10. Topology discovery and protection

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	*Revised according to comments on D0.3.*
Draft 1.1, October 2002	*Revised according to comments on D1.0.*
Draft 2.0, December 2002	*Revised according to comments on D1.1.*
Draft 2.1, February 2003	*Revised according to ballot comments on D2.0.*
Draft 2.2, April 2003	*Revised according to comments on D2.1.*
	Topology discovery and protection clauses combined.
Draft 2.3, June 2003	*Revised according to ballot comments on D2.2.*
Draft 2.4, August 2003	*Revised according to ballot comments on D2.3.*
Draft 2.5, September 2003	*Revised according to ballot comments on D2.4.*
Draft 2.6, October 2003	*Revised according to ballot comments on D2.5.*
Draft 2.7, October 2003	*Revised according to ballot comments on D2.6.*

Topics for discussion:
Jason Fan: 408 342-6417

All C code should have uniform 4-space indents.

All loop-based validity checks should be handled by an “if (!valid) continue” like statement.

An additional state machine (or additional state-machine states) is needed to parse the ATD frame on behalf of the SecondaryUpdate state machine. This state machine would:

Define the behavior of ATTs that extend beyond the end of frame.
Define the behavior of an ATT with an illegal size.
Define the behavior of duplicate ATT entries (last one has precedence?).

Code should conform to the open curly bracket on the left (as opposed to the right).

Why does the TP frame state machine not save the most-recent copy?

Please explain the other “Protection special cases” items, and verify their continued relevance.

Shouldn’t there be a minimum TP frame storage requirement? If thought this was agreed upon.

The following seems excessively large:

The topology discovery time is normally bounded by … discovery time of approximately RRTT*(MAX_STATIONS/2), plus processing time. For 64 stations on a 1000 km ring, this is approximately 160 ms.

Is Table 10.1 intended to be normative, or is this formally specified in state machines?

Where should the following normative statement be made?

All RPR stations shall support steering; RPR stations may support wrapping.

The following seems self-contradictory, since the default value is out-of-range:

A configurable hold-off time *holdOffTimerValue* shall be supported per span of the station for protection triggers with a range of zero to at least 200 ms with 10 ms resolution, and a default value of zero.

What state machine defines the behavior of the *holdOffTimerValue* value?

An RPR keepalive failure on a receive span shall be triggered by the failure to receive any SCFF frames (as defined in 9.3.1) for a configurable time range from 2 ms to 50 ms, with a resolution of 1 ms and a default value of 3 ms.

The normative *keepaliveDelay* values should be attached to its definition, and the normative values for that value should be associated with its definition.

The following recommendation seems to vague to be implemented:

The transmission of RPR keepalives should occur only if all implemented checks of normal MAC operation are fulfilled.

Why is the receipt of a TC frame inhibited by miscabling, rather than a general SF condition?

Is the database *macAddress* update from TP frames or both TP/ATD frames (as tables indicate)?

Why is the following stated? Doesn’t the FS-means-all-else-is-ignored policy cover this one?

If a FS request is present on a span of a station receiving from ringlet ri (*myTopoInfo.spanProtState*[*ri*] equals FS), and an SF condition is detected by the neighbor station on ringlet *Other*(*ri*) on the other end of the same span, the neighbor station accepts and processes the SF condition ignoring the FS. (Without this

rule, the neighbor station would not go into WTR upon clearing of SF, even if the FS had been cleared in the meantime on the original station.)

Does this funky thing still exist?

If FS requests are present on both spans of a station (myTopoInfo.spanProtState[ri] and myTopoInfo.spanProtState[Other(ri)] both equal FS), then the remainder of the stations on the ring see an open ring with idle edges on both ends. Protection conditions lower in the hierarchy than SF anywhere on the open ring are preempted when there are idle edges on both ends.

Many normative behaviors are specified by “shall” code, rather than state machine behaviors.

The secondary MAC address of zero should not be used; this is a legal Xerox assignment

The *GetSpanStatus*() values are inconsistent with new Clause 7 values.

The station bandwidth ATT should be updated to include four linear segments (40 additional bytes).

In their definitions, spell out the meanings of *bf*, *co*, and *mu* abbreviations, so they can be remembered.

