int initial;                 // Added by the editor to eliminate gcc errors
    int nextState;               // Added by the editor to eliminate gcc errors
    int currentState;            // Added by the editor to eliminate gcc errors
    int reservedRate;            // Added by the editor to eliminate gcc errors
    int aboveHighThreshold;      // Added by the editor to eliminate gcc errors
    int aboveLowThreshold;       // Added by the editor to eliminate gcc errors
    int belowHighThreshold;      // Added by the editor to eliminate gcc errors
    int belowLowThreshold;       // Added by the editor to eliminate gcc errors
    int hysteresis;              // Added by the editor to eliminate gcc errors
    int loAllow;                 // Added by the editor to eliminate gcc errors
    int lpAllow;                 // Added by the editor to eliminate gcc errors
    int nodeState;               // Added by the editor to eliminate gcc errors
    int wrapped;                 // Added by the editor to eliminate gcc errors
    int addPacketEnqueued;       // Added by the editor to eliminate gcc errors
    int my_SA;                   // Added by the editor to eliminate gcc errors
    int my_RI;                   // Added by the editor to eliminate gcc errors
}
Station;

typedef struct
{
    uInt1 SA[6];
    Boolean RI;
    uInt2 rate;
}
FairnessPacket;
FairnessPacket fairnessPacket;    //received fairness packet

typedef enum
{
    FAIRNESS_OFF,
    FAIRNESS_ON,
    RAMP_UP,
    RAMP_DOWN,
    DOWNSTREAM_CONGESTED
}
States;

States currentState, nextState;
Boolean initial;


// Constants (Tunable, provisioned LME objects):
uInt2 WEIGHT;                    // Configured weight for this station an integer between 1 and 63
uInt2 w[HOPS];                   // weight vector for all stations on the ring
uInt2 BUCKET_SIZE;               // provisioned RPR-fa dynamic shaper leaky bucket size in bytes
uInt4 MAX_ALLOWANCE;             // Configured value for max allowed rate for this node
uInt4 DECAY_INTERVAL;            //    8,000 byte times @ OC-12,
                                 //   32,000 byte times @ OC-48,
                                 //  128,000 byte times @ OC-192
uInt4 ADVERTISEMENT_INTERVAL;    // Between 1 MTU time and DECAY_INTERVAL
```

```
uInt2 AGECOEFF = 4;              // Aging coeff for addRate and fwdRate
uInt2 LP_FWD = 64;               // Low pass filter for fwdRate
uInt2 LP_MU = 512;               // Low pass filter for my fair rate
uInt2 LP_MUHistory = 511;        // Low pass filter for history rate
uInt2 LP_ALLOW = 64;             // LP filter for allow rate auto increment
uInt2 LP_ALLOW_COEF = 128;       // low-pass filter for lpAllow
uInt2 NULL_RATE = 0xFFFF;        // All 1's in rcvdAdvertisedRate field.

uInt2 TB_LO_THRESHOLD;           // TB depth at which no more LP client traffic can be sent
uInt4 MAX_LRATE;                 // AGECOEFF * DECAY_INTERVAL =
                                      //  512,000 for OC-192
                                      //  128,000 for OC-48
                                      //   32,000 for OC-12

uInt4 AD_THRESHOLD;              // Default value 1ms

//Each station should have knowledge of reserved rate
uInt4 reservedRate;              // High priority reserved rate (# of bytes in a decay interval
Boolean mode;                    // Advertising mode: 1 conservative, 0 (default) aggressive

void UpdatedEveryClockCycle(Station *);
void UpdatedWhenFairnessPacketReceived(Station *, FairnessPacket *);
void CalculatedEveryDecayInterval(Station *);
void CalculatedEveryAdvertisementInterval(Station *);
void Advertise(Station *);
uInt2 F1(Station *, int);

// THESE ARE UPDATED EVERY CLOCK CYCLE:
// addRate is increment by 1 for every LP/eMP byte that is
// Transmitted by the client (does not include data
// Transmitted from the Transit Buffer).

// addRateCongestion is increment by 1 for every LP/eMP byte that is
// Transmitted by the client (does not include data
// Transmitted from the Transit Buffer) and destined beyond congestion
// point (i.e., TTL > hopsToCongestion).

// fwdRate is increment by 1 for every LP/eMP byte that exits the LP Transit Buffer

void
UpdatedEveryClockCycle(Station * sPtr)
{
    // tokenBytes is decremented by 1 for every LP/eMP byte that is transmitted by the client.
    // A packet is sent to completion, why is the below check here?
    if ((sPtr->addRateCongestion < sPtr->allowRateCongestion) &&
     (sPtr->fwdRate + sPtr->addRate + sPtr->rate_iMP < MAX_LRATE - sPtr->reservedRate) &&
     (sPtr->tokenBytes > 0) && !((sPtr->loTbDepth > 0) && (WEIGHT * sPtr->fwdRate < sPtr->addRate)) &&
     (sPtr->addRate < MAX_ALLOWANCE))
        sPtr->addRateOk = TRUE;              // true means OK to send client packets

    // Access delay is:
    //   started when an SOP arrives
    //   reset when packet is transmitted
    //   not increased when a packet is enqueued
```

```
    if (sPtr->addPacketEnqueued)
        sPtr->accessDelay += 1;              // Any packet being added except high priority
    if (sPtr->accessDelay >= AD_THRESHOLD)    // Access delay expires
        sPtr->congested = TRUE;
}

// UPDATED WHEN FAIRNESS_PKT IS RECEIVED:
void
UpdatedWhenFairnessPacketReceived(Station * sPtr, FairnessPacket * fPtr)
{
    if ((fPtr->SA[0] == sPtr->my_SA) &&       // Change by editor to suppress GCC errors
     ((sPtr->nodeState == sPtr->wrapped || fPtr->RI == sPtr->my_RI))) {
        sPtr->rcvdAdvertisedRate = NULL_RATE;
    } else {
        sPtr->rcvdAdvertisedRate = fPtr->rate;
    }
}

//steering case:
//  do not forward on,
//  for wrapping case pass set
//treat single link as segment failure.
// THE FOLLOWING IS CALCULATED EVERY DECAY_INTERVAL:
void
CalculatedEveryDecayInterval(Station * sPtr)
{
    if ((sPtr->loTbDepth > TB_LO_THRESHOLD / 2) ||      // LPTB crosses threshold
     ((sPtr->addRate + sPtr->fwdRate) > (MAX_LRATE - sPtr->reservedRate)) ||    // Link utilization cross high threshold
     (sPtr->accessDelay >= AD_THRESHOLD)) {   // access delay
        sPtr->congested = TRUE;
    } else {
        sPtr->congested = FALSE;
    }

    sPtr->lpAddRate = (sPtr->lpAddRate / (LP_MU_ - 1 / LP_MU) + (sPtr->addRate / LP_MU));       // More generic weights than LP_MU-1
    sPtr->nlpAddRate = sPtr->lpAddRate / WEIGHT;         // modified weight function
    //addRate is decremented by min(allow_rate/AGECOEFF, addRate/AGECOEFF)

    sPtr->lpFwdRate = ((LP_FWD - 1) * sPtr->lpFwdRate + sPtr->fwdRate) / LP_FWD;
    // fwdRate is decremented by fwdRate/AGECOEFF
    //(Note: lp values calculated prior to decrement of non-lp values).

    // tokenBytes is incremented at a rate which equals to
    // lpAllow /(AGECOEFF * DECAY_INTERVAL)
    sPtr->lpAllow = ((LP_ALLOW_COEF - 1) * sPtr->lpAllow + sPtr->allowRateCongestion) / LP_ALLOW_COEF;

    if (sPtr->rcvdAdvertisedRate != NULL_RATE)
        sPtr->allowRateCongestion = (sPtr->rcvdAdvertisedRate * WEIGHT);
    else
        sPtr->allowRateCongestion += (MAX_LRATE - sPtr->reservedRate - sPtr->allowRateCongestion) / LP_ALLOW;

    if (sPtr->fairnessOn && sPtr->mode == 1)
        sPtr->allowRateCongestion = sPtr->advertisedRate * WEIGHT;
```

```
    // fwd_rate_iMP is increment by 1 for every iMP byte that
    // exits the LP Transit Buffer
}

// THE FOLLOWING IS CALCULATED EVERY ADVERTISEMENT_INTERVAL:
void
CalculatedEveryAdvertisementInterval(Station * sPtr)
{
    Advertise(sPtr);
    if ((sPtr->addRate + sPtr->fwdRate) < (MAX_LRATE - sPtr->reservedRate - sPtr->hysteresis)) {
        sPtr->belowLowThreshold = TRUE;
        sPtr->aboveLowThreshold = FALSE;
    } else {
        sPtr->belowLowThreshold = FALSE;
        sPtr->aboveLowThreshold = TRUE;
    }

    if ((sPtr->addRate + sPtr->fwdRate) >= (MAX_LRATE - reservedRate)) {
        sPtr->aboveHighThreshold = TRUE;
        sPtr->belowHighThreshold = FALSE;
    } else {
        sPtr->belowHighThreshold = TRUE;
        sPtr->aboveHighThreshold = FALSE;
    }
}

void                            // rate advertisement machine
Advertise(Station * sPtr)
{
    int I = 1, J = 1;           // Added by editor to eliminate gcc errors

    sPtr->nextState = sPtr->currentState;     // No change
    switch (sPtr->currentState) {
    case FAIRNESS_OFF:
        if (sPtr->rcvdAdvertisedRate != NULL_RATE)
            //downstream_congested
            sPtr->nextState = DOWNSTREAM_CONGESTED;     //next state is downstream congested

        if (sPtr->congested)
            sPtr->nextState = FAIRNESS_ON;    // next state is congested
        sPtr->advertisedRate = NULL_RATE;     // advertises is still maximum bw

        // allowRateCongestion += (MAX_LRATE-allowRateCongestion)/LP_ALLOW;
        sPtr->initial = TRUE;

    case FAIRNESS_ON:
        if (sPtr->initial) {
            sPtr->advertisedRate = sPtr->mode ? sPtr->nlpAddRate : (MAX_LRATE - sPtr->reservedRate) / F1(sPtr, HUH);
            // calculated initial rate:fair rate per unit weight
            sPtr->initial = FALSE;
        }
        // advertise min(low pass add rate, received advertise rate)
        sPtr->advertisedRate = (sPtr->nlpAddRate < sPtr->rcvdAdvertisedRate) ? sPtr->nlpAddRate : sPtr->rcvdAdvertisedRate;
```

```
        if (sPtr->rcvdAdvertisedRate != NULL_RATE)
            sPtr->nextState = DOWNSTREAM_CONGESTED;     // next state is downstream congested
        if (sPtr->belowLowThreshold)
            sPtr->nextState = RAMP_UP;
        //allowRateCongestion


    case RAMP_UP:
        sPtr->advertisedRate = sPtr->mode ? NULL_RATE : sPtr->lpAddRate + sPtr->lpAddRate * I / J;
        if (sPtr->advertisedRate > (MAX_LRATE - sPtr->reservedRate))
            sPtr->advertisedRate = NULL_RATE;
        if (sPtr->aboveHighThreshold)
            sPtr->nextState = RAMP_DOWN;
        if (sPtr->aboveLowThreshold)
            sPtr->nextState = FAIRNESS_ON;

        if (sPtr->advertisedRate == NULL_RATE)
            sPtr->nextState = FAIRNESS_OFF;
        //allowRateCongestion

    case RAMP_DOWN:
        sPtr->advertisedRate = sPtr->mode ? sPtr->lpAddRate : sPtr->lpAddRate - sPtr->lpAddRate * I / J;
        if (sPtr->belowHighThreshold)
            sPtr->nextState = FAIRNESS_ON;

        if (sPtr->belowLowThreshold)
            sPtr->nextState = RAMP_UP;
        //allowRateCongestion

    case DOWNSTREAM_CONGESTED:
        if (sPtr->lpFwdRate > (sPtr->allowRateCongestion / WEIGHT))
            sPtr->advertisedRate = sPtr->rcvdAdvertisedRate;
        else
            sPtr->advertisedRate = NULL_RATE;
        sPtr->allowRateCongestion = (sPtr->rcvdAdvertisedRate * WEIGHT);
        // WEIGHT is local weight * received control message

        if (sPtr->rcvdAdvertisedRate == NULL_RATE)
            sPtr->nextState = FAIRNESS_OFF;
        sPtr->currentState = sPtr->nextState;
    }
}

uInt2
F1(Station * sPtr, int weightStation)
{
    int i;
    uInt2 weightTotalActive = 0;  // Local variable weightTotalActive
    // input weightStation;
    // mode 1: weight == stationWeight;
    //
```

```
    for (i = 1; sPtr->numberActiveSources; i += 1)
        weightTotalActive += w[i];           // Add all active weight global variable
    weightTotalActive += weightStation;      // Add local station weight
    return (weightTotalActive);
}

#endif // OLD_CODE
// ********************************************************************************************************************************
// ********************************************** IEEE C-code illustrations  ******************************************************
// ************************ This new code is preliminary and is being updated to track RPR developments ***************************
// ********************************************************************************************************************************
#define NEW_CODE 1

#if NEW_CODE
#define HOPS 64                                     // Worst-case number of hops
#define MTU 9216                                    // Worst-case frame length
#define FTU (MTU+8)                                 // Fifo occupancy size

#define MINIMUM(x,y) (((x)-(y))>0 ? (x):(y))        // Modulo-arithmetic-compatible maximum
#define MAXIMUM(x,y) (((x)-(y))<0 ? (x):(y))        // Modulo-arithmetic-compatible minimum
#define MINIMUM3(x,y,z) MINIMUM(MINIMUM(x,y),z)     // Minimum of three values
#define MIN_MAX(x,min,max) MAXIMUM(MINIMUM(x,max),min)  // Min/Max bounding function
#define ONE (((uInt8)1)<<32)                        // Scalar for {integer:32, fraction:32} values
#define MCAST_BIT (((uInt8)1)<<48)                  // Multicast bit with MAC addresses
#define UCAST 0                                     // Distinguishing unicast of unicast/multicast states
#define MCAST 8                                     // Distinguishing multicast of unicast/multicast states

//       1         2         3         4         5         6         7         8         9         0         1         2         3
// 34567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
typedef enum
{
    EOP_STB,                        // Secondary transmit buffer
    EOP_SPATIAL_AB                  // Spatial class-A and class-B provisions
}
Options;
#define MOP_STB ((sPtr->options&(1<<EOP_STB))!=0)
#define MOP_SPATIAL_AB ((sPtr->options&(1<<EOP_SPATIAL_AB))!=0)

typedef enum
{
    CLASS_A,                        // Lowest jitter provisioned bandwidth
    CLASS_B,                        // Higher jitter provisioned bandwidth
    CLASS_C                         // Unprovisioned or unused provisioned bandwidth
}
BaseClasses;
```

```
typedef enum
{
    CLASS_A0,                       // Class-A temporal-unrecoverable bandwidth
    CLASS_A1,                       // Class-A temporal-recoverable bandwidth
    CLASS_B0,                       // Class-B committed bandwidth
    CLASS_B1,                       // Class-B excess bandwidth
    CLASS_Cx                        // Class-C opportunistic bandwidth
}
SubClasses;

typedef enum
{
    SC_Ca0,                         // Client sourced class-A0
    SC_Ca1,                         // Client sourced class-A1
    SC_Cb0,                         // Client sourced class-B0
    SC_Cb1,                         // Client sourced class-B1
    SC_Cc,                          // Client sourced class-C

    SC_Ma0=8,                       // MAC sourced class-A0

    SC_Ta0=16,                      // Transit sourced class-A0
    SC_Ta1,                         // Transit sourced class-A1
    SC_Tb0,                         // Transit sourced class-B0
    SC_Tb1,                         // Transit sourced class-B1
    SC_Tc,                          // Transit sourced class-C

    SC_Not=24                       // Default sourced idle
}
SrcClasses;

typedef struct
{
    // Internal 2-byte header identifies next-frame boundary
    unsigned sourceTag:2;                               // Sourced location (client, MAC, or transit)
    unsigned frameSize:14;                              // Frame-size specifier

    // Header values themselves
    unsigned timeToLive:8;                              // Aging field
    unsigned runId:1;                                   // Run identifier
    unsigned subClass:1;                                // Subclass identifier
    unsigned frameType:2;                               // Frame identifier
    unsigned serviceClass:2;                            // Base class specifier
    unsigned wrapping:1;                                // Wrapping is allowed
    unsigned flooding:1;                                // Flooding to remote MAC addresses
    unsigned destinationMacAddressHi:32;                // Destination MAC addresses (MSBs)
    unsigned destinationMacAddressLo:16;                // Destination MAC addresses (LSBs)
    unsigned sourceMacAddressHi:16;                     // Destination MAC addresses (LSBs)
    unsigned sourceMacAddressLo:32;                     // Destination MAC addresses (LSBs)
    unsigned etherType:16;                              // Payload type field
    unsigned headerCrc:16;                              // HeaderCRC
```

```
    // Following quadlet-aligned payload values
    unsigned basePayload:32;                              // Payload start address
}
Frame;

typedef struct {
    // These arrays entries support spatial and uniform provisioning (for uniform provisioning, only first entry is used).
    uInt8 credits;                      // Transmission credits
    uInt4 rate;                         // Transmission rate
    uInt4 hiLimit;                      // High credit limit
    uInt4 loLimit;                      // Low credit limit
    uInt1 pass;                         // Allowed transmission indication
}
UniformShaper;

typedef struct {
    // These arrays entries support spatial and uniform provisioning (for uniform provisioning, only first entry is used).
    uInt8 credits[HOPS];                // Transmission credits
    uInt4 rate[HOPS];                   // Transmission rate
    uInt4 hiLimit;                      // High credit limit
    uInt4 loLimit;                      // Low credit limit
    uInt1 pass;                         // Allowed transmission indication
}
SpatialShaper;

typedef struct
{
    uInt8 rprRingRxUcastClassA0Frames;
    uInt8 rprRingRxUcastClassA0Bytes;
    uInt8 rprRingRxUcastClassA1Frames;
    uInt8 rprRingRxUcastClassA1Bytes;
    uInt8 rprRingRxUcastClassB0Frames;
    uInt8 rprRingRxUcastClassB0Bytes;
    uInt8 rprRingRxUcastClassB1Frames;
    uInt8 rprRingRxUcastClassB1Bytes;
    uInt8 rprRingRxUcastClassCxFrames;
    uInt8 rprRingRxUcastClassCxBytes;

    uInt8 rprRingRxMcastClassA0Frames;
    uInt8 rprRingRxMcastClassA0Bytes;
    uInt8 rprRingRxMcastClassA1Frames;
    uInt8 rprRingRxMcastClassA1Bytes;
    uInt8 rprRingRxMcastClassB0Frames;
    uInt8 rprRingRxMcastClassB0Bytes;
    uInt8 rprRingRxMcastClassB1Frames;
    uInt8 rprRingRxMcastClassB1Bytes;
    uInt8 rprRingRxMcastClassCxFrames;
    uInt8 rprRingRxMcastClassCxBytes;
```