The following statement appears to be wrong: isn’t it only A1 that can detect this duplicate?

b) A0=B=C=D=A1=E=F=G=A2 (detectable by A0 and A2)

With text like “10 ms”, the space between “10” and “ms” should be a nonbreaking space.

Consistency: use ms (as opposed to milliseconds) throughout.

What is the functional difference between the observation (by other stations) of the co and mu bits?
If there is no functional difference, the value of having two distinct bits should be noted.

State machine names should be run-together words, so their meanings can be easily understood within sentences. Also, this consistency is useful for C code, since the state machine name can be used as the C routine name, rather than mandating use of a similar (but different) names depending on the context (C code versus English text) of the identifier.

Use RETN, rather than RETURN, to increase the width of C code state machine tables.

We once stated that stations were required to provide space for queueing one TP frame for each of the possible MAX_STATION sources, and the most recent one should overwrite previous ones in the queue. Is this still mandated and (if so) where is this requirement stated.

The value of *timeSinceFailDegrade* definition is vague and not formally specified by a state machine.
Once started, does this timeout continue even when restored?
If not, does a minute restore clear this value?

10.1 Overview

10.1.1 Scope

This clause specifies the topology discovery and protection functions of the RPR MAC, including:

a) Topology discovery for plug-and-play topology modification.
b) Protection switching for sub 50 ms resiliency, with no reordering/duplication of strict data frames.
c) Attribute discovery (ATD) for reporting of additional, less time-critical information.
d) Topology checksum (TC) exchange to minimize loss of strict data frames due to topology changes.
e) Loop round trip time (LRTT) measurement to optimize conservative mode fairness performance.

The topology, protection, ATD, TC, and LRTT measurement functions reside in the MAC control sublayer, as shown in the shaded region of Figure 10.1.

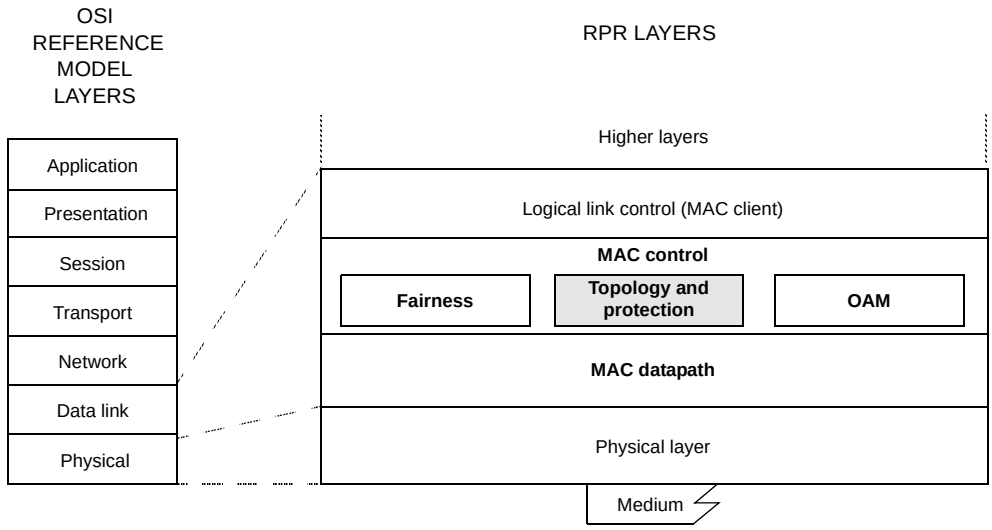


Figure 10.1—Topology and protection entities relationship to the ISO/IEC OSI reference model

The protection function provides reliable mechanisms for under 50 ms protection switching (not counting configurable holdoff time or failure detection time) for all protected traffic on an RPR ring. It enables a mandatory protection mechanism called steering, and an optional protection mechanism called wrapping. This protocol ensures that each station receives link status change information, e.g., link failure or restoration information, required to make protection switching decisions reliably, and in the time frame required. This protocol also ensures that stations steer or wrap in accordance with the RPR protection hierarchy.