```
uInt8 rprRingTxUcastClassA0Frames;
uInt8 rprRingTxUcastClassA0Bytes;
uInt8 rprRingTxUcastClassA1Frames;
uInt8 rprRingTxUcastClassA1Bytes;
uInt8 rprRingTxUcastClassB0Frames;
uInt8 rprRingTxUcastClassB0Bytes;
uInt8 rprRingTxUcastClassB1Frames;
uInt8 rprRingTxUcastClassB1Bytes;
uInt8 rprRingTxUcastClassCxFrames;
uInt8 rprRingTxUcastClassCxBytes;

uInt8 rprRingTxMcastClassA0Frames;
uInt8 rprRingTxMcastClassA0Bytes;
uInt8 rprRingTxMcastClassA1Frames;
uInt8 rprRingTxMcastClassA1Bytes;
uInt8 rprRingTxMcastClassB0Frames;
uInt8 rprRingTxMcastClassB0Bytes;
uInt8 rprRingTxMcastClassB1Frames;
uInt8 rprRingTxMcastClassB1Bytes;
uInt8 rprRingTxMcastClassCxFrames;
uInt8 rprRingTxMcastClassCxBytes;

uInt8 rprClientRxUcastClassA0Frames;
uInt8 rprClientRxUcastClassA0Bytes;
uInt8 rprClientRxUcastClassA1Frames;
uInt8 rprClientRxUcastClassA1Bytes;
uInt8 rprClientRxUcastClassB0Frames;
uInt8 rprClientRxUcastClassB0Bytes;
uInt8 rprClientRxUcastClassB1Frames;
uInt8 rprClientRxUcastClassB1Bytes;
uInt8 rprClientRxUcastClassCxFrames;
uInt8 rprClientRxUcastClassCxBytes;

uInt8 rprClientRxMcastClassA0Frames;
uInt8 rprClientRxMcastClassA0Bytes;
uInt8 rprClientRxMcastClassA1Frames;
uInt8 rprClientRxMcastClassA1Bytes;
uInt8 rprClientRxMcastClassB0Frames;
uInt8 rprClientRxMcastClassB0Bytes;
uInt8 rprClientRxMcastClassB1Frames;
uInt8 rprClientRxMcastClassB1Bytes;
uInt8 rprClientRxMcastClassCxFrames;
uInt8 rprClientRxMcastClassCxBytes;

uInt8 rprClientTxUcastClassA0Frames;
uInt8 rprClientTxUcastClassA0Bytes;
uInt8 rprClientTxUcastClassA1Frames;
uInt8 rprClientTxUcastClassA1Bytes;
uInt8 rprClientTxUcastClassB0Frames;
uInt8 rprClientTxUcastClassB0Bytes;
uInt8 rprClientTxUcastClassB1Frames;
uInt8 rprClientTxUcastClassB1Bytes;
uInt8 rprClientTxUcastClassCxFrames;
uInt8 rprClientTxUcastClassCxBytes;
```

```
    uInt8 rprClientTxMcastClassA0Frames;
    uInt8 rprClientTxMcastClassA0Bytes;
    uInt8 rprClientTxMcastClassA1Frames;
    uInt8 rprClientTxMcastClassA1Bytes;
    uInt8 rprClientTxMcastClassB0Frames;
    uInt8 rprClientTxMcastClassB0Bytes;
    uInt8 rprClientTxMcastClassB1Frames;
    uInt8 rprClientTxMcastClassB1Bytes;
    uInt8 rprClientTxMcastClassCxFrames;
    uInt8 rprClientTxMcastClassCxBytes;

    uInt8 rprRingRxErrorHeaderCrcFrames;
    uInt8 rprRingRxErrorPayloadCrcFrames;
    uInt8 rprRingRxErrorTimeToLive0Frames;
    uInt8 rprRingRxErrorTimeToLive1Frames;
    uInt8 rprRingRxErrorMcastSourceFrames;
    uInt8 rprRingRxErrorWrongRunFrames;
}
Counters;

// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ********************************** Functional FifoQueue parameters to be supplied ***************************************
// ********************************************************************************************************************************
typedef struct
{
    uInt4 dummy0;                       // Dummy information
    uInt4 dummy1;                       // Dummy information
}
FifoQueue;

typedef struct
{
    uInt1 runId;                        // Run identifier
    sInt4 options;                      // Options
    Frame *macControlPtr;               // MAC control frame
    Counters counters;                  // Performance and error-logging counters
    UniformShaper uniformM;             // MAC control shaper
    UniformShaper uniformS;             // Sustaining STB shaper
    UniformShaper uniformA;             // Class-A shaper
    UniformShaper uniformB;             // Class-B shaper
    UniformShaper uniformD;             // Downstream shaper
    SpatialShaper uniformC;             // Type-A fairness placeholder
    SpatialShaper spatialA;             // Class-A shaper
    SpatialShaper spatialB;             // Class-B shaper
    SpatialShaper spatialD;             // Downstream shaper
    SpatialShaper spatialC;             // Type-B fairness placeholder
    FifoQueue ptb;                      // Primary transit buffer
    FifoQueue stb;                      // Secondary transit buffer (when applicable)
    FifoQueue stage;                    // Staging buffer
    uInt1 sendA, sendB, sendC;          // Control indications for the client
}
Attachment;
```

```
typedef struct
{
    unsigned promiscuous:1;             // Promiscuous mode
    unsigned reserved:31;              // Reserved
}
Modes;

typedef struct
{
    Modes modes;                       // Operational modes
    uInt4 *eastStation;                // East station connection
    uInt4 *westStation;                // West station connection
    sInt4 options;                     // Options
    uInt8 macAddress;                  // Unique MAC address
    Attachment attachment[2];          // Attachment point states
}
Station;

// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ********************************** Functional idle-frame template to be provided ***************************************************
// ********************************************************************************************************************************
Frame idleFrame;

void  ReceiveFrame(Station *, Attachment *, Frame *);
uInt4 HeaderCrcCheck(Frame *);
uInt4 PayloadCrcCheck(Frame *);
void  Enqueue(Station *, Frame *, FifoQueue *);
void  AddressExtract(Frame *, uInt8 *, uInt8 *);
int   RejectClass(Station *, Attachment *, Frame *, uInt8);
int   CopyClass(Station *, Attachment *, Frame *, uInt8, uInt8);
int   StripClass(Station *, Attachment *, Frame *, uInt8, uInt8);
void  SelectTransmit(Station *, Attachment *);
Frame *SelectMonoTransmit(Station *, Attachment *);
Frame *SelectDualTransmit(Station *, Attachment *);
void  UpdateSends(Station *, Attachment *);
void  TransmitFrame(Station *, Attachment *, Frame *);
void  StagingFrame(Station *, Attachment *, Frame *);
void  UniformUpdate(Station *, UniformShaper *, uInt4, int, int);
void  SpatialUpdate(Station *, SpatialShaper *, uInt4, int, int);
uInt4 DepthToRateBC(uInt4);
Frame *Dequeue(Station *, FifoQueue *);
void  StompCrc(Frame *);
void  CopyToClient(Station *, Attachment *, Frame *);
uInt4 FifoDepth(FifoQueue *);
uInt4 FifoSized(FifoQueue *);
uInt4 DepthToRateBC(uInt4);
void  UpdateRxCounters(Station *, Attachment *, Frame *, uInt8);
void  UpdateTxCounters(Station *, Attachment *, Frame *, uInt8);
```

```
void
ReceiveFrame(Station *sPtr, Attachment *aPtr, Frame *fPtr)
{
    uInt4 delta;
    uInt8 destinMacAddress, sourceMacAddress;

    if (HeaderCrcCheck(fPtr)!=0) {                                  // Discard frames if the header CRC is bad
        aPtr->counters.rprRingRxErrorHeaderCrcFrames+= 1;
        return;
    }

    delta= PayloadCrcCheck(fPtr);                                   // Check the payload CRC also
    if (delta!=0) {                                                 // Stomp all bad payload CRC values
        if (delta!= 0XFFFFFFFF)                                     // Log only the non-stomped CRC values
            aPtr->counters.rprRingRxErrorPayloadCrcFrames+= 1;
        StompCrc(fPtr);
    }

    AddressExtract(fPtr, &destinMacAddress, &sourceMacAddress);     // Extract the destination & source MAC addresses
    if (RejectClass(sPtr, aPtr, fPtr, sourceMacAddress))            // Discard frames with illegal contents
        return;

    UpdateRxCounters(sPtr, aPtr, fPtr, destinMacAddress);           // Update MIB counters
    if (CopyClass(sPtr, aPtr, fPtr, destinMacAddress, sourceMacAddress))  // Check which frames are saved,
        CopyToClient(sPtr, aPtr, fPtr);                             //  and copy them to the client

    fPtr->timeToLive-= 1;                                           // Decrement timeToLive in pass-through frames
    if (StripClass(sPtr, aPtr, fPtr, destinMacAddress, sourceMacAddress))  // Strip frames if this is the target location
        return;

    if (MOP_STB==0 || fPtr->serviceClass==CLASS_A) {                // All class-A frames are saved in the PTQ
        Enqueue(sPtr, fPtr, &(aPtr->ptb));                          // For the mono-queue option, save everything in PTQ
    } else {                                                        // For the dual-queue option,
        Enqueue(sPtr, fPtr, &(aPtr->stb));                          //  save the non class-A traffic in the STQ
    }
}

// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ********************************** Functional CRC check routine to be supplied ***************************************************
// ********************************************************************************************************************************
uInt4
HeaderCrcCheck(Frame *fPtr)
{
    IS_USED1(fPtr);
    return(0);
}
```

```
// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ************** Functional CRC check routine to be supplied, through interface to preceding CRC checking code ******************
// ********************************************************************************************************************************
uInt4
PayloadCrcCheck(Frame *fPtr)
{
    IS_USED1(fPtr);
    return(0);
}

// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ************************** Functional FIFO structure, Enqueue(), and Dequeue() to be supplied ********************************
// ********************************************************************************************************************************
void
Enqueue(Station *sPtr, Frame *fPtr, FifoQueue *fifoPtr)
{

    IS_USED3(sPtr, fPtr, fifoPtr);
}

// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ************************************** StompCrc() algorithm to be supplied ***************************************************
// ********************************************************************************************************************************
void
StompCrc(Frame *fPtr)
{
    IS_USED1(fPtr);
}

// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ************************************* CopyToClient() algorithm to be supplied ************************************************
// ********************************************************************************************************************************
void
CopyToClient(Station *sPtr, Attachment *aPtr, Frame *fPtr)
{
    IS_USED3(sPtr, aPtr, fPtr);
}

// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ************************************** FifoSized() algorithm to be supplied **************************************************
// ********************************************************************************************************************************
uInt4
FifoSized(FifoQueue *fifoPtr)
{
    IS_USED1(fifoPtr);
    return(0);
}
```

```
// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ************************************** FifoDepth() algorithm to be supplied ****************************************************
// ********************************************************************************************************************************
uInt4
FifoDepth(FifoQueue *fifoPtr)
{
    IS_USED1(fifoPtr);
    return(0);
}

void                                                                    // Extract destination and source MAC addresses
AddressExtract(Frame *fPtr, uInt8 *destin, uInt8 *source)
{
    uInt8 sourceMac, destinMac;

    destinMac= fPtr->destinationMacAddressHi;                           // Capture the most-significant bits
    destin[0]= (destinMac<<16)|fPtr->destinationMacAddressLo;           //  and include the least-significant bits
    sourceMac= fPtr->sourceMacAddressHi;                                // Capture the most-significant bits
    source[0]= (sourceMac<<32)|fPtr->sourceMacAddressLo;                //  and include the least-significant bits
}

int                                                                     // Reject illegal frame parameters
RejectClass(Station *sPtr, Attachment *aPtr, Frame *fPtr, uInt8 sourceMacAddress)
{
    if (fPtr->timeToLive==0) {                                          // Frames with a zero-valued timeToLive
        aPtr->counters.rprRingRxErrorTimeToLive0Frames+= 1;             // are discarded before passing-onward
        return(1);
    }
    if ((sourceMacAddress&MCAST_BIT)!=0) {                              // The sourceMacAddress never identifies
        aPtr->counters.rprRingRxErrorMcastSourceFrames+= 1;             // a multicast source location
        return(1);
    }
    if (fPtr->wrapping==0 && fPtr->runId!=aPtr->runId) {                 // Frames that don’t support wrapping
        aPtr->counters.rprRingRxErrorWrongRunFrames+= 1;                // can never appear on the opposing run
        return(1);
    }
    return(0);
}
int                                                                     // Check whether frame is copied to client
CopyClass(Station *sPtr, Attachment *aPtr, Frame *fPtr, uInt8 destinMacAddress, uInt8 sourceMacAddress)
{
    if (sPtr->modes.promiscuous!=0)                                      // While in the promiscuous mode,
        return(1);                                                      // the client gets everything

    if (fPtr->runId!=aPtr->runId)                                       // While wrapping on the opposing run
        return(0);                                                      // frames are not copied to the client

    if (destinMacAddress==sPtr->macAddress)                             // Matching destinationMacAddress frames
        return(1);                                                      // are copied to the client

    if (sourceMacAddress==sPtr->macAddress)                             // Matching sourceMacAddress frames
        return(0);                                                      // are ignored by the client
```

```
    if (fPtr->flooding!=0)                                          // Flooding frames on bridge-featured stations
        return(1);                                                  // are copied to the client

    return(0);                                                      // By default, pass-through frames are ignored
}

int                                                                 // Strip frames after allowing copy-to-client
StripClass(Station *sPtr, Attachment *aPtr, Frame *fPtr, uInt8 destinMacAddress, uInt8 sourceMacAddress)
{
    if (fPtr->timeToLive==0) {                                      // Aged frames that would be discarded at the
        aPtr->counters.rprRingRxErrorTimeToLive0Frames+= 1;         // downstream station are discarded here
        return(1);
    }

    if (fPtr->runId!=aPtr->runId)                                   // While wrapping on the opposing run,
        return(0);                                                  // frames are ignored and pass through

    if (destinMacAddress==sPtr->macAddress)                         // Frames with a matching destinationMacAddress
        return(1);                                                  // are discarded at their destination

    if (sourceMacAddress==sPtr->macAddress)                         // Frames with a matching sourceMacAddress
        return(1);                                                  // are discarded having circulated once

    return(0);                                                      // By default, frame pass-through stations
}

// Called after each transmission to select the next frame transmission
void
SelectTransmit(Station *sPtr, Attachment *aPtr)
{
    Frame *fPtr;
    uInt4 fifoSize;
    uInt1 allow;

    fPtr= aPtr->macControlPtr;
    fifoSize= FifoSized(&(aPtr->ptb))-FifoDepth(&(aPtr->ptb));
    allow= fPtr!=NULL && fPtr->frameSize < fifoSize;
    if (allow && aPtr->uniformM.pass!=0) {
        TransmitFrame(sPtr, aPtr, aPtr->macControlPtr);
        aPtr->macControlPtr= NULL;
        return;
    }
    if (MOP_STB==0)
        fPtr= SelectMonoTransmit(sPtr, aPtr);
    else
        fPtr= SelectDualTransmit(sPtr, aPtr);
    TransmitFrame(sPtr, aPtr, fPtr);
}

Frame *
SelectMonoTransmit(Station *sPtr, Attachment *aPtr)
{
    Frame *fPtr;
```

```
    fPtr= Dequeue(sPtr, &(aPtr->ptb));
    if (fPtr!=NULL) {
        TransmitFrame(sPtr, aPtr, fPtr);
        return(fPtr);
    }
    fPtr= Dequeue(sPtr, &(aPtr->stage));
    if (fPtr!=NULL) {
        TransmitFrame(sPtr, aPtr, fPtr);
        return(fPtr);
    }
    return(&(idleFrame));
}

// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ********************************** SelectDualTransmit() algorithm to be supplied ***************************************************
// ********************************************************************************************************************************
Frame *
SelectDualTransmit(Station *sPtr, Attachment *aPtr)
{
    IS_USED2(sPtr, aPtr);
    return(&(idleFrame));
}

// ********************************************************************************************************************************
// ***************************** Editor note (DVJ): To be removed prior to final publication. ***********************************
// ************************************** Dequeue() algorithm to be supplied ******************************************************
// ********************************************************************************************************************************
Frame *
Dequeue(Station *sPtr, FifoQueue *fifoPtr)
{
    IS_USED2(sPtr, fifoPtr);
    return(NULL);
}

void                                                    // Update client's send indications based
UpdateSends(Station *sPtr, Attachment *aPtr)            // on shaper-supplied pass indications
{
    FifoQueue *fifoPtr= &(aPtr->stage);
    uInt4 fifoSpace;
    uInt1 passA, passB, passD, passC;
```

```
    fifoSpace= FifoSized(fifoPtr)-FifoDepth(fifoPtr);
    if (fifoSpace < FTU) {                                  // When the stage buffer is effectively full,
        aPtr->sendA= aPtr->sendB= aPtr->sendC= 0;           // all send indications are withdrawn
        return;
    }
    if (EOP_SPATIAL_AB==0) {                                       // The class-B/C sends are also inhibited
        passA= aPtr->uniformA.pass;
        passB= aPtr->uniformB.pass;
        passC= aPtr->uniformC.pass;
        passD= aPtr->uniformD.pass;
    } else {
        passA= aPtr->spatialA.pass;
        passB= aPtr->spatialB.pass;
        passD= aPtr->spatialD.pass;
        passC= aPtr->spatialC.pass;
    }

    aPtr->sendA= passA;                                     // The class-A send depends on no others

    if (MOP_STB==0) {                                       // The class-B/C sends are also inhibited
        aPtr->sendB= MINIMUM(passB, passD);                 // to support downstream class-A traffic
        aPtr->sendC= MINIMUM(passC, passD);
    } else {                                                // Client shaping avoids STQ starvation
        aPtr->sendB= MINIMUM3(passB, passD, aPtr->uniformS.pass);
        aPtr->sendC= MINIMUM3(passC, passD, aPtr->uniformS.pass);
    }
}

// Update credits based on transmitted frame size
void
TransmitFrame(Station *sPtr, Attachment *aPtr, Frame *fPtr)
{
    uInt2 size= fPtr->frameSize;
    uInt4 typed, types;

    typed= (fPtr->sourceTag<<3)|((1<<(fPtr->serviceClass))|fPtr->subClass);     // The MAC control traffic is rate-limited
    UniformUpdate(sPtr, &(aPtr->uniformM), size, typed&(SC_Ma0), 1);           // based on recovery time between sends

    types= SC_Ca0|SC_Ca1|SC_Ta0|SC_Ta1|SC_Ma0|SC_Not;                          // The downstream shaper...
    if (EOP_SPATIAL_AB==0)
        UniformUpdate(sPtr, &(aPtr->uniformD), size, typed&types, 1);
    else
        SpatialUpdate(sPtr, &(aPtr->spatialD), size, typed&types, 1);
}