The topology discovery function provides a reliable and accurate means for all stations on a ring to discover the topology of the stations on the ring, and any changes to that topology. It also provides a mechanism for rapid detection of topology changes. The topology discovery and protection functions are closely related in that they share a common control messaging mechanism.

The information gathered by the topology discovery, protection, ATD, TC, and LRTT measurement functions are logically stored together in a shared topology database. Information in the topology database is used by the RPR ringlet selection protocol (see Clause 6), and the RPR fairness protocol (see Clause 9).

The services and features provided are as follows:

a) Responsive in multiple ways:
   1) Under 50 ms protection switching for unicast and multicast traffic based on media failures, media degradation, or operator invoked command.
   2) Quick dissemination on the ring of changes in protection state indicating media failures, media degradation, or operator invoked command.

b) Robust in multiple ways:
   1) Support of a comprehensive protection hierarchy.
   2) Support of dynamic addition and removal of stations to/from the ring.
   3) Tolerance of frame loss.
   4) Operation without any master station on the ring.
   5) Validation of topology, including detection of mis-cabling between stations.
   6) Operation independent of and in the absence of any management systems.
   7) Assurance that all stations on the ring converge to a uniform and current image of the topology.
   8) Support of context containment, which enables the guarantee of no duplication and re-ordering of strict mode traffic.

c) Flexible in multiple ways:
   1) Support of revertive and non-revertive protection switching operational modes.
   2) Operation with all supported topologies: closed ring and open ring.
   3) Scalability from one to 255 stations.
   4) Means of sharing additional information between stations.
   5) Means of determining loop round trip time information that can be used to optimize the performance of conservative mode fairness.

d) Informative: Determination and validation of connectivity and ordering of stations on the ring.

e) Efficient: Support of the above services and features with insignificant additional ring traffic, software execution time, or hardware facilities.

10.1.2 Protocol overview

10.1.2.1 Discovery protocol

The topology discovery protocol provides each station on the ring with knowledge of the number, basic capabilities, and arrangement of other stations on the ring. This collection of information is referred to as the topology image; each station's copy of the topology image is called the topology database. The topology database includes an array of station information, where the hop-count distance from other stations determines their location of their information within this array.

The critical topology database information is derived from information supplied by topology and protection (TP) control frames received on each ringlet. Link protection state, edge status, and protection configuration information are included in the received TP frames. The TP frame is sent when triggered by state-changes, and periodically thereafter.

Station preferences and less time-critical information is communicated via the attribute discovery (ATD) control frame and placed into the topology database. The ATD frames include standard station preferences and optional organization-specific information, although the use of organization-specific information is beyond the scope of this standard.

Each station maintains its topology database based on rules for transmission and reception of TP and ATD frames. If the topology image is stable, periodic TP and ATD frame transmissions leave the topology database unchanged.

10.1.2.2 Protection protocol

The protection protocol uses link protection state information from the topology database to determine if the edge state has changed for any span on the ring. This edge state information to the MAC datapath sublayer, which steers traffic via ringlet selection and which controls wrapping. Network operation specified parameters (such as hold-off time and wait-to-restore time) effect the protection protocol behavior. Operator-initiated protection requests and protection requests triggered by link monitoring interact robustly, within the framework of the RPR protection hierarchy.

10.1.2.3 Topology discovery and protection time

Basic RPR topology discovery (without passthrough) has a worst-case notification of topology change of approximately RRTT (plus small amounts of processing time to trigger and process TP frames), assuming no TP fame loss. When TP frames are lost, the topology discovery time increases by *txFastDelay*, the fast period of the TP frame (see 10.2.3). For a 1000 km ring, the nominal worst-case notification time is approximately 5 ms; if both copies of a TP frame (one transmitted on each ringlet) are lost, the notification time increases to 15 ms.

The ring round trip time (RRTT) makes up most of the topology notification time, and is the key to sub-50 ms protection. For relaxed mode traffic, protection occurs immediately upon notification of a topology change, and before the topology is determined to be stable and valid. For strict mode traffic, the topology is first determined to be stable and valid before this traffic is restored.

The topology discovery time is normally bounded by the topology discovery stabilization time; with passthrough, the worst-case stabilization time is thus approximately RRTT*(MAX_STATIONS/2), plus processing time. For 64 stations on a 1000 km ring, this is approximately 160 ms.