// Update credits based on staged frame size
void
StagingFrame(Station *sPtr, Attachment *aPtr, Frame *fPtr)
{
    FifoQueue *fifoPtr;
    uInt4 typed, type1, type2, depth;
    uInt2 size= fPtr->frameSize;

    typed= (fPtr->sourceTag<<3)|((1<<(fPtr->serviceClass))|fPtr->subClass);
```

```
    if (MOP_STB!=0) {
        fifoPtr= &(aPtr->stb);
        depth= (FifoDepth(fifoPtr)*ONE)/FifoSized(fifoPtr);                    // The shaper rate depends on the
        aPtr->uniformS.rate= DepthToRateBC(depth);                             // normalized depth of the STQ
        type1= SC_Cb0|SC_Cb1|SC_Cc;                                            // For dual-queue designs, rate-limiting
      type2= SC_Tb0|SC_Tb1|SC_Tc;                                        // applies to the ratio of non-class-A
        UniformUpdate(sPtr, &(aPtr->uniformS), size, typed&type1, typed&type2); // traffic from the client and transit
    }

    if (EOP_SPATIAL_AB==0) {
        UniformUpdate(sPtr, &(aPtr->uniformA), size, typed&(SC_Ca0|SC_Ca1), 1); // The class-A traffic is rate limited
        UniformUpdate(sPtr, &(aPtr->uniformB), size, typed&(SC_Cb0|SC_Cb1), 1); // The class-B traffic is rate limited
    } else {
        SpatialUpdate(sPtr, &(aPtr->spatialA), size, typed&(SC_Ca0|SC_Ca1), 1); // The class-A traffic is rate limited
        SpatialUpdate(sPtr, &(aPtr->spatialB), size, typed&(SC_Cb0|SC_Cb1), 1); // The class-B traffic is rate limited
    }
}

// Called by TransmitFrame() or StagingFrame(), for uniform-shaper adjustments:
//   sPtr      - pointer to station context
//   usPtr     - uniform shaper context
//   size      - transmission size
//   sending   - transmission credit reduction
//   bumping   - weighted credit accumulations
void
UniformUpdate(Station *sPtr, UniformShaper *usPtr, uInt4 size, int sending, int bumping)
{
    uInt8 credits, hiLevel, loLevel;
    uInt4 sendSize, bumpSize;

    sendSize= sending!=0 ? size:0;
    bumpSize= bumping!=0 ? size:0;
    credits = usPtr->credits;
    hiLevel = usPtr->hiLimit * ONE;
    loLevel = usPtr->loLimit * ONE;
    credits = credits + (bumpSize * usPtr->rate) - (sendSize * ONE);
    credits = MIN_MAX(credits, loLevel, hiLevel);
    usPtr->pass= (credits>loLevel) ? HOPS : 0;
    usPtr->credits= credits;
}

// Called by TransmitFrame() or StagingFrame(), for spatial-shaper adjustments:
//   sPtr      - pointer to station context
//   ssPtr     - spatial shaper context
//   size      - transmission size
//   sending   - transmission credit reduction
//   bumping   - weighted credit accumulations
void
SpatialUpdate(Station *sPtr, SpatialShaper *ssPtr, uInt4 size, int sending, int bumping)
{
    uInt8 credits, hiLevel, loLevel;
    uInt4 sendSize, bumpSize;
```

```
    sendSize= sending!=0 ? size:0;
    bumpSize= bumping!=0 ? size:0;
    credits = ssPtr->credits[0];
    hiLevel = ssPtr->hiLimit * ONE;
    loLevel = ssPtr->loLimit * ONE;
    credits = credits + (bumpSize * ssPtr->rate[0]) - (sendSize * ONE);
    credits = MIN_MAX(credits, loLevel, hiLevel);
    ssPtr->pass= (credits>loLevel) ? HOPS : 0;
    ssPtr->credits[0]= credits;
}

uInt4                                                    // Compute depth-dependent shaper-S rate
DepthToRateBC(uInt4 depth)
{
    uInt4 rate;

    rate = ONE - (2 * depth);
    rate = MIN_MAX(rate, 0, (ONE*7)/8);
    return(rate);
}


void      // Update ring-receiver performance monitoring statistics
UpdateRxCounters(Station *sPtr, Attachment *aPtr, Frame *fPtr, uInt8 destinMacAddress)
{
    int class= (fPtr->serviceClass<<1)|fPtr->subClass;
    int size= fPtr->frameSize;
    int multi= (destinMacAddress&MCAST_BIT)!=0 ? MCAST : UCAST;

    switch(multi|class) {
    case UCAST|CLASS_A0:
        aPtr->counters.rprRingRxUcastClassA0Frames+= 1;
        aPtr->counters.rprRingRxUcastClassA0Bytes+= size;
        break;

    case UCAST|CLASS_A1:
        aPtr->counters.rprRingRxUcastClassA1Frames+= 1;
        aPtr->counters.rprRingRxUcastClassA1Bytes+= size;
        break;

    case UCAST|CLASS_B0:
        aPtr->counters.rprRingRxUcastClassB0Frames+= 1;
        aPtr->counters.rprRingRxUcastClassB0Bytes+= size;
        break;

    case UCAST|CLASS_B1:
        aPtr->counters.rprRingRxUcastClassB1Frames+= 1;
        aPtr->counters.rprRingRxUcastClassB1Bytes+= size;
        break;

    case UCAST|CLASS_Cx:
        aPtr->counters.rprRingRxUcastClassCxFrames+= 1;
        aPtr->counters.rprRingRxUcastClassCxBytes+= size;
        break;
```

```
    case MCAST|CLASS_A0:
        aPtr->counters.rprRingRxMcastClassA0Frames+= 1;
        aPtr->counters.rprRingRxMcastClassA0Bytes+= size;
        break;

    case MCAST|CLASS_A1:
        aPtr->counters.rprRingRxMcastClassA1Frames+= 1;
        aPtr->counters.rprRingRxMcastClassA1Bytes+= size;
        break;

    case MCAST|CLASS_B0:
        aPtr->counters.rprRingRxMcastClassB0Frames+= 1;
        aPtr->counters.rprRingRxMcastClassB0Bytes+= size;
        break;

    case MCAST|CLASS_B1:
        aPtr->counters.rprRingRxMcastClassB1Frames+= 1;
        aPtr->counters.rprRingRxMcastClassB1Bytes+= size;
        break;

    case MCAST|CLASS_Cx:
        aPtr->counters.rprRingRxMcastClassCxFrames+= 1;
        aPtr->counters.rprRingRxMcastClassCxBytes+= size;
        break;
    }
}

void      // Update ring-transmitter performance monitoring statistics
UpdateTxCounters(Station *sPtr, Attachment *aPtr, Frame *fPtr, uInt8 destinMacAddress)
{
    int class= (fPtr->serviceClass<<1)|fPtr->subClass;
    int size= fPtr->frameSize;
    int multi= (destinMacAddress&MCAST_BIT)!=0 ? MCAST : UCAST;

    switch(multi|class) {
    case UCAST|CLASS_A0:
        aPtr->counters.rprRingTxUcastClassA0Frames+= 1;
        aPtr->counters.rprRingTxUcastClassA0Bytes+= size;
        break;
    case UCAST|CLASS_A1:
        aPtr->counters.rprRingTxUcastClassA1Frames+= 1;
        aPtr->counters.rprRingTxUcastClassA1Bytes+= size;
        break;

    case UCAST|CLASS_B0:
        aPtr->counters.rprRingTxUcastClassB0Frames+= 1;
        aPtr->counters.rprRingTxUcastClassB0Bytes+= size;
        break;

    case UCAST|CLASS_B1:
        aPtr->counters.rprRingTxUcastClassB1Frames+= 1;
        aPtr->counters.rprRingTxUcastClassB1Bytes+= size;
        break;
```

```
    case UCAST|CLASS_Cx:
        aPtr->counters.rprRingTxUcastClassCxFrames+= 1;
        aPtr->counters.rprRingTxUcastClassCxBytes+= size;
        break;

    case MCAST|CLASS_A0:
        aPtr->counters.rprRingTxMcastClassA0Frames+= 1;
        aPtr->counters.rprRingTxMcastClassA0Bytes+= size;
        break;

    case MCAST|CLASS_A1:
        aPtr->counters.rprRingTxMcastClassA1Frames+= 1;
        aPtr->counters.rprRingTxMcastClassA1Bytes+= size;
        break;

    case MCAST|CLASS_B0:
        aPtr->counters.rprRingTxMcastClassB0Frames+= 1;
        aPtr->counters.rprRingTxMcastClassB0Bytes+= size;
        break;

    case MCAST|CLASS_B1:
        aPtr->counters.rprRingTxMcastClassB1Frames+= 1;
        aPtr->counters.rprRingTxMcastClassB1Bytes+= size;
        break;

    case MCAST|CLASS_Cx:
        aPtr->counters.rprRingTxMcastClassCxFrames+= 1;
        aPtr->counters.rprRingTxMcastClassCxBytes+= size;
        break;
    }
}

#endif   // NEW_CODE
```

Annex H

(informative)

Spatial indications and shaping

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	No changes.
Draft 0.3, June 2002	Added Class A/B shaping information.
Draft 1.0, August 2002	Modified according to comments on D0.3.
Draft 1.1, October 2002	Modified according to comments on D1.0.
Draft 2.0, December 2002	Modified according to comments on D1.1 for WG ballot.

H.1 Overview

This clause describes a possible extension beyond the scope of the standard. This is text is included only for illustrative purposes for those implementers who might want to extend this standard 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. This clause describes a means beyond this standard to accomplish spatial traffic handling.

H.2 Spatial bandwidth allocation

This subclause describes how bandwidth allocation profiles are created in a spatially aware system.

H.2.1 Single-queue spatial allocation

Each station has allocated levels of classA and classB traffic, as illustrated in Figure H.1, which shows allocations for station S3. Spatial allocation makes a distinction between classA traffic sent (panel 1) from S3-to-S2 and classA traffic (panel 2) sent from S3-to-S1. Similarly, spatial allocation makes a distinction between classB traffic (panel 3) sent from S3-to-S1 and classB traffic (panel 4) sent from S3-to-S4.

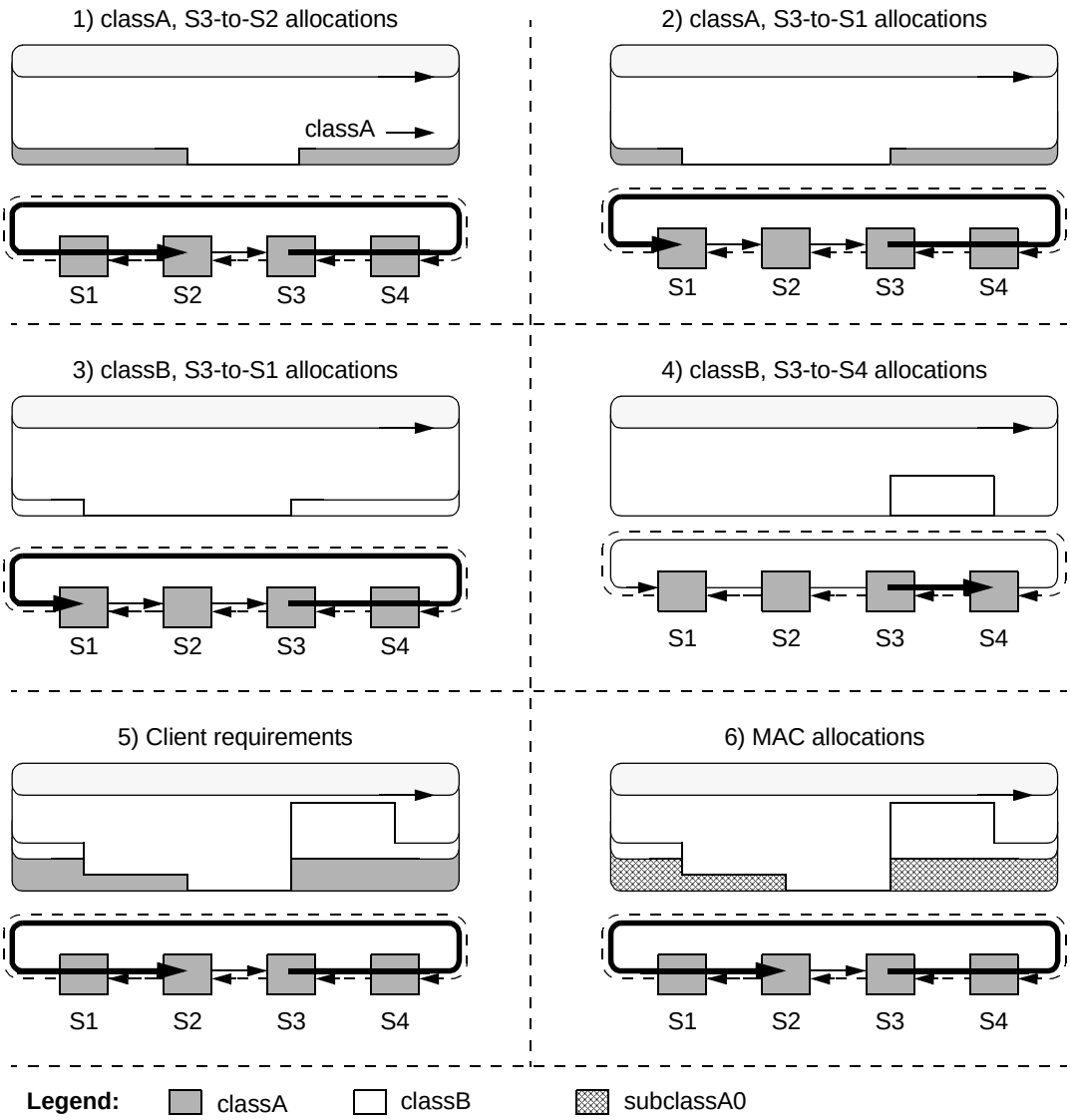


Figure H.1—Single-queue spatial allocation

For each station, the sum of class profiles forms a cumulative bandwidth profile (panel 5) of client-visible classA and classB bandwidth allocations. The classA service (panel 6) consists of only subclassA0 bandwidth, since a second transit queue is necessary to support subclassA1.

H.2.2 Dual-queue spatial allocation

Each station has allocated levels of classA and classB traffic, as illustrated in Figure H.2, which shows allocations for station S4. Spatial allocation of classA traffic makes a distinction between (panel 1) traffic sent from S4-to-S3 and (panel 2) traffic sent from S4-to-S2. Similarly, spatial allocation makes a distinction between classB traffic (panel 3) sent from S4-to-S2 and classB traffic (panel 4) sent from S4-to-S1.

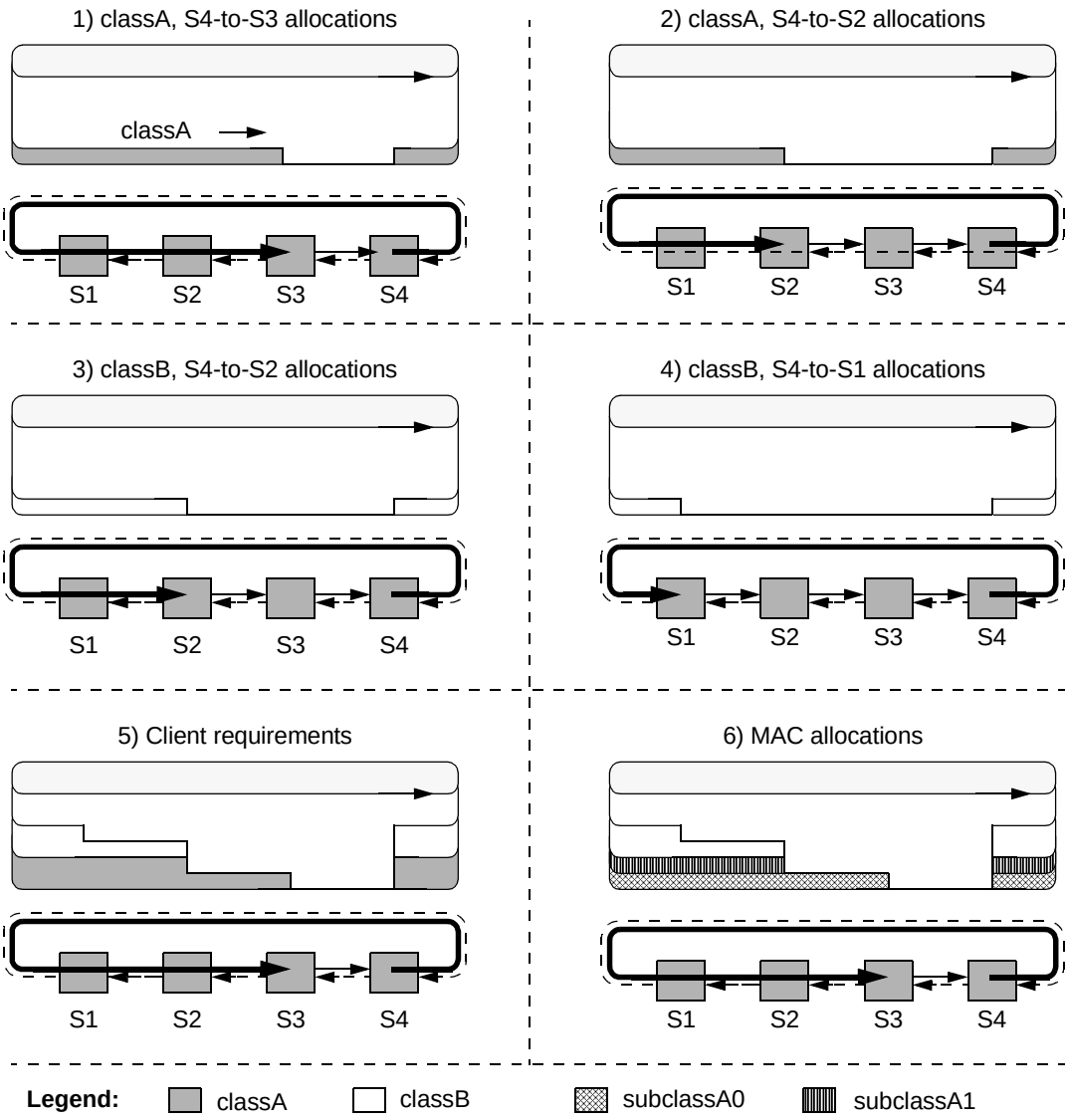


Figure H.2—Dual-queue spatial allocation

For each station, the sum of class profiles forms a cumulative bandwidth profile (panel 3) of client-visible classA and classB bandwidth allocations. The classA service (panel 4) consists of subclassA0 and subclassA1 bandwidths, where the level of supportable subclassA1 bandwidth is proportional to the size of the secondary transit queue.

H.2.3 Cumulative ringlet allocation

Each station has its own (panels 1, 2, 3, and 4) cumulative class profile, as illustrated in Figure H.3. These can be combined (panel 5) into a ringlet class profile. Within consistent ringlet profiles, the sum of subclassA0, subclassA1, and classB profiles (*allocated* in panel 5) shall be less than any individual link capacity.

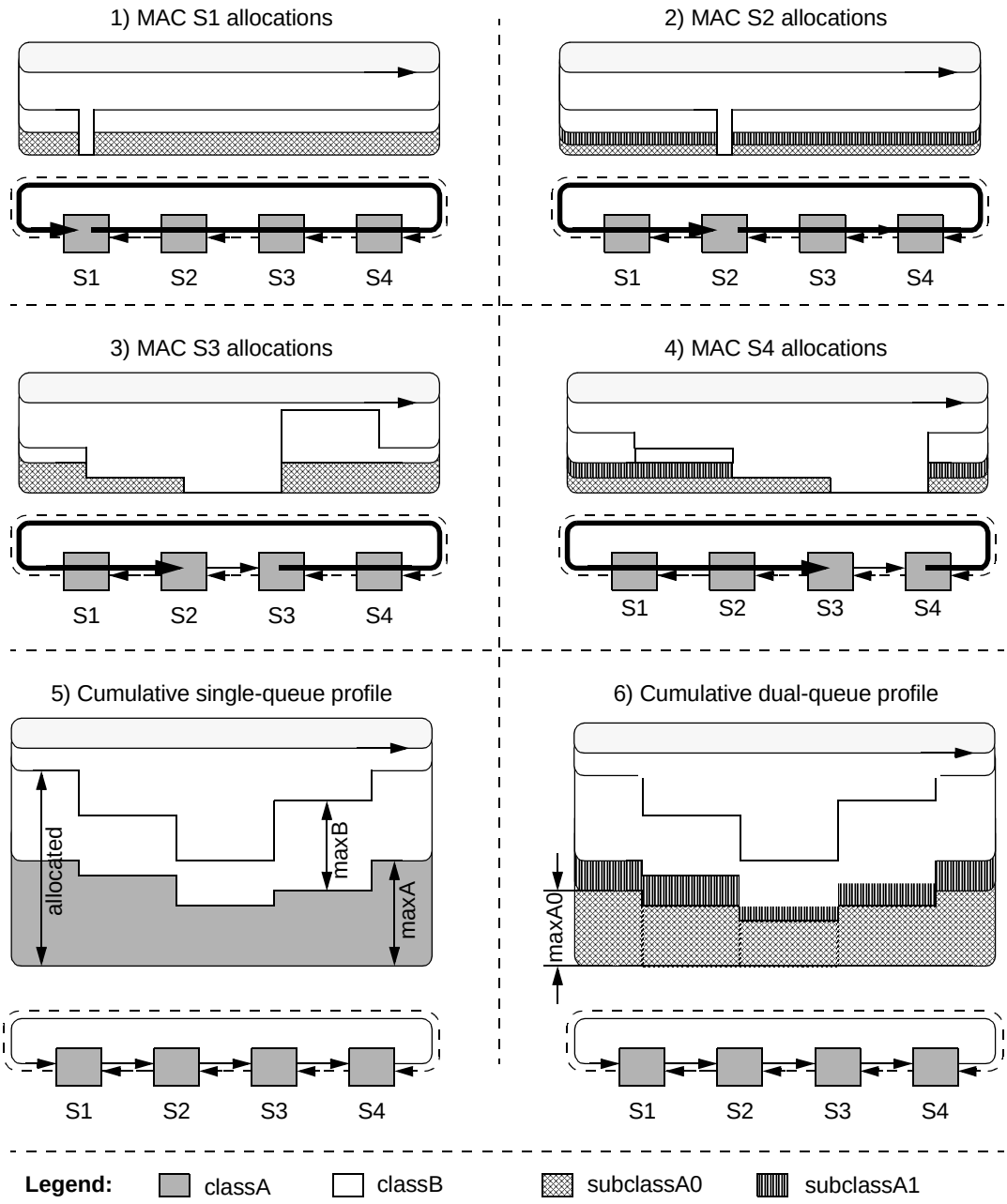


Figure H.3—Cumulative station allocations

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* shall 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*.

H.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 H.1.

Table H.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.

H.4 Spatial shaping

H.4.1 Reclamation

Provisioned bandwidth can be reused, or reclaimed, by a lower priority class when the reclamation does not effect the delay and jitter bounds of the higher priority class(es) on the local station or on any other station on the ring. This subclause provides possible methods a station can use to make a local decision on how to reclaim allocated bandwidth in a such a way that it does not violate the delay and jitter bound properties of other traffic.

Traffic can be sent more than one hop when the local station is VDQ aware and knows that there is no allocation that would be violated on the links subsequent to the first link, and when there is no traffic in the PTQ, regardless of any allocation on the first link. This is because the maximum delay for any PTQ traffic coming into a station is 1 MTU, because no allocation would be violated on the links subsequent to the first link, and because the add frame gets stripped at the destination station, creating an idle space for any PTQ traffic that needs to enter at the destination station.

Traffic can be sent to any destination station when there is no traffic in the PTQ, if the local station has a STQ sufficiently large to hold the maximum queueable traffic, and if the STQ is below its low threshold. This is because the maximum delay for any PTQ traffic coming into the local station is 1 MTU, and because any STQ traffic coming into the local station can be buffered long enough to advise the sender through a fairness message to decrease its rate.

H.4.2 Spatial shaping overview

H.4.2.1 Spatial flow-control

An objective of this standard is to support spatial reuse in the ring, i.e. to provide for efficient link utilization for frame flows with arbitrary (source, destination) pairs.

If the MAC does not allow an independent access rate per destination, the MAC typically has to set a low access rate, to satisfy the bandwidth allocated to one congested destination, which limits the access rates to other uncongested destinations.

Another potential performance limitation is that associated with head of line (HOL) blocking. If the MAC client uses a single FIFO to buffer frames awaiting access on to the ringlet, a frame that is destined to a congested destination (but is at the head of the FIFO) may block transmissions to other uncongested destinations.

This annex addresses these problems through spatial shapers, where each element of the shaper corresponds to a single link or contiguous set of links (called a segment) on the ringlet. This allows the client to support multiple output queues, where each queue represents one segment.

To fully utilize the spatial properties of the ringlet, the MAC and client need to support independent shapers for each ring segment. Furthermore, the classA and classB flow control information supplied by the MAC needs to specify the hop-count distance to the destination.

H.4.2.1.1 Head-of-line blocking limitations of uniform shaping

In the case of RPR, head-of-line blocking involves blocking of a nearby transmission due to congestion at a distant station, as illustrated in the top half of Figure H.4. In this example, station *S1* has two aggregate flows, one destined to station *S5* and another destined to station *S2*. If any link between station *S2* and station *S5* is congested, the later arriving station *S1*-to-*S2* traffic will also be delayed, as though the station *S1*-to-*S2* link were congested.

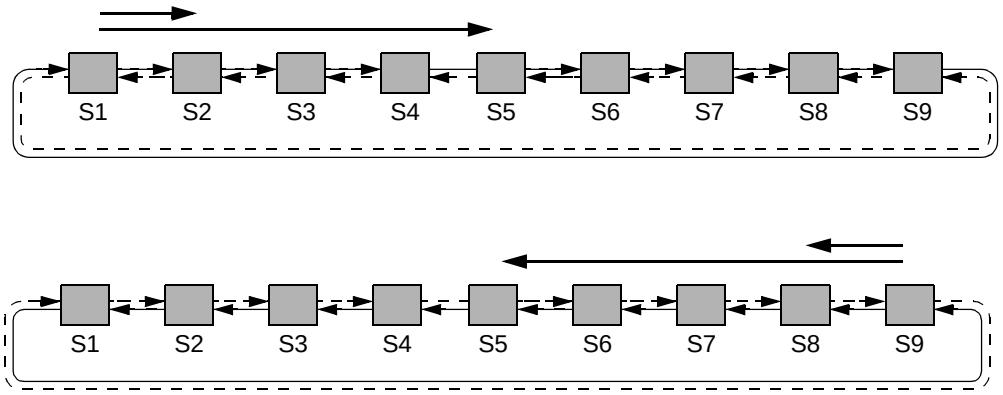


Figure H.4—Head-of-line blocking limitations of uniform shaping

In the example shown in the bottom half of Figure H.4, station *S9* could have two aggregate flows, one destined to station *S5* and the other destined to station *S8*. If any link between station *S8* and station *S5* is congested, the later arriving station *S9*-to-*S8* traffic would also be delayed, as though the station *S9*-to-*S8* link were congested.

H.4.2.1.2 Central-hub limitations of uniform allocation

A simple example of the potential advantages of spatial allocation is provided by a ringlet which supports central-hub accesses, as illustrated in Figure H.5. In this example, a large portion of the accesses on ringlet-0 flow into or out-of the central hub *S5*. The common accesses on ringlet-0 and ringlet-1 are illustrated in the top and bottom halves of Figure H.5, respectively.

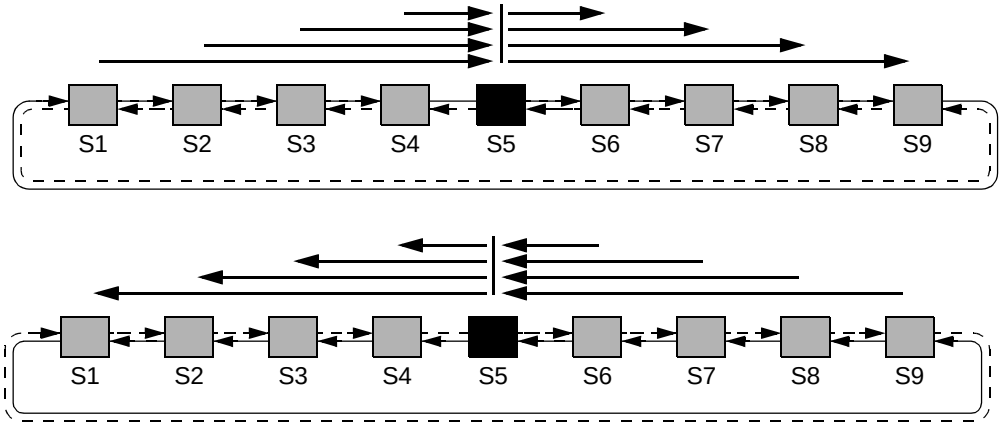


Figure H.5—Central-hub limitations of uniform allocation

With uniform allocation, the cumulative bandwidths of all eight flows on ringlet-0, as well as all eight flows on ringlet-1 are constrained to be less than any link capacity. With spatial allocation, the cumulative bandwidths of any four flows, {S1-to-S5, S2-to-S5, S3-to-S5, S4-to-S5}, {S5-to-S6, S5-to-S7, S5-to-S8, S5-to-S9}, {S5-to-S1, S5-to-S2, S5-to-S3, S5-to-S4}, and {S6-to-S5, S7-to-S5, S8-to-S5, S9-to-S5}, are constrained to be less than any link capacity. Thus, twice the levels of allocated bandwidth can be supported.

H.4.2.2 Spatial flow-control model

Using classA as an example, the hop-count information allows a single logical path (illustrated as the classA layer within Figure H.6) to behave as multiple virtual paths, each with their own go/no-go indication. The *sendA* indication indicates the number of hops before the first shaper-limited per-destination flow. Only those virtual queues before that congested location (in this case addA[0] through addA[2]) are enabled. The virtual queues after the congested location are disabled.

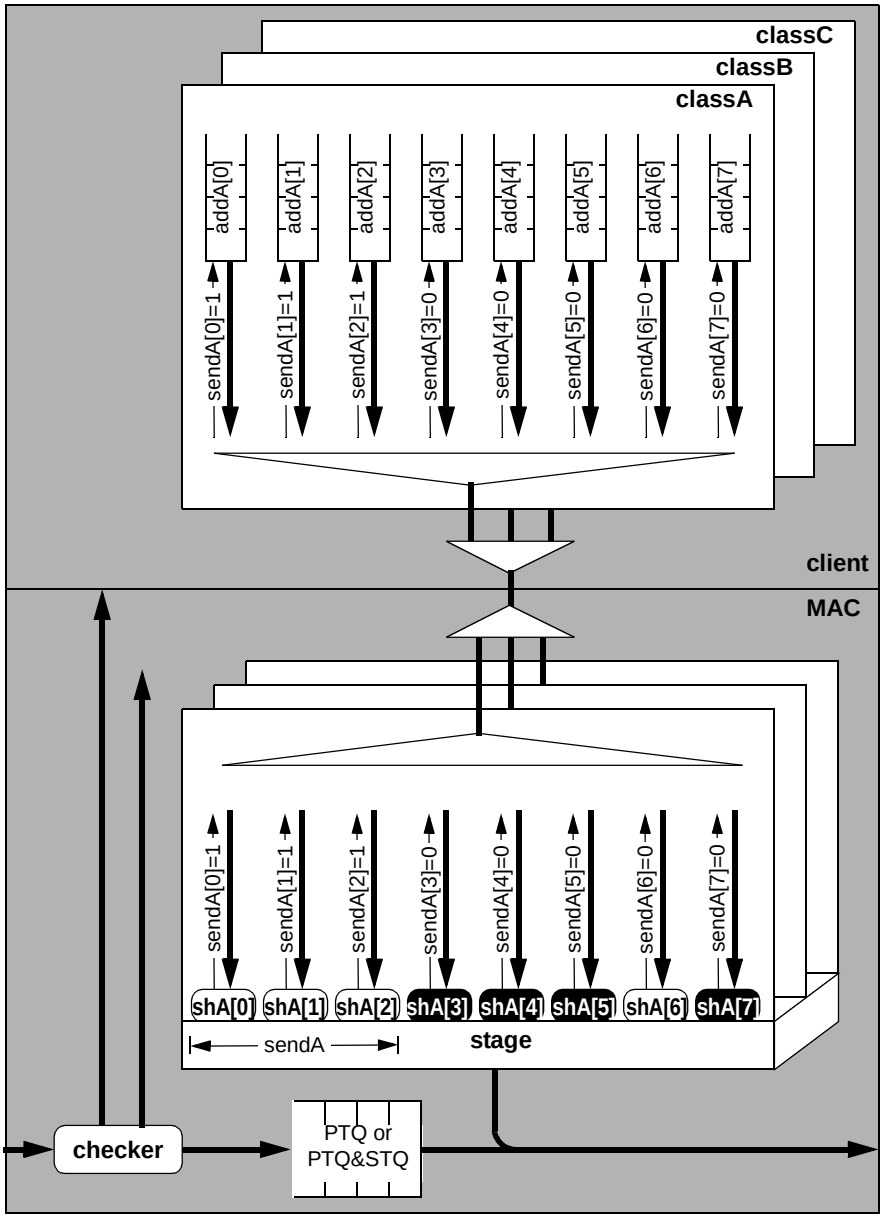


Figure H.6—Spatial flow-control model

To generate the send information within the MAC, arrays of classA, classB, and classC shapers are required, one for each of the relevant segments. The MAC-generated *sendA* indication represents the number of passing shapers (shaded white) before the first of the blocking shapers (shaded black). These shapers are described in the following subclauses.

H.4.3 Spatial shapers

For spatial shaping, each of the uniform shapers described in 6.7 is modified to shape on a per-link basis, as shown in the subclauses below, where *n* refers to each of the n links over which traffic is being shaped.

H.4.3.1 classA spatial shapers

For spatial shaping, the classA shaper descriptions are modified from Table 6.11 to Table H.2, and from Table 6.12 to Table H.3.

Table H.2—shaperA0[n] shaper parameters and results

creditA0[n] adjustment amounts		**creditA0[n] limit amounts**		**passA0[n] result**	
decSize	**incSize**	**hiLimitA0[n] value**	**loLimitA0[n] value**	creditA0[n] >= loLimitA0[n]	creditA0[n] < loLimitA0[n]
clientA0[n]	rateA0[n]*time	MTU + *rateA0[n]* * MAX_JITTER/2	MTU	TRUE	FALSE

Table H.3—shaperA1[n] shaper parameters and results

creditA1[n] adjustment amounts		**creditA1[n] limit amounts**		**passA1[n] result**	
decSize	**incSize**	**hiLimitA1[n] value**	**loLimitA1[n] value**	creditA1[n] >= loLimitA1[n]	creditA1[n] < loLimitA1[n]
clientA1[n]	rateA1[n]*time	MTU + *rateA1[n]* * MAX_JITTER/2	MTU	TRUE	FALSE

The classA spatial shaper output indications, *passA0* and *passA1*, are iterated to determine the maximum number of consecutive hops over which they are TRUE, as illustrated in Figure H.7 and Figure H.8.

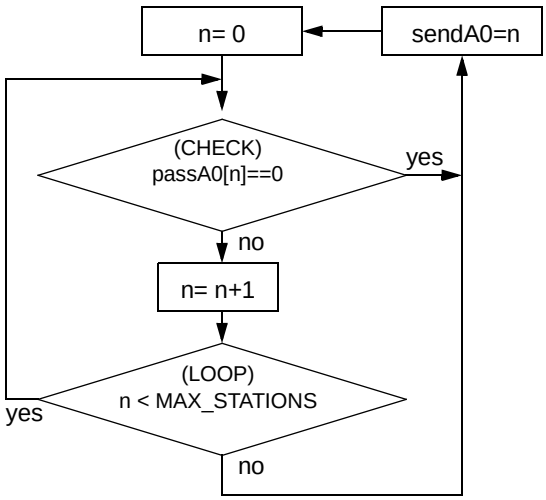


Figure H.7—Spatial sendA0 generation

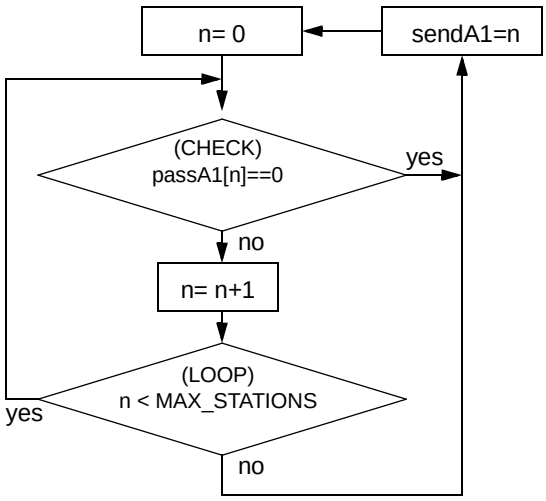


Figure H.8—Spatial sendA1 generation

Exact definitions of the *passA0* and *passA1* iterations are specified in Table H.4 and Table H.5, where rows are evaluated in top-to-bottom order.

Table H.4—Spatial passA0 generation

Last state	Condition	Row	Action	Next state
CHECK	passA0[n]	1	n= n+1	LOOP
	—	2	sendA0= n, n= 0	CHECK
LOOP	n < MAX_STATIONS	3	—	CHECK
	—	4	sendA0= n, n= 0	

Row H.4-1: Checking continues beyond the successful hop count distances.
Row H.4-2: Checking terminates when insufficient spatial *creditA0* shaper credits are discovered.
Row H.4-3: Checking continues up to the maximum hop count distance.
Row H.4-4: The range value is set to the checked hop count distance and the comparisons continue.

Table H.5—Spatial passA1 generation

Last state	Condition	Row	Action	Next state
CHECK	passA1[n]	1	n= n+1	LOOP
	—	2	sendA1= n, n= 0	CHECK
LOOP	n < MAX_STATIONS	3	—	CHECK
	—	4	sendA1= n, n= 0	

Row H.5-1: Checking continues beyond the successful hop count distances.
Row H.5-2: Checking terminates when insufficient spatial *creditA1* shaper credits are discovered.
Row H.5-3: Checking continues up to the maximum hop count distance.
Row H.5-4: The range value is set to the checked hop count distance and the comparisons continue.

The *sendA* generation is then modified from what is specified in Table 6.5 to what is specified in Table H.6, where rows are evaluated in top-to-bottom order.

Table H.6—Spatial sendA indication

Current state		Row	Next state	
state	condition		action	state
CHECK	stageQueue can accept client frame	1	sendA = MAXIMUM(sendA0, MINIMUM(sendA1, sendD));	CHECK
	—	2	sendA = 0;	

Row H.6-1: Setting *sendA* to the maximum of the *sendA0* and *sendA1* (MINIMUMed with the *sendD*) values enables throttled classA transmissions when the stage queue and the PHY are available. shaperA provides the *sendA* permission indication based on the furthest hop count allowed for client-supplied classA traffic.
Row H.6-2: For store and forward stage queue implementations, at least one MTU of stage-queue storage is required to safely buffer client-supplied frames. For cut through stage queue implementations, the logical stage queue must be capable of immediately transmitting the client frame.

H.4.3.2 classB spatial shaper

For spatial shaping, the classB shaper description is modified from Table 6.13 to Table H.7.

The classB spatial shaper output indication, *passB*, is iterated to determine the maximum number of consecutive hops over which it is TRUE, as illustrated in Figure H.9.

Table H.7—shaperB[n] shaper parameters and results

creditB[n] adjustment amounts		creditB[n] limit amounts		passB[n] result	