10.1.2.4 Topology validation

The purpose of topology validation is to ensure that a variety of defects in the topology can be detected and reported to the operator. These defects include miscabling detection, topology instability, topology inconsistency, and an excessive number of stations on the ring.

In addition, the topology checks performed as part of topology validation are essential to the determination of when context containment for strict mode traffic can be cleared. This is described in 10.1.3.5.

10.1.2.5 Edges

The topology discovery protocol is robust for both fully connected bidirectional rings (closed rings) and for bidirectional rings in which edges are detected (open rings). Edges are spans through which data does not transit.

An edge is a span on which a protection condition exists that is sufficiently high in the protection hierarchy to cause a wrap in a wrapping ring, or stations to steer traffic away from that span in a steering ring. Only topology and protection (TP) and single-choke fairness (SCFF) control frames transmitted across an edge span. The edge-crossing TP frames communicate useful information; the SCFF frames provide keepalive function for detecting link failures. Data frames and other control frames are discarded by stations transmitting over an edge span and discarded by station receiving from an edge span, as described in 6.6.3.6.

10.1.3 Initialization

10.1.3.1 Initial TP frame transmissions

An initializing station broadcasts TP frames on both ringlets (see 10.6.7), communicating its information to others. The station monitors others' TP frames and updates its topology database with information supplied by unknown stations, or from known stations whose TP frame contents have changed. Topology validation checks confirm the consistency of stable topologies. An initializing station also broadcasts ATD frames on all ringlets.

When one or more stations come out of passthrough mode, those stations behave as newly initialized stations. The behavior of the other stations on the ring is similar to that for stations detecting a newly initialized station on the ring.

10.1.3.2 Addition of a station

When a station is added to an existing ring, its neighbors detect the station as a new neighbor by receiving its TP frame (indicating the frame has traveled exactly one hop) and an *sa* (source MAC address) different from that of the previously observed neighbor.

After detection a new station on the ring, stations quickly transmit TP frames on both ringlets. Other stations detect the new station by receiving its TP frame, and each responds by quickly transmitting TP frame on both ringlets. Valid TP frames are used to update the topology database, until each entry in the database has been marked valid.

10.1.3.3 Span and station failures

When a span fails, the stations detecting the failure store the knowledge of their neighbor station address on the failed span and clear their current knowledge of their neighbor station address on that span. The stations send TP frames that report exactly which links failed to the other stations on the ring. In the case of span degradation or protection requests, the protection hierarchy described in 10.1.5.6 is applied.

When a station is removed from a ring, the neighbor stations activate protection actions (see 10.1.3.3). Each neighbor station detects a signal fail (SF) condition and responds by broadcasting a TP frame on its remaining ringlet. Upon receipt of the TP frames and update of the topology database, stations beyond the reported edges are normally marked invalid.

10.1.3.4 Entering passthrough

When one or more stations go into passthrough mode, those stations effectively disappear from the ring, from both a control and data transfer perspective. Other stations continue periodic transmissions of their TP frames; passthrough is detected when a station receives a TP frame with a different *sa* (source MAC address) than expected from a station at that hop-count based location.

The speed of passthrough detection depends on the setting of the slow TP frame transmission period, which defaults to 100 ms. Passthrough detection is not covered by the 50 ms protection switching and service restoration that is applicable to media failures, e.g., ring and station failures resulting from a broken ring, or an operator commanded protection event.

10.1.3.5 Context containment

The purpose of context containment is to prevent duplication and re-ordering of strict mode data frames in the event of topology changes, as defined in 6.6.1.1. To prevent duplication and re-ordering, context containment (once entered) is not cleared until two conditions are satisfied.

a) The new topology discovered by that station is determined to be stable and valid.

b) The checksum computed based on the topology database is determined to match the checksums computed at both neighbor stations, based on values communicated by the TC protocols. (The checksum for any station is ignored until the topology at that station is stable and valid.)

The duration of context containment is primarily made up of the topology stabilization time if the ring does not support passthrough. If the ring does support passthrough, the duration is also significantly impacted by the overall topology discovery time.