decSize	**incSize**	**hiLimitB[n] value**	**loLimitB[n] value**	creditB[n] >= loLimitB[n]	creditB[n] < loLimitB[n]
clientB[n]	rateB[n]*time	(2*N*MTU+RTT)*R	MTU	TRUE	FALSE

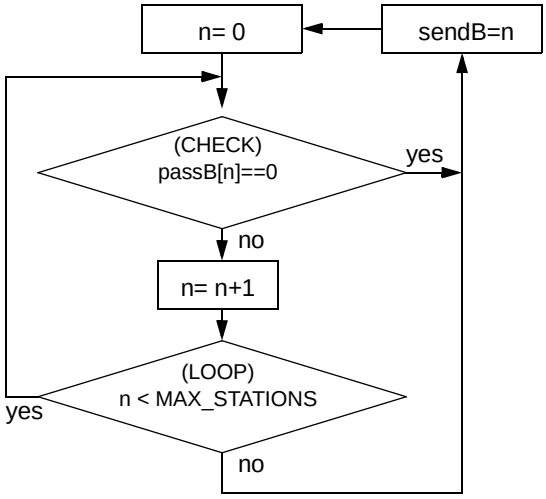


Figure H.9—Spatial sendB generation

An exact definition of the *passB* iteration is specified in Table H.8, where rows are evaluated in top-to-bottom order.

Table H.8—Spatial passB generation

Last state	Condition	Row	Action	Next state
CHECK	passB[n]	1	n= n+1	LOOP
	—	2	sendB= n, n= 0	CHECK
LOOP	n < MAX_STATIONS	3	—	CHECK
	—	4	sendB= n, n= 0	

Row H.8-1: Checking continues beyond the successful hop count distances.
Row H.8-2: Checking terminates when insufficient spatial *creditB* shaper credits are discovered.
Row H.8-3: Checking continues up to the maximum hop count distance.
Row H.8-4: The range value is set to the checked hop count distance and the comparisons continue.

The *sendB* generation is then modified from what is specified in Table 6.6 to what is specified in Table H.9, where rows are evaluated in top-to-bottom order.

Table H.9—Spatial sendB indication

Current state		Row	Next state	
state	**condition**		**action**	**state**
CHECK	stageQueue can accept client frame	1	sendB = MINIMUM(sendB, sendD);	CHECK
	—	2	sendB = 0;	

Row H.9-1: Setting *sendB* to the minimum of the *sendB* and the *sendD* values enables throttled classB transmissions when the stage queue and the PHY are available. shaperB provides the *sendB* permission indication based on the furthest hop count allowed for client-supplied classB traffic.
Row H.9-2: For store and forward stage queue implementations, at least one MTU of stage-queue storage is required to safely buffer client-supplied frames. For cut through stage queue implementations, the logical stage queue must be capable of immediately transmitting the client frame.

H.4.3.3 Fairness eligible spatial shapers

There is no spatial fairness algorithm, so the classC and classCc shaper descriptions are not modified from Table 6.14 and Table 6.15, respectively.

H.4.3.4 Downstream spatial shaper

For spatial shaping, the downstream shaper description is modified from Table 6.16 to Table H.10.

Table H.10—shaperD[n] shaper parameters and results

creditD[n] adjustment amounts		creditD[n] limit amounts		passD[n] result	
decSize	**incSize**	**hiLimitD[n] value**	**loLimitD[n] value**	creditD[n] >= loLimitD[n]	creditD[n] < loLimitD[n]
!clientA0[n]	unreservedRate[n]*time	MTU	MTU	TRUE	FALSE

The classD spatial shaper output indication, *passD*, is iterated to determine the maximum number of consecutive hops over which it is TRUE, as illustrated in Figure H.10.

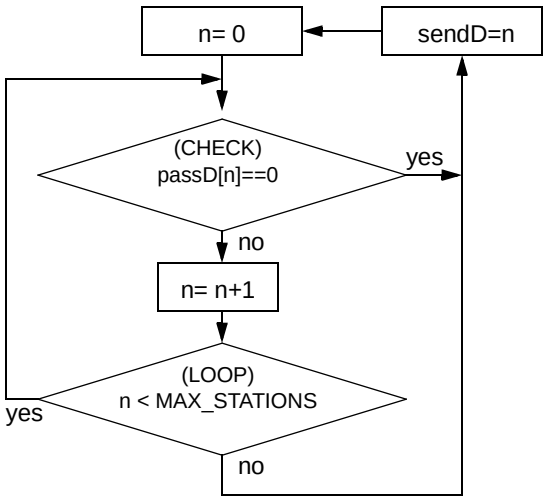


Figure H.10—Spatial sendD generation

An exact definition of the *passD* iteration is specified in Table H.11, where rows are evaluated in top-to-bottom order.

Table H.11—Spatial passD generation

Last state	Condition	Row	Action	Next state
CHECK	passD[n]	1	n= n+1	LOOP
	—	2	sendD= n, n= 0	CHECK
LOOP	n < MAX_STATIONS	3	—	CHECK
	—	4	sendD= n, n= 0	

Row H.11-1: Checking continues beyond the successful hop count distances.
Row H.11-2: Checking terminates when insufficient spatial *creditD* shaper credits are discovered.
Row H.11-3: Checking continues up to the maximum hop count distance.
Row H.11-4: The range value is set to the checked hop count distance and the comparisons continue.

Annex I

(informative)

Data path scenarios

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 0.1, February 2002	Initial draft document for RPR WG review.
Draft 0.2, April 2002	No changes.
Draft 0.3, June 2002	Added Class A/B shaping information.
Draft 1.0, August 2002	Modified according to comments on D0.3.
Draft 1.1, October 2002	Modified according to comments on D1.0.
Draft 2.0, December 2002	Separated from Annex I of D1.1 and modified for WG ballot.

This annex provides information about possible adverse scenarios on the data path. To better understand the implications of possible approaches to different data path 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 its recirculates beyond its virtual source, as illustrated by the *x* mark within Figure I.1.

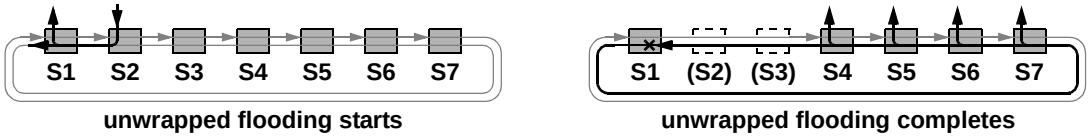


Figure I.1—Duplicate scenario 1: Unidirectional source bypass

Cause: The source (that was responsible for frame deletion) disappears before its frame returns.
Problem: The frame passing through stations *S3*&*S2* may be falsely accepted by station *S1* (and others).
Solution: Station S1 discards frames based on *(frame.WS==0)* check passing. We will only return-wrap if the frame has been marked by the source station.

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.

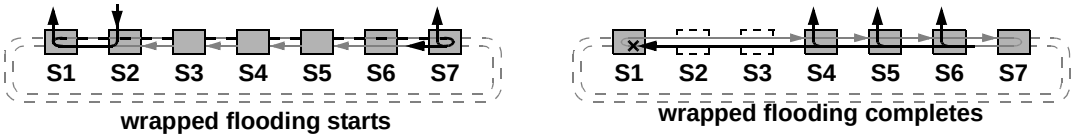


Figure I.2—Duplicate scenario 2: Unidirectional wrapped source bypass

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: wrapStatus 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.1. In this example, destination stations *S5*&*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.

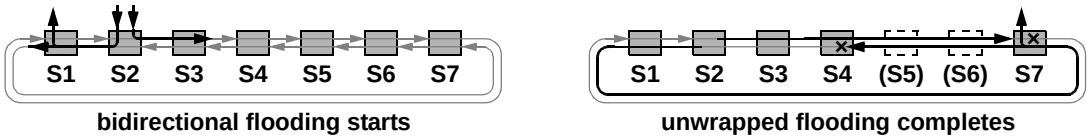


Figure I.3—Duplicate scenario 3: Bidirectional destination bypass

Cause: The destination (that was responsible for frame deletion) disappears before its frame arrives.
Problem: The frame passing through stations *S5*&*S6* may be falsely duplicated at station *S4*, *S7*, and others.
Solution: Frame discarded at S4 and S7 since *(SRC[ttlBase-timeToLive]!=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*&*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.

Cause: The destination (that was responsible for frame deletion) disappears before its frame arrives.
Problem: The frame wrapped before stations *S5*&*S6* may be falsely duplicated at station *S4*, *S7*, and others.
Solution: S7 and S4 will discard frame based on *(SRC[ttlBase-timeToLive]!=frame.SA)* for received ringlet check.

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*,

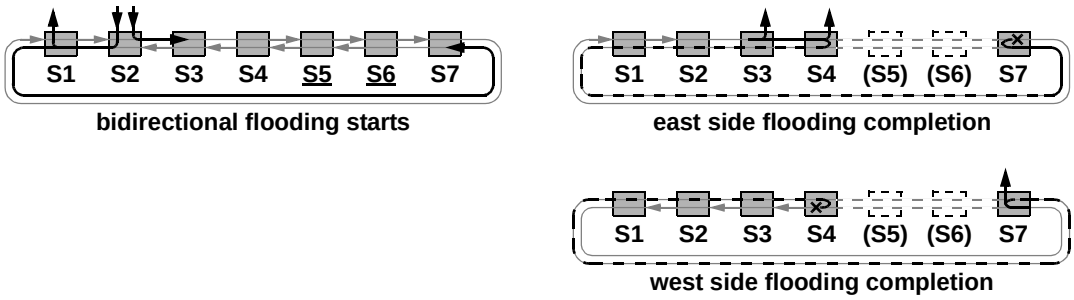


Figure I.4—Duplicate scenario 4: Bidirectional destination removals

S7, and *S8* are removed while the *S2*-sourced frame is circulating. Correct processing involves discarding returning frames when their source is missed.

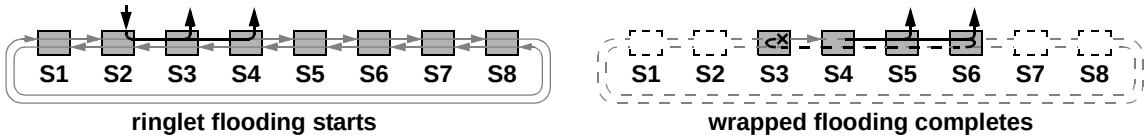


Figure I.5—Duplicate scenario 5: Source and destination removals

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: Frame discarded at S3 the second time around due to *(SRC[ttlBase-timeToLive]!=frame.SA)* check.

I.2 Reordered frame scenarios

I.2.1 Protection switch during bidirectional flood

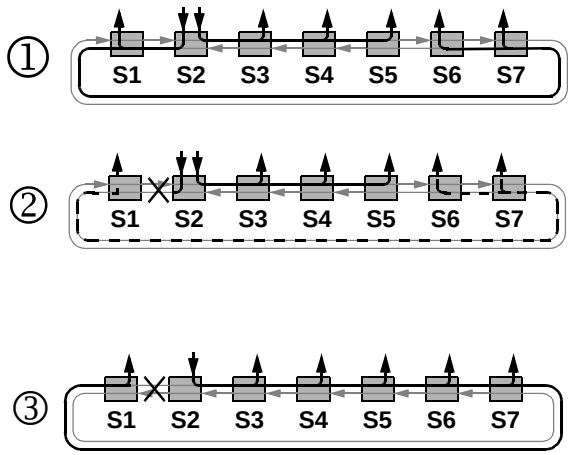


Figure I.6—Reorder scenario 1: Protection switch during bidirectional flood

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.

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.

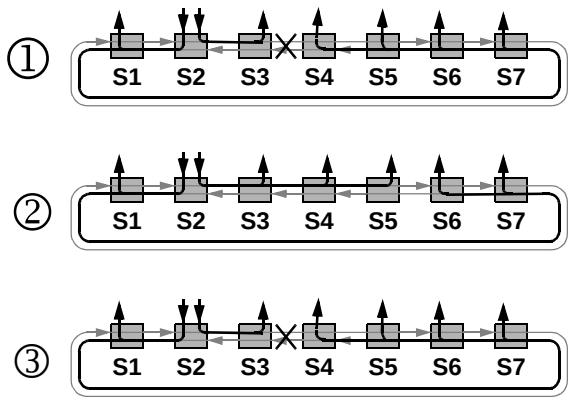


Figure I.7—Reorder scenario 2: Cascading failures during bidirectional flood

Cause: At step 1, consider frames in-flight on ringlet1 (using context #1). At step 2, 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 step 3, 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 S1and S2, causing station S2 to dispatch the unicast traffic destined to S6 over ringlet0.

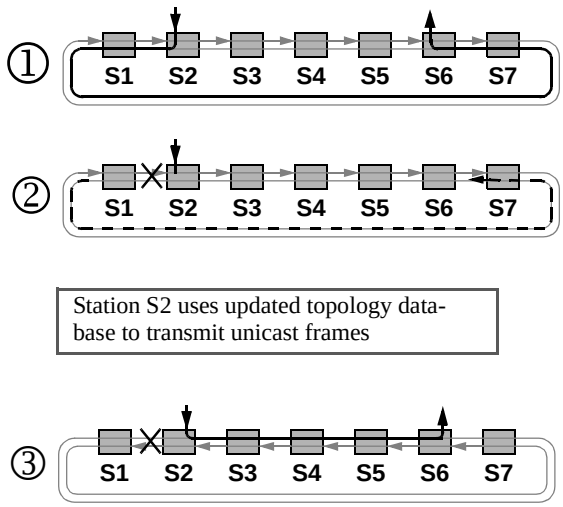


Figure I.8—Reorder scenario 3: Protection switching during unicast transmission

Cause: At step 1, consider frames in-flight on ringlet1 (using context #1). At step 2, 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. Unicast frames destined to S6 now 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

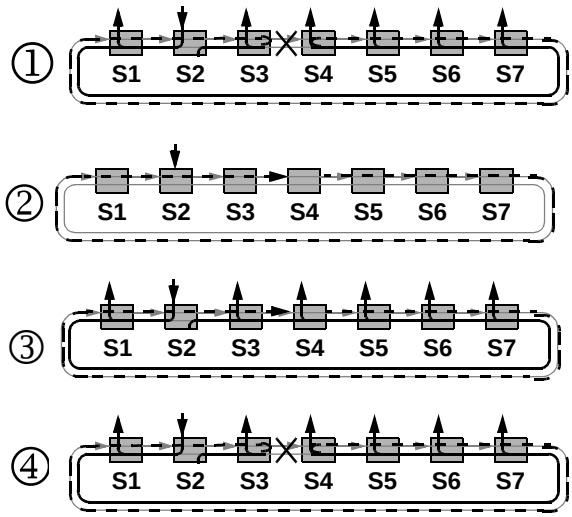


Figure I.9—Reorder scenario 4: Cascading failures during unidirectional flood

Cause: At step 1, consider wrapped frames on secondary ringlet. At step 2, the ring heals, however the frames on secondary ringlet continue to circulate until *timeToLive* expires. During frame circulation on secondary ringlet, frames are transmitted by station S2 (as shown in step 3). If another failure occurs (at step 4), prior to circulation of frames on secondary ringlet having their *timeToLive* expire, frames launched at step 3 can be received by station S3 before un-expired 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!=side.RI* for a specified duration. All wrapped strict data frames using an outdated context will be removed from the ring.

I.3 Fairness scenarios

Weighted fairness with spatial reuse is a goal of the RPR protocol. The MAC fairness model regulates the flow aggregates from a given ingress station, while allowing the client to regulate independent end-to-end flows within that aggregate. Flow aggregates may be sub partitioned by the client, based on their destination station, to ensure maximal spatial reuse opportunities.

To better understand the implications of these fairness strategies, multiple traffic-load scenarios are illustrated. For each scenario, the desired and possible problematic behaviors are defined.

The following scenarios demonstrate the aggregate flow rates, expressed as a share of the total capacity. In each case, a tandem segment of a ring is depicted. The input traffic is assumed to be constant rate; if not otherwise specified, the offered input rate equals the link capacity for each depicted flow.

Within the scenario illustrations, the percentage number associated with a flow represents the percent of the nominal link bandwidth that is consumed by that flow. The numbers above the flow line correspond to the desired bandwidth (from a theoretical perspective); the numbers below the flow line correspond to the bandwidths predicted for a suboptimal implementation.

I.3.1 Parking lot

The first parking lot scenario illustrates the desire to avoid overly throttling one source over another, based on the distance of the source from the most congested (choke-point) link, as illustrated in Figure I.10 and Figure I.11. The objective is to evenly partition the available choke-point bandwidth between each of the contending upstream stations.

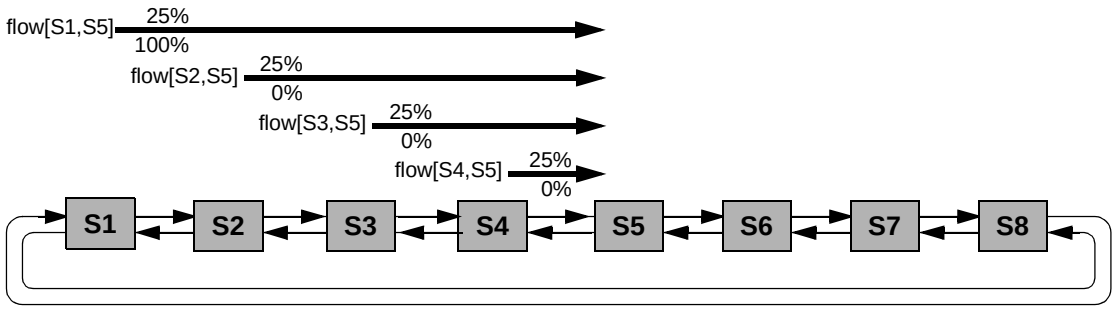


Figure I.10—Fairness scenario 1a: Parking lot

Concern: Station S1 consumes the entire link bandwidth, due to transit-queue transmission precedence.
Applicable: This problem is addressed by single-choke and multi-choke fairness algorithms.
Solution: Throttle upstream stations with congestion information sent from downstream stations.

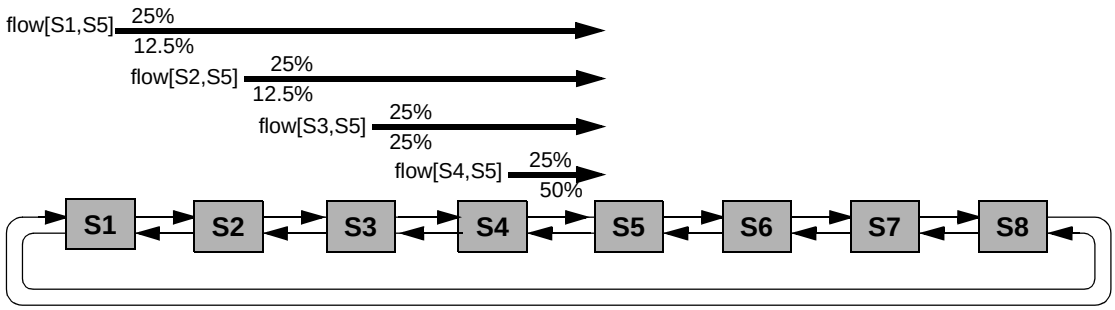


Figure I.11—Fairness scenario 1b: Parking lot

Concern: Station S1 traffic is overly restricted, due to others' alternate selections of transit-queue and stage-queue transmissions. This is an example where local fairness does not ensure global fairness.
Applicable: This problem is addressed by single-choke and multi-choke fairness algorithms.
Solution: Communicate global knowledge by passing congestion information beyond the upstream station.

I.3.2 Parallel parking lot

The second parking lot scenario illustrates the desire to support spatial reuse of sub aggregate flows, as illustrated in Figure I.12. The objective is to evenly partition the available choke-point bandwidth between each of the contending upstream stations, without throttling nonconflicting flows.

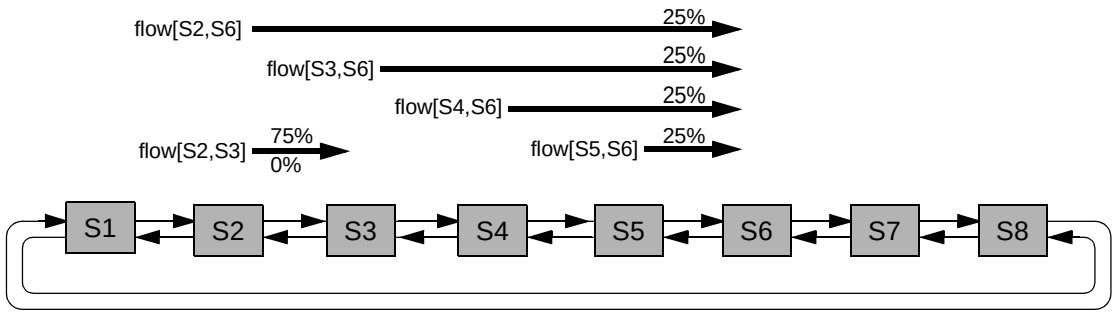


Figure I.12—Fairness scenario 2: Parallel parking lot

Concern: The S2-to-S3 transmissions are throttled by S5-to-S6 congestion indications.
Applicable: Occurs when the hop-count and congestion level are not both reported to the client.
This problem is addressed by single-choke and multi-choke fairness algorithms.
Solution: Two capabilities are required to avoid near-side starvation:
a) The MAC provides hop-count as well as congestion-level information to the client.
b) The client provides distinct virtual destination queues (VDQs) for several hop-count distances.

I.3.3 Upstream parallel parking lot

The next parking lot scenario illustrates the desire to support spatial reuse of nonconflicting flows, as illustrated in Figure I.13. The objective is to evenly partition the available choke-point bandwidth between each of the contending upstream stations, without throttling nonconflicting pass-through flows.

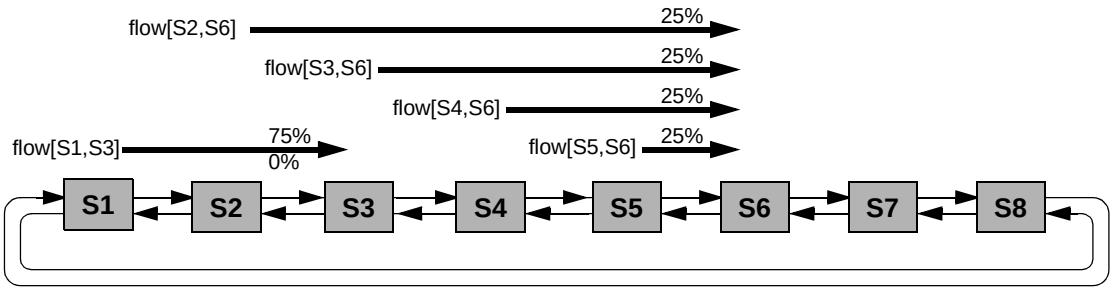


Figure I.13—Fairness scenario 3: Upstream parallel parking lot

Concern: The S1-to-S3 transmissions could be unnecessarily throttled.
This topology also exhibits the oscillatory behaviors of scenario 7 (see I.3.7).
Applicable: Occurs when the hop-count and congestion level are not both reported to the client.
This problem is addressed by single-choke and multi-choke fairness algorithms.
Solution: Two capabilities are required to avoid near-side starvation:
a) The MAC provides hop-count as well as congestion-level information to the client.
b) The client provides distinct virtual destination queues (VDQs) for several hop-count distances.

I.3.4 Multi-flow parking lot

This parking lot scenario illustrates the effect of supporting weighted aggregate flows, as illustrated in Figure I.14. The objective is to evenly partition the available choke-point bandwidth between each of the contending upstream stations' aggregate flows, rather than equally weighting individual flows within the aggregate.

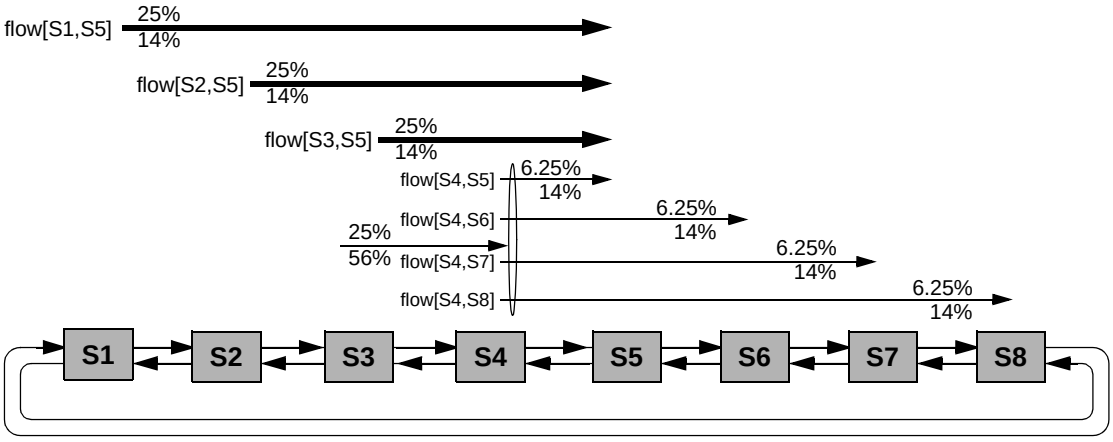


Figure I.14—Fairness scenario 4: Multi-flow parking lot

Concern: Insufficient information is available for the client to allocate per-destination flow bandwidths.
This could be the effect of using an on-off MAC-to-client flow-control indication.

Applicable: This appears to not occur with either single-choke or multi-choke fairness.
Solution: Provide a hop-count based congestion indication to the client.

I.3.5 Dual-exit parking lot (multiple choke points)

This dual parking lot scenario illustrates an effect of having multiple choke points, as illustrated in Figure I.15. The objective is to evenly partition the available choke-point bandwidth based on primary as well as secondary choke-point locations, so that station S2 and station S3 evenly distribute the 40% residual bandwidth available on the S4-to-S5 link.

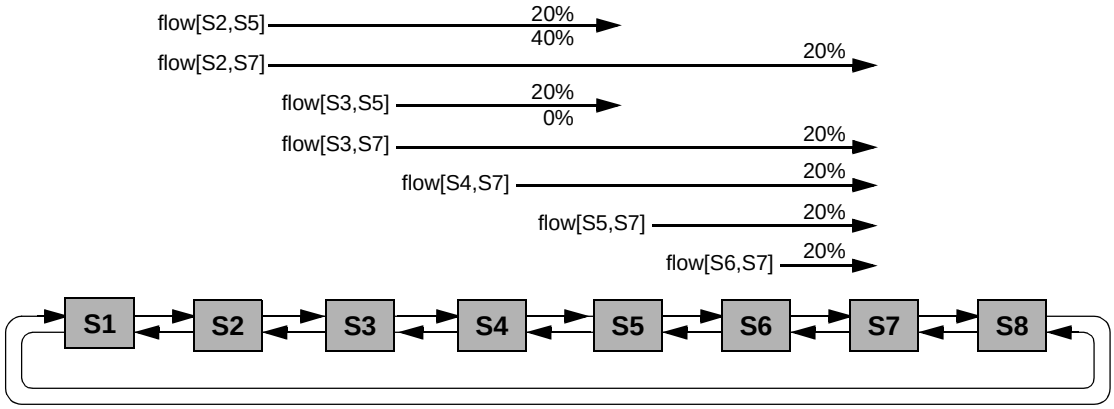


Figure I.15—Fairness scenario 5: Dual-exit parking lot (multiple choke points)

Concern: Flows into the less congested S4-to-S5 segment observe only the most-congested indication from station S7.
Applicable: This appears to happen with single-choke but not multi-choke fairness.
Solution: Each station must be aware of all choke-point conditions, not just the worst congestion point.

I.3.6 Migrating choke point

This migrating choke-point scenario illustrates another effect of having multiple choke points, as illustrated in Figure I.16. In this example, station S5 is the original choke point that forces bandwidth to be divided equally between flow[S1,S8], flow[S2,S8], flow[S4,S7], and flow[S5,S6].

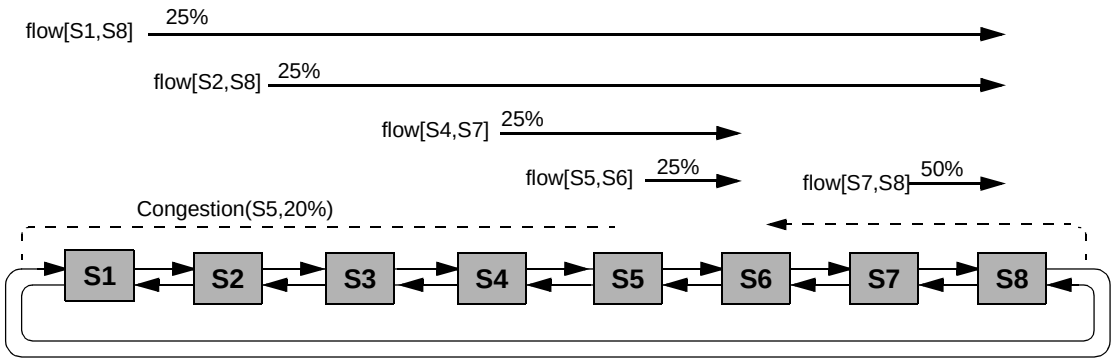


Figure I.16—Fairness scenario 6a: Stable station S5 choke point

Reductions in flow[S5,S6] and flow[S4,S7] cause station S5 to deassert its congestion indication, as illustrated in Figure I.17. A stale version of that assertion remains observable to station S7, inhibiting station S7's assertion of its less-congested indication. Thus, nearly a full ringlet circulation time can pass between:

a) The sensing of a not-congested indication at S1&S2 and
b) The reassertion of a station-S7-asserted congestion indication.

During that time, flow[S1,S8] will be momentarily unconstrained before flow[S2,S8] congestion reduces its flow (not illustrated); station S7 remains starved.

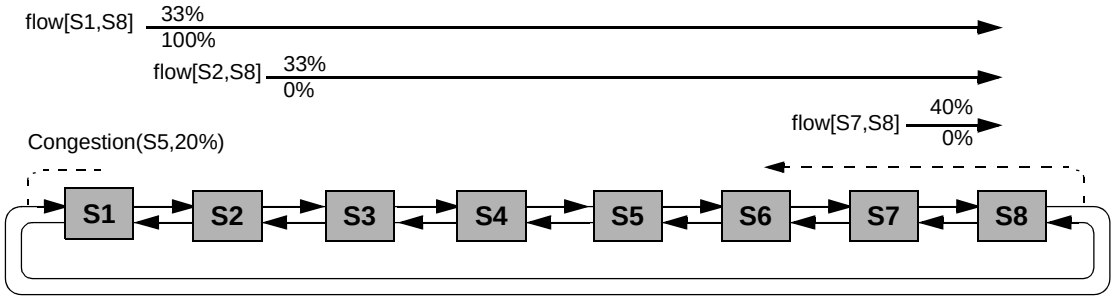


Figure I.17—Fairness scenario 6b: Migrating choke point

The transient condition will eventually stabilize and normal fairness algorithms will apply, as illustrated in Figure I.18.

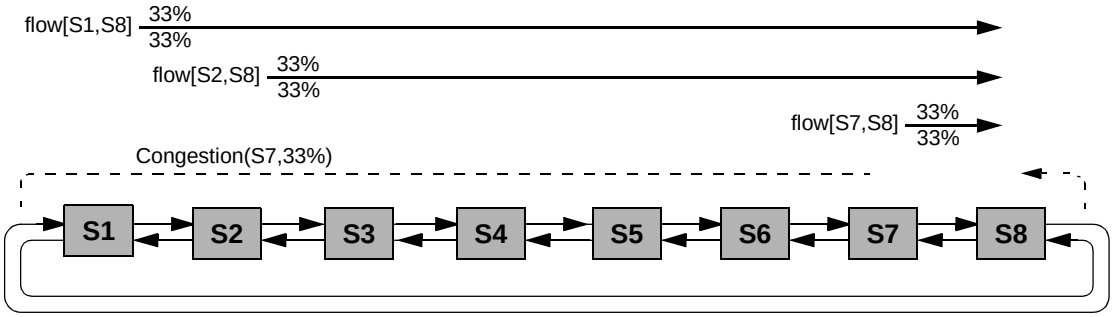


Figure I.18—Fairness scenario 6c: Stable station S7 choke point

Concern: The congestion indication temporarily vanishes when station S5 congestion is relieved, as illustrated in Figure I.17. This occurs during the distributed computation of the new most-choked indication.
Applicable: Addressed by both single-choke and multi-choke fairness.
Solution: The solution differs for single-choke and multi-choke fairness:
a) Single-choke. A slow ramp time avoids excessive flow[S1,S8] and flow[S2,S8] transmissions. This is more effective when the ramp time more than exceeds the ringlet-circulation time.
b) Multi-choke. Each station has knowledge of all congestion conditions, allowing the worst-case congestion location(s) to be determined without this worst-case distributed-congestion computation glitch.

I.3.7 Choked high/low bandwidth pairs

This scenario illustrates an effect of having time-delayed transmission of flow-control indications, as illustrated in Figure I.19. In this scenario, station S1 is a high bandwidth source trying to send steady traffic at the highest possible rate; station S5 is lower-bandwidth source providing an offered load of approximately 5% of the link-bandwidth.

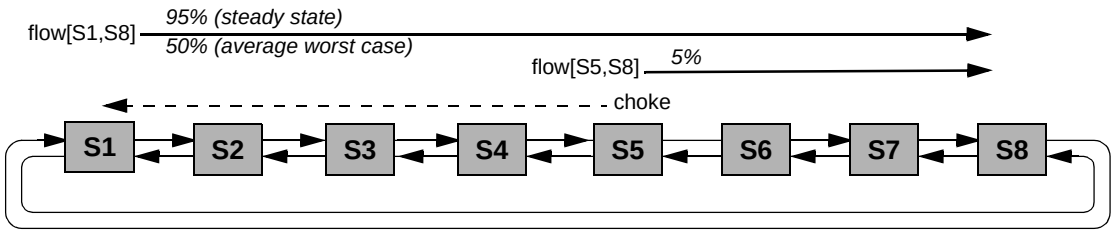


Figure I.19—Fairness scenario 7: Choked high/low bandwidth pairs

This scenario can produce oscillatory flow conditions, as illustrated in Figure I.20. At time t_A, flow[5,8] is active in the absence of upstream-flow conflicts; flow[1,8] is stopped in the absence of a station-S5 sourced choke indication. At time t_B, flow[5,8] is stopped due to upstream-flow conflicts; flow[1,8] is active in the absence of a station-S5 sourced choke indication. At time t_C, flow[5,8] is stopped due to upstream-flow conflicts; flow[1,8] is active in the absence of a station-S7 sourced choke indication. At time t_D, flow[5,8] is active in the absence of upstream-flow conflicts; flow[1,8] is active in the absence of a station-S7 sourced choke indication. This cycle repeats indefinitely.

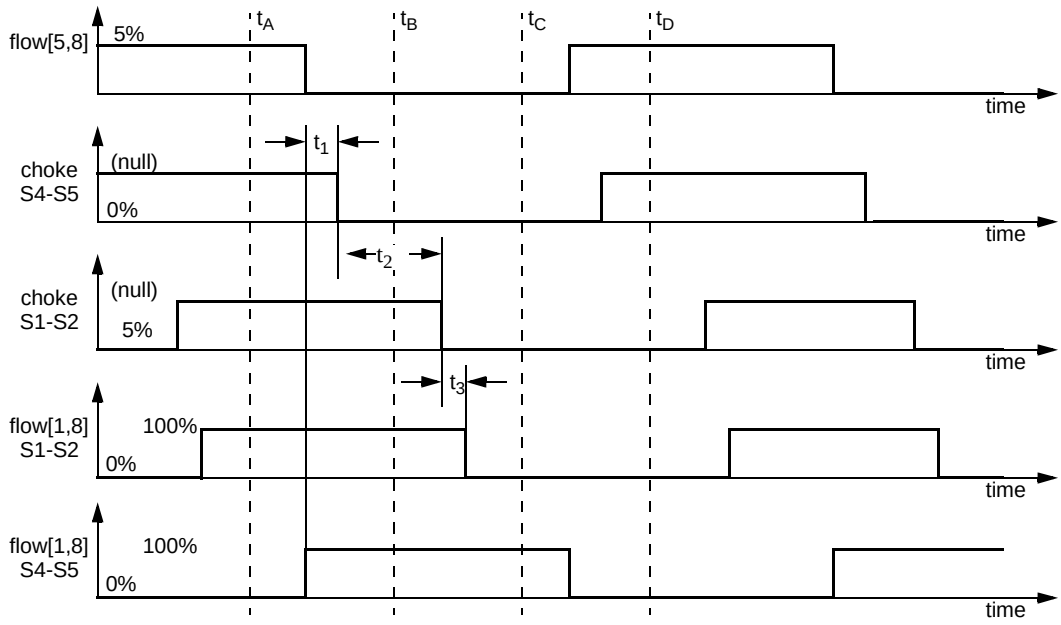


Figure I.20—Fairness scenario 7a: Oscillatory flows & indications, fast ramp

Within Figure I.20: time ***t1*** represents the station S5 delay between sensing and reporting of congestion; time ***t2*** is the propagation delay between S5 and S1; time ***t3*** is S1's delay between receiving S5's congestion condition and inhibiting the excessive station S1 data transmissions.

Concern: The flows from station S1 are unnecessarily and severely limited in a cyclical fashion.
Applicable: This could happen with either single-choke or multi-choke fairness protocols.
The problem exists with mono-queue and dual-queue designs, with distinct t1 time parameters:
a) For a mono-queue design, t1 is based on the rate-averaging interval.
b) For a dual-queue design, t1 is based on the congestion-depth threshold of the STB.
Solution: Setting the weights proportionally to the sourced traffic mitigates the problem. Additionally, a multi-choke fairness solution would be to provide linear flow-rate and congestion levels, to support a more linear (as opposed to on-off) feedback control mechanism.

A longer single-choke ramp-up can avoid problems associated with round-trip oscillations. However, the ramp-up protocols exhibit their own oscillatory behaviors, as illustrated in Figure I.20. In this case, station S1 is periodically shut off by station S5 and has to ramp up slowly.

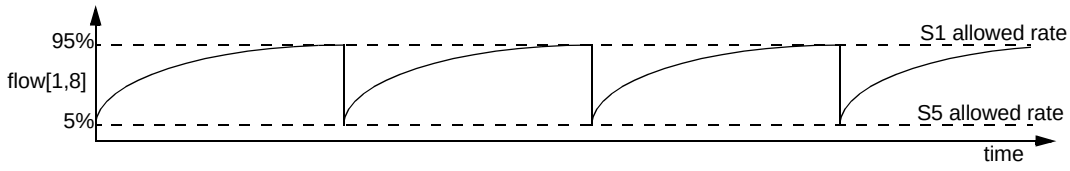


Figure I.21—Fairness scenario 7b: Oscillatory flows & indications, slow ramp

I.3.8 Rotating choked pairs

The rotating choked pairs scenario illustrates an effect of measuring transient rates, as opposed to overall byte transfers, as illustrated in Figure I.22. In this scenario, two pairs of overlapping high/low transmission segments (as described in I.3.7) are present. The high/low S2-to-S4 and S5-to-S7 segments produce cyclical choke-point indications, *choke3* and *choke6* respectively.

The objective is provide station S1 with a flow[S1,S8] allowance equal to the average time-averaged flows from station S2 and station S5.

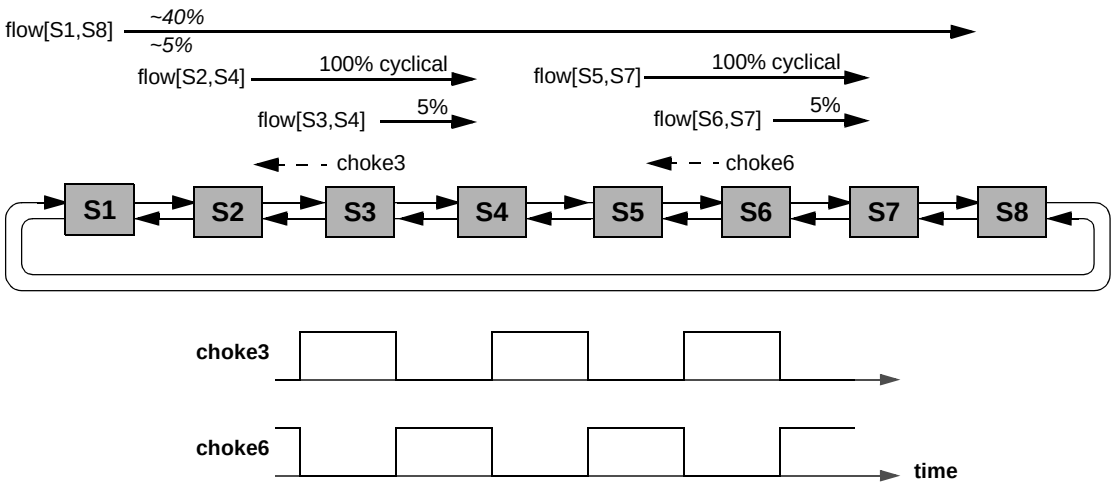


Figure I.22—Fairness scenario 8: Rotating choked pairs

Concern: Station1 is unnecessarily throttled due to constant observation of a 5% worst-case choke indication, based on alternate observations of choke3 or choke6 conditions.

Applicable: Could occur with single-choke fairness protocols.
Solution: Base flow progress on total-bytes-transferred, not an interval-dependent bytes-per-second rate. The total-bytes-transferred is more easily supported by multi-choke fairness.

Annex J

(informative)

Topology discovery and protection scenarios

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 1.1, October 2002	Introduced as a result of comments from D1.0. Modifications made: Addition of diagnostic scenarios section Minor cleanup of existing scenarios
Draft 2.0, December 2002	Addition of basic failure examples section (moved from Clause 11) Addition of operator commanded scenarios section Minor cleanup of existing scenarios for consistency with Clauses 10 and 11

Editors' Notes: *To be removed prior to final publication.*

The goal of this annex is to illustrate the key scenarios for topology discovery and protection switching. These include (for example) station initialization, addition of a station, removal of a station, isolation of a station, handling of bandwidth allocation, and handling of misconfigurations. Description will be included as to how topology interacts with protection and fairness.

J.1 Overview

This clause describes a variety of scenarios relevant to topology discovery and protection switching. The purpose of the scenarios is to clearly illustrate how the topology discovery and protection switching protocols respond to various events within an RPR ring.

J.2 Topology database generation

This subclause contains the following scenarios relating to the generation of the topology database at the stations making up an RPR ring:

1) Topology database generation in an RPR ring containing a single failed link.
2) Topology database generation in an RPR ring containing a station with two failed transmit links.
3) Topology database generation in an RPR ring containing a station with two failed receive links.
4) Topology database generation in an RPR ring containing a station with one failed receive link and one failed transmit link that are part of different spans.
5) Topology database generation in an RPR ring containing a station with two failed receive links and two failed transmit links.

J.2.1 Single failed link

In this scenario, the link from S1 to S2 goes into signal fail state. Before the link went into signal fail state, S2 could determine that S1 was its west neighbor. After S2 detects a signal fail on its west interface, S2 clears S1 as its west neighbor but continues to report S3 as its east neighbor to the rest of the ring via the topology message. The notation TS{S2, 0, S3} indicates that station S2 reports that its west neighbor is unknown, with zero MAC address. Stations can also see that the link from S1 to S2 has gone into signal fail state based on receipt of a protection message from S2.

On the other hand, S1 still detects S2 as its neighbor on its east interface. S1 will continue to report S2 as its east neighbor.

At all stations, the failure of the link from S1 to S2 results in modification of the topology database based on the fact that the span connecting S1 to S2 is no longer usable for data. The distance from each station to a given station contained in its topology database is maintained if a downstream path for data is maintained. For example, Figure J.1shows that stations S0, S3, and S2 are reachable downstream from station S1 with hop count distances 1, 2, and 3 hops, respectively.

S2 reports a receive link status of signal fail on its west interface to the rest of the ring. Though the span connecting S1 and S2 is not usable for data, S1 reports a receive link status of idle on its east interface.

All stations continue to be reachable from all other stations for data, as denoted by the entries in the tables marked with a Y for yes.

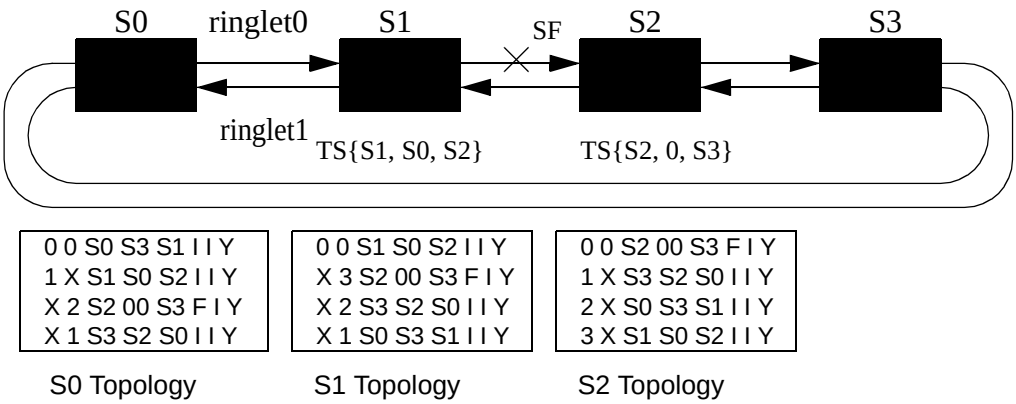


Figure J.1—Single egress link failure.

J.2.2 Two failed transmit links

Editors' Notes: *To be removed prior to final publication.*

This scenario needs further discussion. Based on the rule currently defined in 11.7.1.3 that topology messages shall be stripped from the ring by a station in wrapping state, the discovered topology in this case results in no messages from S3 being forwarded to S1, even though S0 and S2 refer to S3 as being one of their neighbors. Currently there is no such stipulation in 11.7.1.3 for steering rings, and as a result S3 would be visible to S1 for steering rings. The information visible in steering rings but not visible in wrapping rings is shown in italics.

In this scenario, both of the transmit links of S1 (S1 to S0 and S1 to S2) go into signal fail state. When this occurs, S0 clears S1 as its east neighbor and S2 clears S1 as its west neighbor. S1 therefore disappears as a neighbor from all topology messages received at any station on the ring other than S1. S1 is also removed from the topology database at any station on the ring other than S1, as S1 is downstream of a failed span in both ring directions from any station on the ring other than S1.

At S1, topology messages from S0 and S2 can be received in wrapping rings. A topology message from S3 can additionally be received in steering rings. In all cases, any station on the ring other than S1 is not reachable for data from S1. Since topology messages can be received from other stations, from the perspective of S1 those other stations are considered control reachable and remain in the topology database, though with data reachability columns marked with N for no.

The usage of signal fail and idle indications follows the equivalent behavior as in J.2.1.

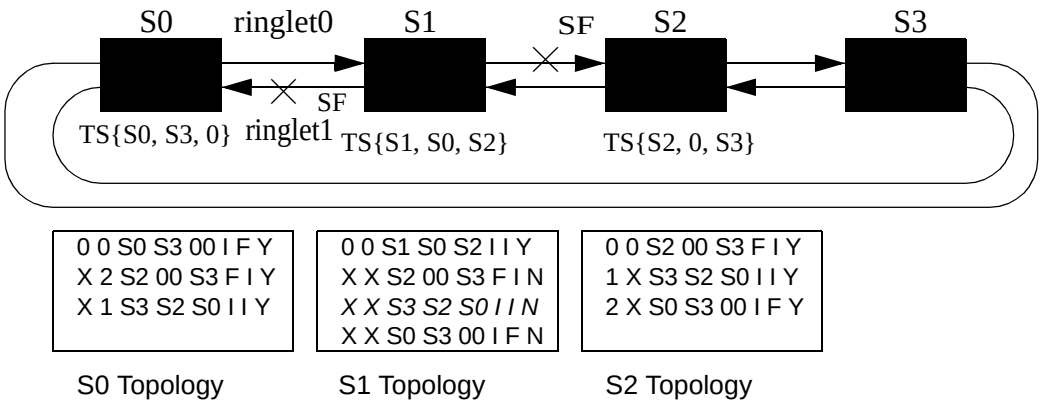


Figure J.2—Failure of two egress links.

J.2.3 Two failed receive links

Editors' Notes: *To be removed prior to final publication.*

This scenario will be added later.

J.2.4 One failed receive link and one failed transmit link, different spans

Editors' Notes: *To be removed prior to final publication.*

This scenario needs further discussion. Based on the rule currently defined in 11.7.1.3 that topology messages shall be stripped from the ring by a station in wrapping state, the discovered topology in this case results in no messages from S3 being forwarded to S1, even though S0 and S2 refer to S3 as being one of their neighbors. Currently there is no such stipulation in 11.7.1.3 for steering rings, and as a result S3 would be visible to S1 for steering rings. The information visible in steering rings but not visible in wrapping rings is shown in italics.