10.1.3.6 Topology checksums in TC frames

Whenever the topology is not stable or valid, the checksum in the TC control frame is marked as invalid. When a station is not in context containment, the detection that its checksum does not match the checksum received from either neighbor station does not trigger context containment.

A checksum computation involves one step for each of the configured stations. Two 32-bit numbers are added in each of these checksum computation steps, as specified in Equation 10.18. The first consists of the most significant 4 bytes of the source MAC address for that entry. The second (in most-significant to least-significant bit order) consists of the 2 least significant bytes of the source MAC address, 10 reserved bits, and the 6-bit sequence number, as illustrated in Figure 10.2.

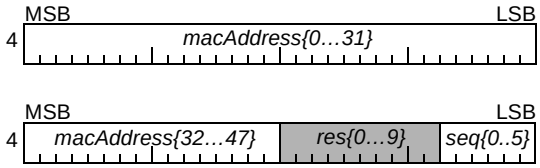


Figure 10.2—Alignment of the 32-bit fields used to compute topology checksum

10.1.4 Special services

10.1.4.1 Secondary MAC addresses

The purpose of secondary MAC address support is to enable the use of higher layer redundancy protocols on an RPR ring. Topology enables stations to communicate their secondary MAC addresses to the rest of the ring. Secondary MAC addresses are communicated using the ATD frame. As secondary MAC address information is stored in the topology database, topology also provides validation to ensure that no secondary MAC address duplicates an existing station MAC address, and to ensure that the total number of valid secondary MAC addresses on the ring does not exceed 32. Failure of these validation checks results in the triggering of defects. Configuration of secondary MAC addresses at each station is also validated before updates are made to the topology database.

10.1.4.2 LRTT measurement protocol

A station deploying the conservative method of fairness rate computation sends unicast loop round trip time (LRTT) request frames individually on each ringlet to all other stations on the ring. LRTT response frames are sent back to each station sending LRTT request frames. All of these frames are sent as subclassA0, enabling the most accurate possible measurement of the actual loop round trip time. The measured round trip time information is stored in the topology database, and is used to optimize the performance of conservative mode fairness, as described in 9.5. This measurement is done when a new topology is validated, and can be done at a slow periodic interval.

10.1.5 Fault response mechanisms

10.1.5.1 Steering protection

Station(s) detecting a span or station failure report the protection-state change to the other stations using the TP frame. Steering-protected stations receive this frame and update their topology database (see 10.5) accordingly. The stations then direct traffic onto ringlet0 or ringlet1 on a per destination-station basis, thereby avoiding the failed link.