In this scenario, one of the transmit links of S1 (S1 to S2) and one of the receive links of S1 (S0 to S1) go into signal fail state. When this occurs, S1 clears S0 as its west neighbor and S2 clears S1 as its west neighbor. S1 is no longer data reachable from other stations on the ring, but remains control reachable (see editorial note above).

The usage of signal fail and idle indications follows the equivalent behavior as in J.2.1.

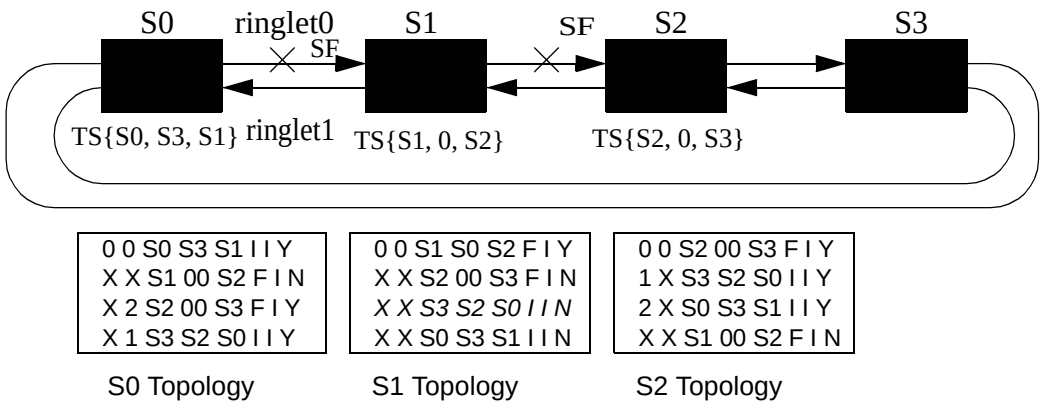


Figure J.3—One egress and ingress link failure.

J.2.5 Two failed receive links and two failed transmit links

Editors' Notes: *To be removed prior to final publication.*

This scenario will be added later.

J.3 Insertion and removal of stations

This subclause contains the following scenarios relating to the insertion and removal of stations to/from an RPR ring:

Editors' Notes: *To be removed prior to final publication.*

Scenario 5 will be added when it is determined how pass-through mode will be handled. It is expected that a trigger bit will be added to protection messages for this.

The PAH needs to add insertion of information for topology extended status messages into these scenarios.

1) Replacement of one or more stations without use of operator commands to change the protection status of links in the ring. The basic scenarios for insertion and/or removal of a station to/from an RPR ring are subsets of this scenario.
2) Replacement of one or more stations with use of operator commands to change the protection status of links in the ring.
3) Scenario 1), with existing ring being a wrapping ring and the newly inserted station being only steering-capable.
4) Removal of the only steering-only station from a wrapping-capable ring.
5) Addition or deletion of one or more stations using a mechanism such as electrical passthrough that will not trigger any change of protection status on any link in the ring during the replacement.

J.3.1 Replacement of stations without use of operator commands

The initial configuration of the ring is shown in Figure J.4. Station B is replaced with a new station G. G is fully connected into the ring prior to being powered up. In the case of a wrapping ring, station G is assumed to be wrapping capable. The final configuration of the ring is shown in Figure J.5.

This scenario is written for a single station replacement. The scenario can be directly extended for replacement of multiple adjacent stations.

The protection switching specific portions of this scenario are different for steering and wrapping rings. The topology specific portions of this scenario do not vary for steering and wrapping.

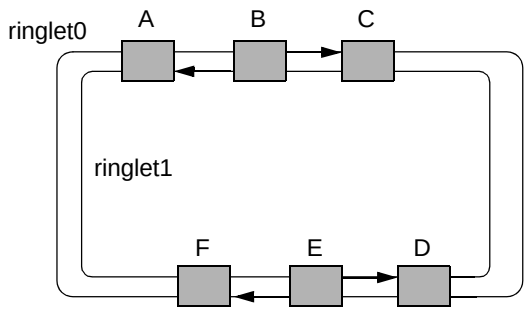


Figure J.4—An RPR ring with six stations.

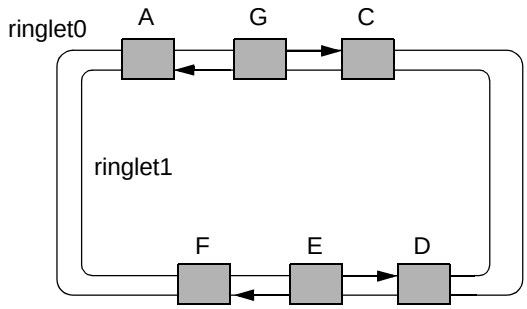


Figure J.5—An RPR ring with six stations after replacement of B by G.

J.3.1.1 Steering ring

The configuration of the ring during the replacement of station B with station G is shown in Figure J.6. Station B is disconnected from the ring via manual unplugging of fibers. Station G is then fully connected into the ring while powered down. Station G is then powered up. The final configuration of the ring is shown in Figure J.5.

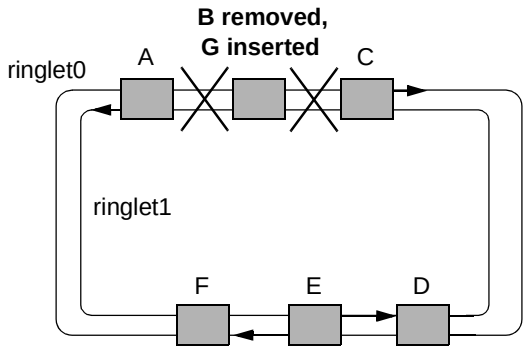


Figure J.6—Replacement of a station in a steering RPR ring

J.3.1.1.1 Removal of station B

The removal of station B may be done via one fiber removal at a time, via bidirectional fiber removal on the east and west interfaces of B, or via simultaneous bidirectional fiber removal, e.g. power-down of B.

Single fiber removal (fiber from A->B):

Actions at A and B:

1) Station B detects SF on ringlet0, transmits {SF, B, I, S} towards A on ringlet1, and transmits {SF, B, I, L}on ringlet0.
2) [Topology database at B is updated.]
3) Station B steers traffic appropriately.
4) Station A receives a protection request on ringlet1.
5) [Topology database at A is updated.]
6) Station A steers traffic appropriately.
7) Station A receives a protection request on ringlet0. As it is a duplicate request, A takes no additional action.

Actions at other stations on ring:

1) Each station receives a protection request from station B.
2) [Topology database at the station is updated.]
3) Each station steers traffic appropriately.

Simultaneous bidirectional fiber removal (e.g. power-down of station B):

Actions at A and C:

1) A detects SF on ringlet1, transmits {SF, A, I, S} towards B on ringlet0, and transmits {SF, A, I, L} on ringlet1.
2) C detects SF on ringlet0, transmits {SF, C, I, S} towards B on ringlet1, and transmits {SF, C, I, L} on ringlet0.
3) [Topology database at A is updated.]
4) Station A steers traffic appropriately.
5) [Topology database at C is updated.]
6) Station C steers traffic appropriately.
7) Station A receives a protection request on ringlet0.
8) Station C receives a protection request on ringlet1.
9) [Topology database at A is updated.]
10) Station A steers traffic appropriately.
11) [Topology database at C is updated.]
12) Station C steers traffic appropriately.

Actions at other stations on ring:

1) Each station receives protection requests from stations A and C.
2) [Topology database at the station is updated. For illustration, this is what occurs. The entries for link availability are updated to SF on links A->B, B->A, B->C, and C->B. The entries for B in the topology database are deleted, as are the east neighbor value for the A entry, and the west neighbor value for the C entries. All fields in the topology database showing upstream distances on ringlet1 beyond A are cleared. All fields in the topology database showing upstream distances on ringlet0 beyond C are cleared.]
3) Each station steers traffic appropriately.

The bidirectional fiber removal scenario on a single span (such as between A and B) follows the same steps as for the simultaneous bidirectional fiber removal, except that references to station C are replaced by station B. Modifications to the topology databases will also not result in the deletion of entries for station B.

J.3.1.1.2 Insertion of station G

Note: A station will usually return to service with one segment coming up after the other (with the time delta potentially close to 0). In this scenario, a station is powered up with the links connected and fault free.

Actions at G:

1) G is attached to the ring.
2) G is powered up.
3) Station G and the span between A and G return to service (SF on both links between A and G disappears).
4) Station G and the span between C and G return to service (SF on both links between C and G disappears).
5) Station G, not seeing any faults, transmits idle messages {Idle, G, I, S} and {Idle, G, I, S} on both ringlets.
6) The messages are forwarded through A and C to the rest of the ring.
7) All other stations on the ring respond to the protection message from station G with their own transmitted protection messages. G updates its topology database.

Actions at A:

1) The fault disappears on A, and A enters WTR state (briefly).
2) [Station A updates its topology database.]
3) Station A steers traffic appropriately.
4) Station A transmits {WTR, A, I, S} on ringlet0 and {WTR, A, I, L} on ringlet1.
5) Station A receives an idle protection message from station G.
6) [Topology database at A is updated with new entry for G. When full bidirectional connectivity is restored, A updates its topology database as it sees protection messages arrive from stations upstream on ringlet1.]
7) [Protection learns from the topology database that A's neighbor has changed to G.]
8) Based on G being different from the previously connected neighbor, station A drops the WTR. Station A transmits {Idle, A, I, S} on ringlet0 and {Idle, A, I, S} on ringlet1.
9) [Station A updates its topology database.]
10) Station A steers traffic appropriately.

Actions at C:

1) The fault disappears on C, and C enters WTR state (briefly).
2) [Station C updates its topology database.]
3) Station C steers traffic appropriately.
4) Station C transmits {WTR, C, I, S} on ringlet0 and {WTR, C, I, L} on ringlet1.
5) Station C receives an idle protection message from station G.
6) [Topology database at C is updated with new entry for G. When full bidirectional connectivity is restored, C updates its topology database as it sees stations appear upstream on ringlet0.]
7) [Protection learns from the topology database that C's neighbor has changed to G.]
8) Based on G being different from the previously connected neighbor, station C drops the WTR. Station C transmits {Idle, C, I, S} on ringlet0 and {Idle, C, I, S} on ringlet1.
9) [Station C updates its topology database.]
10) Station C steers traffic appropriately.

Actions at other stations:

1) At all other stations, protection messages from A, C, and G are received.
2) [Topology database at each station is updated appropriately based on modified information from A and C, new information from station G, and restoration of full bidirectional connectivity.]
3) Stations steer traffic appropriately.

J.3.1.2 Wrapping ring

The configuration of the ring during the replacement of station B with station G is shown in Figure J.7. Station B is disconnected from the ring via manual unplugging of fibers. Stations A and C wrap as a result. Station G is then fully connected into the ring while powered down. Station G is then powered up. The final configuration of the ring is shown in Figure J.5.

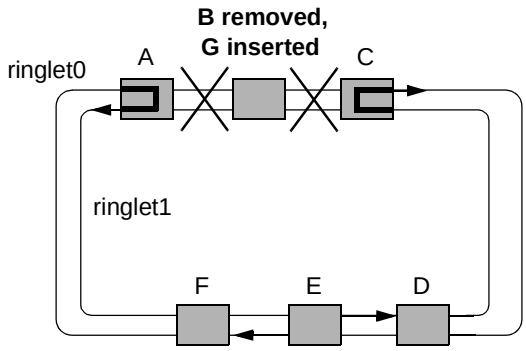


Figure J.7—Replacement of a station in a wrapping RPR ring

J.3.1.2.1 Removal of station B

The removal of station B may be done via one fiber removal at a time, via bidirectional fiber removal on the east and west interfaces of B, or via simultaneous bidirectional fiber removal, e.g. power-down of B.

Single fiber removal (fiber from A->B):

Actions at A and B:

1) Station B detects SF on ringlet0, transitions to Wrapped state (performs a wrap) on its west interface, transmits {SF, B, W, S} towards A on ringlet1, and transmits {SF, B, W, L}on ringlet0.
2) [Topology database at B is updated.]
3) Station A receives a protection request on ringlet1, and transitions to Wrapped state on its east interface.
4) [Topology database at A is updated.]
5) Station A receives a protection request on ringlet0. As it is a duplicate request, A takes no additional action.

Actions at other stations on ring:

1) Each station receives a protection request from station B.
2) [Topology database at the station is updated.]

Simultaneous bidirectional fiber removal (e.g. power-down of station B):

Actions at A and C:

1) A detects SF on ringlet1, transitions to Wrapped state (performs a wrap), transmits {SF, A, W, S} towards B on ringlet0, and transmits {SF, A, W, L} on ringlet1.
2) C detects SF on ringlet0, transitions to Wrapped state (performs a wrap), transmits {SF, C, W, S} towards B on ringlet1, and transmits {SF, C, W, L} on ringlet0.
3) [Topology database at A is updated.]
4) [Topology database at C is updated.]

5) Station A receives a protection request on ringlet0. Station A should already be wrapped on its east interface due to local detect of SF.
6) Station C receives a protection request on ringlet1. Station C should already be wrapped on its west interface due to local detect of SF.
7) [Topology database at A is updated.]
8) [Topology database at C is updated.]

Actions at other stations on ring:

1) Each station receives protection requests from stations A and C.
2) [Topology database at the station is updated. For illustration, this is what occurs. The entries for link availability are updated to SF on links A->B, B->A, B->C, and C->B. The entries for B in the topology database are deleted, as are the east neighbor value for the A entry, and the west neighbor value for the C entries. All fields in the topology database showing upstream distances on ringlet1 beyond A are cleared. All fields in the topology database showing upstream distances on ringlet0 beyond C are cleared.]

The bidirectional fiber removal scenario on a single span (such as between A and B) follows the same steps as for the simultaneous bidirectional fiber removal, except that references to station C are replaced by station B. Modifications to the topology databases will also not result in the deletion of entries for station B.

J.3.1.2.2 Insertion of station G

Note: A station will usually return to service with one segment coming up after the other (with the time delta potentially close to 0). In this scenario, a station is powered up with the links connected and fault free.

Actions at G:

1) G is attached to the ring.
2) G is powered up.
3) Depending on timing of transmission of protection messages, G may receive a message {SF, A, W, S} from A or a message {SF, C, W, S} from C while it is coming up. This will cause G to wrap. When A or C determines that the incoming link from G has now transitioned to idle status (and that G is a new neighbor), then the standard unwrapping sequence will take place, as described in J.4.
4) Station G and the span between A and G return to service (SF on both links between A and G disappears).
5) Station G and the span between C and G return to service (SF on both links between C and G disappears).
6) Station G, not seeing any faults, transmits idle messages {Idle, G, I, S} and {Idle, G, I, S} on both ringlets.
7) Prior to A, C, and G unwrapping, protection messages from G are stripped from the ring at A and C. Protection messages from A and C are received at G, but protection messages from other stations are stripped by A or C and do not reach G.
8) [A, C, and G update their topology databases appropriately.]
9) After A, C, and G unwrap, station G transmits a protection message on each ringlet. All other stations on the ring respond to the protection message from station G with their own transmitted protection messages.
10) [G updates its topology database.]

Actions at A:

1) The fault disappears on A, and A enters WTR state (briefly).
2) [Station A updates its topology database.]

3) Station A transmits {WTR, A, W, S} on ringlet0 and {WTR, A, W, L} on ringlet1.
4) Station A receives an idle protection message from station G.
5) [Topology database at A is updated with new entries for G. When full bidirectional connectivity is restored, A updates its topology database as it sees protection messages from stations upstream on ringlet1.]
6) [Protection learns from the topology database that A's neighbor has changed to G.]
7) Based on G being different from the previously connected neighbor, station A drops the WTR. Station A unwraps. Station A transmits {Idle, A, I, S} on ringlet0 and {Idle, A, I, S} on ringlet1.
8) [Station A updates its topology database.]

Actions at C:

1) The fault disappears on C, and C enters WTR state (briefly).
2) [Station C updates its topology database.]
3) Station C transmits {WTR, C, W, S} on ringlet0 and {WTR, C, W, L} on ringlet1.
4) Station C receives an idle protection message from station G.
5) [Topology database at C is updated with new entries for G. When full bidirectional connectivity is restored, C updates its topology database as it sees protection messages from stations upstream on ringlet0.]
6) [Topology informs protection that C's neighbor has changed to G.]
7) Based on G being different from the previously connected neighbor, station C drops the WTR. Station C unwraps. Station C transmits {Idle, C, I, S} on ringlet0 and {Idle, C, I, S} on ringlet1.
8) [Station C updates its topology database.]

Actions at other stations:

1) At all other stations, protection messages from A, C, and G are received (after A, C, and G unwrap).
2) [Topology database at each station is updated appropriately based on modified information from A and C, new information from station G, and restoration of full bidirectional connectivity.]

J.3.2 Replacement of stations using operator commands

Consider the scenario of J.3.1. For removal of a station, the only difference is that spans are commanded down via, for example, a forced switch (FS) command. This command is reported from the management station or command line interface to the OAM module via the MLME interface. The OAM module informs the protection module of the change in link status. From this point forward, the scenarios behave identically to the scenarios of J.3.1 for steering and wrapping rings, except that the link status being reported is FS rather than SF.

For insertion of a station via clearing of FS, once the protection module finds out about the management command, again the scenarios behave identically to the scenarios of J.3.1 for steering and wrapping rings, except that the link status being cleared to Idle is FS rather than SF.

J.3.3 Insertion of a steering-only station in a wrapping ring

Consider the scenario of J.3.1. For insertion of a steering station into a ring where all other stations are wrapping-capable and the ring is configured to prefer wrapping, the following additional steps are inserted, keeping all other steps the same:

1) G sends out protection messages on both ringlets indicating that it is not wrapping capable.
2) The neighbors of G (A and C) unwrap immediately upon finding out that G is not wrapping capable.
3) After A and C unwrap, protection messages from G are forwarded to the rest of the stations on the ring. Upon receipt of these messages, all other stations on the ring proceed to steer their traffic.

J.3.4 Removal of the only steering-only station from a wrapping-capable ring

Consider the scenario of J.3.1. For removal of a steering station from a ring where all other stations are wrapping-capable and the ring is configured to prefer wrapping, the following additional steps are inserted, keeping all other steps the same:

1) When other stations remove the entry for G from their topology databases, those stations check at that time if all remaining stations on the ring are wrapping capable and the ring is configured to prefer wrapping.
2) If so, A and C (neighbors of G) then wrap immediately.
3) If so, other stations on the ring re-steer their traffic appropriately (back to the default directions for the traffic).

J.4 Basic failure examples

J.4.1 Signal failure - single fiber cut scenario

This sample scenario is a ring of six stations: A, B, C, D, E, and F, with a unidirectional failure on a link from A to B, detected on B. There are no non-Idle protection states on the ring prior to the failure.

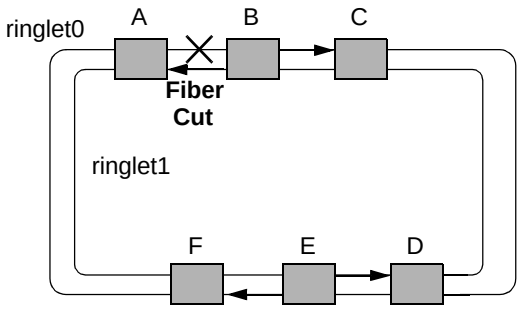


Figure J.8—An RPR ring with a fiber cut in ringlet1

J.4.1.1 Signal fail scenario

This scenario is described in J.3.1.2.1.

J.4.1.2 Signal fail clears

1) SF on Station B clears. Station B sets its WTR timer. If the ring is a wrapping ring, station B does not unwrap, and transmits {WTR, B, W, S} on ringlet1 and {WTR, B, W, L} on ringlet0. If the ring is a steering ring, station B transmits {WTR, B, I, S} on ringlet1 and {WTR, B, I, L} on ringlet0.

2) Station A receives a WTR request on the short path. If the ring is a wrapping ring, A does not unwrap. Short path protection messages from B continue to be stripped from the ring at A, and short path protection messages from A continue to be stripped from the ring at B.
3) WTR times out on Station B. If the ring is a wrapping ring, station B unwraps. Station B transmits {Idle, B, I, S} on ringlet1 and {Idle, B, I. S} on ringlet0.
4) If the ring is a wrapping ring, A unwraps upon receipt of {Idle, B, I, S}, then transmits {Idle, A, I, S} on ringlet0 and {Idle, A, I. S} on ringlet1.

All stations on the ring receiving protection messages update their topology databases accordingly.

J.4.2 Signal failure - bidirectional fiber cut scenario

This sample scenario is a ring of six stations: A, B, C, D, E, and F, with a bidirectional failure between A and B. There are no non-Idle protection states on the ring prior to the failure.

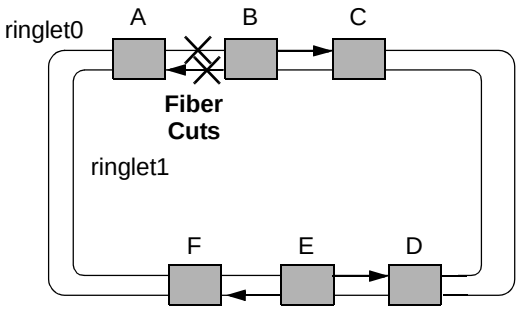


Figure J.9—An RPR ring with bidirectional fiber cut

J.4.2.1 Signal fail scenario

1) Station A detects SF on ringlet1. A transmits {SF, A, WSC_A, S} towards B on ringlet0, and transmits {SF, A, WSC_A, L} on ringlet1. WSC_A is W for a wrapping ring and I for a steering ring. If the ring is a wrapping ring, A wraps on its west span.
2) Station B detects SF on ringlet0. B transmits {SF, B, WSC_B, S} towards A on ringlet1, and transmits {SF, B, WSC_B, L} on ringlet0. WSC_B is W for a wrapping ring and I for a steering ring. If the ring is a wrapping ring, B wraps on its east span.

All stations on the ring receiving protection messages update their topology databases accordingly.

J.4.2.2 Signal fail clears

1) SF on Station A clears. Station A sets its WTR timer. If the ring is a wrapping ring, station A does not unwrap, and transmits {WTR, A, W, S} on ringlet0 and {WTR, A, W, L} on ringlet1. If the ring is a steering ring, station B transmits {WTR, A, I, S} on ringlet0 and {WTR, A, I, L} on ringlet1.
2) SF on B clears. If the ring is a wrapping ring, B does not unwrap. B transmits {WTR, B, WSC_B, S} on ringlet1, and transmits {WTR, B, WSC_B, L} on ringlet0, where WSC_B is W for a wrapping ring and I for a steering ring. Short path protection messages from B continue to be stripped from the ring at A, and short path protection messages from A continue to be stripped from the ring at B.

3) WTR times out on A. If the ring is a wrapping ring, A unwraps. A transmits {Idle, A, I, S} on ringlet0, and transmits {Idle, A, I, S} on ringlet1.
4) If the ring is a wrapping ring, B sees an idle request on the short path and unwraps. B transmits {Idle, B, I, S} on ringlet1, and transmits {Idle, B, I, S} on ringlet0.

All stations on the ring receiving protection messages update their topology databases accordingly.

J.5 Diagnostic scenarios

This subclause contains the following diagnostic scenarios:

1) Station connected to itself in optical loopback

J.5.1 Optical loopback

A station A is connected to itself in optical loopback when its east transmit is connected to its west receive, and its west transmit is connected to its east receive. The following steps occur when a station is powered up when connected in optical loopback.

1) A is powered up.
2) Station A, not seeing any faults, transmits protection messages {Idle, A, I, S} and {Idle, A, I, S} on both ringlets.
3) [Station A updates its topology database upon receipt of these messages. Station A detects a station with a MAC address duplicating its own on the ring. It generates a duplicate MAC address indication, but the ability of the station to send and receive data is not restricted.]

J.6 Operator commanded scenarios

This subclause contains the following operator commanded scenarios:

1) Forced switch commanded on an RPR span (east or west from the perspective of a station)
2) Forced switch cleared on an RPR span

J.6.1 Forced switch commanded on a span

Figure J.10 shows a scenario where a forced switch is commanded on the east span of station A. The following steps occur when this command is executed.

1) A checks in its topology database for a SF condition on its east transmit link (to B). If there is such a condition, A rejects the FS. This is to prevent the possibility of a single-ended wrap if the FS is later cleared.
2) If the FS is accepted, then if the ring is a steering ring A transmits {FS, A, I, S} on ringlet0 and {FS, A, I, L} on ringlet1. If the ring is a wrapping ring, A wraps on its west side and transmits {FS, A, W, S} on ringlet0 and {FS, A, W, L} on ringlet1.
3) If there was already a SF or SD condition on A's receive link on ringlet1, then that condition is stored as pending.
4) If the ring is a wrapping ring, B wraps on its east side based on receipt of {FS, A, W, S}, then transmits {Idle, B, W, S} on ringlet1 and {Idle, B, W, S} on ringlet0.
5) In the event that the above steps have already been completed and an SF condition now is detected on the link A->B on ringlet0, A rejects its existing FS upon being informed of the SF

condition by received protection messages from B. The appropriate steps are then taken to clear the forced switch.

All stations on the ring receiving protection messages update their topology databases accordingly.

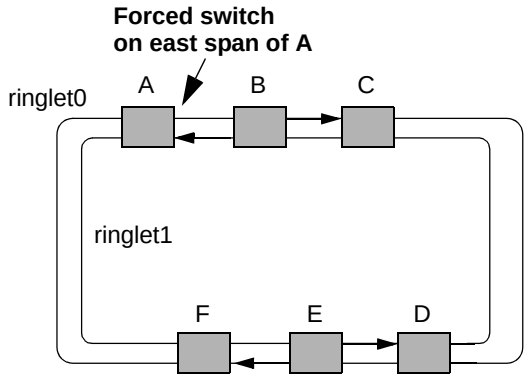


Figure J.10—An RPR ring with six stations.

J.6.2 Forced switch cleared on a span

The following steps occur when the forced switch command is cleared.

1) A checks for a pending SF or SD condition on its receive link on ringlet1. If such a condition exists, A reports that condition as REQ_A in the steps below. If not, A reports Idle as REQ_A in the steps below. If the ring is a wrapping ring, A also performs the necessary checks described in 11.8 to determine if it should remain wrapped for the SD condition. If A needs to remain wrapped, WSC_A is set to W in the steps below. If A does not need to remain wrapped, WSC_A is set to I in the steps below and A unwraps. WSC_A is also set to I in the steps below if the ring is a steering ring.
2) A transmits{REQ_A, A, WSC_A, S} on ringlet0 and {REQ_A, A, WSC_A, L} on ringlet1.
3) If the ring is a wrapping ring and WSC_A equals W, B remains wrapped. If WSC_A equals I, B unwraps.

All stations on the ring receiving protection messages update their topology databases accordingly.

Annex K

(informative)

Connectivity monitoring using echo request/response

Editors' Notes: To be removed prior to final publication.

References:
None.

Definitions:
None.

Abbreviations:
None.

Revision History:

Draft 1.1, October 2002	Draft 1.1 for WG review, added according to comments on D1.0.
Draft 2.0, December 2002	Draft 2.0 for WG ballot, modified according to comments on D1.1.

K.1 Background

RPR stations are connected by point to point links that form a ring. The whole 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.2 Scope

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 RPR compliant 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.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 2 possible paths (east and west) that can carry up to 3 service classes (A, B and C). 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.3.1 Monitored paths

As stated before, there are up to 2 possible paths between stations. Stations may monitor:

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 the station may monitor

a) A single service class (A, B or C)
b) Two out of the three service classes
c) All the service classes

K.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. The echo request processing time for all stations shall be less than 10 milliseconds.

K.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.