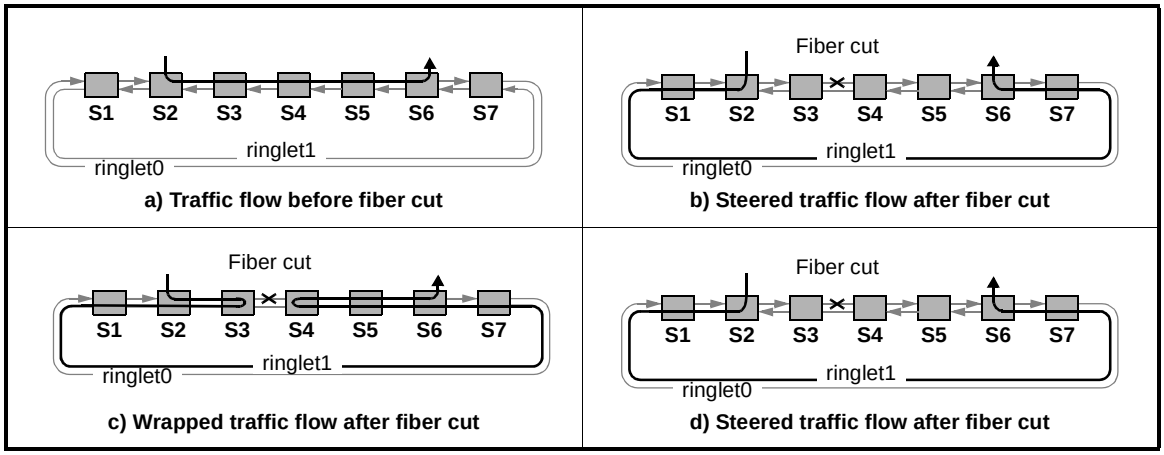


Figure 10.3—Steering protected traffic flows

As an example, station S2 normally sends to station S6 via the ringlet0 path S2⇒S3⇒S4⇒S5⇒S6, as illustrated in Figure-Cell 10.3-a. After the fiber cut, station S2 updates its topology database and steers protected S6-destined traffic via the ringlet1 path S2⇒S1⇒S7⇒S6, as illustrated in Figure-Cell 10.3-a

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, are dropped at the edge. Multicast frames are 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.

10.1.5.2 Wrap protection

On a wrapping ring, a span or station failure causes frames to be wrapped (looped back) at the edge station. As an example, station S2 normally sends to station S6 via the ringlet0 path S2⇒S3⇒S4⇒S5⇒S6, as illustrated in Figure-Cell 10.3-a. After the fiber cut, station S2 sends the traffic via the ringlet0 path S2⇒S3⇒S2⇒S1⇒S7⇒S6⇒S5⇒S4⇒S5⇒S6. Station S6 does not strip frames from the opposite ringlet of that indicated in the wrapped frame (ringlet1 in this case) to avoid frame misordering during the protection event.

Subsequently, a new ring topology is discovered and a new optimal path S2⇒S1⇒S7⇒S6 can be used. The client, using the options defined in Clause 6, can choose to re-steer the traffic, as illustrated in Figure-Cell 10.3-d, or can choose to keep the traffic wrapped, as illustrated in Figure-Cell 10.3-c. Note that the subsequent optimal path selection is not part of the standard protection protocol.

NOTE—Wrap protection uses signaling similar to SONET/SDH bidirectional line switched ring (BLSR) protection and meets the timing requirements for BLSR as per ANSI T1.105.01-2000, not counting configured holdoff time.

10.1.5.3 Protection hierarchy

The signal failure (SF) or signal degrade (SD) conditions can be detected on a span. The MAC can be directed to generate a forces switch (FS) or manual switch (MS) condition. The effect of the protection condition depends on the severity order of these protection conditions, listed below, where the top-through-bottom conditions are listed in order of most-through-least severity.

a) FS—A management directive that forces a link to be deactivated.
b) SF—A signal loss or major signal degradation that deactivates the link.
c) SD—A minor signal degradation that (if not overridden) deactivates the link.
d) MS—A management directive that (if not overridden) deactivates the link.
e) WTR—An active wait-to-restore timer, after SF or SD.
(the WTR timer avoids chattering conditions by extending signal-quality-related recovery times).

These protection conditions affect the RPR topology, as illustrated in Figure 10.4. The left column specifies the *P1*-span and *P2*-span degradations that lead to the protected topology figure in the right column. More severe *P2* locations are not illustrated; these can be derived by interchanging the *P1* and *P2* span labels.

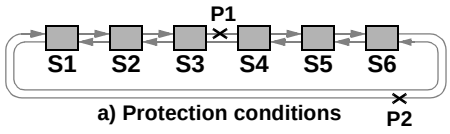
The operator originated FS is often used to add or remove a station from a ring in a controlled fashion. The operator originated MS can deactivate a marginally operating link, while allowing its automatic reactivation in the presence of a serious failure on another span.

A link status value of SF corresponds to a miscabling error, severe signal degradation or signal loss (PHY_LINK_STATUS.indication, see 7.2.3), loss of an expected keepalive indication (see 10.1.5.6), or partial/total loss of a station (see 10.7). A SD is caused by a degradation of signal quality (PHY_LINK_STATUS.indication, see 7.2.3). These link-status indications represent a merger of link-specific and internal-MAC status information, as specified in Table 10.1.

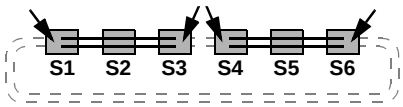
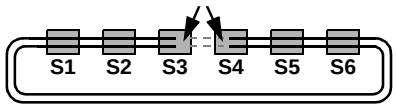
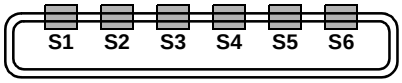
Table 10.1— Mapping of LINK_STATUS and MAC status to protection status

LINK_STATUS from PHY	Internal MAC status	*spanProtectStatus*
NO_DEFECT	IDLE	IDLE
	SF	SF
SIGNAL_DEGRADE	IDLE	SD
	SF	SF
SIGNAL_FAIL	IDLE	SF
	SF	
DEGRADE_FAIL	IDLE	
	SF	

The SF and SD (signal fail or signal degrade) are reported throughout the ring, even when a higher priority request or protection status is present on other links. This link-status knowledge is intended to be beneficial from a diagnostic perspective. All TP frames (whether sourced by neighbor stations, non-neighbor stations, edge-span stations, or non-edge span stations) are used to determine if protection conditions such as MS or WTR are preempted by protection conditions reported from elsewhere on the ring.



a) Protection conditions

P1 link	P2 link	Row	Resulting protection topology
FS	FS, SF	1	 b) Multi-break open-ring topology
SF	SF	2	
FS	SD, MS, WTR, IDLE	3	 c) Single-break open-ring topology
SF	SD, MS, WTR, IDLE	4	
SD	MS, WTR, IDLE	5	
MS	WTR, IDLE	6	
WTR	IDLE	7	
SD	SD	8	 d) Retained closed-ring topology
MS	MS	9	
WTR[e]	WTR[e]	10	
IDLE	IDLE	11	

[c]A MS directive is rejected when a MS is observed on a distinct-span location.
[d]A WTR is cancelled when
a FS, SF, SD, or MS is observed.
[e]A WTR is cancelled when a WTR is observed on a distinct-span location.

Figure 10.4—Protection topologies

Row 10.4-1: An FS directive forces an edge, even when the link is severed elsewhere.
Row 10.4-2: A SF degradation forces an edge, even when the link is severed elsewhere.

Row 10.4-3: An FS overrides SD, MS, and WTR.
Furthermore, other MS directives are rejected and WTR conditions are cleared.
Row 10.4-4: An SF overrides SD, MS, and WTR.
Furthermore, other MS directives are rejected and WTR conditions are cleared.
Row 10.4-5: An SD overrides MS and WTR.
Furthermore, other MS directives are rejected and a WTR condition is cleared.
Row 10.4-6: An MS directive overrides a WTR condition.
Furthermore, a WTR condition is cleared in the presence of an FS condition.
Row 10.4-7: A WTR condition has precedence over an IDLE condition.

Row 10.4-8: An SD on two distinct spans cause both to be ignored, since the precedence is ambiguous. (SD on two links of a span still forces adjacent edges.)
Row 10.4-9: An MS on two distinct spans cause both to be cleared, since the precedence is ambiguous. (MS on two links of a span still forces adjacent edges).

Row 10.4-10: A WTR on two distinct spans cause both to be cleared, since the precedence is ambiguous. (WTR on two links of a span still forces adjacent edges).
Row 10.4-11: No directives or degradation conditions allows the ring to operate in a closed-ring topology.

10.1.5.4 Passthrough mode

The capability of a station to enter passthrough mode is optional. Passthrough mode is a way to enable stations to enter or exit the ring topology without disconnection of fibers, particularly upon detection of internal failure conditions. Passthrough can potentially be achieved without the triggering of a signal fail protection state on any span.

The ongoing periodic transmission of TP frames is relied upon to indicate the topology change to all stations on the ring. When any station receives such an indication, transmission of TP frames is triggered to facilitate fast rediscovery of the topology, and faster restoration of strict mode traffic.

Passthrough allows a station to leave the ring with minimal impact to the traffic that is transiting the station. As a result, there is no requirement in RPR for protected traffic to switch directions on the ring because of a passthrough event.

When a station goes into passthrough, TP frames sent by that station can go around the ring many times due to the absence of its nominal source-stripping station. This recirculation interval impacts both topology discovery time in passthrough scenarios (discussed in 10.1.2.3) and the duration of context containment for strict mode traffic (discussed in 10.1.3.5).

10.1.5.5 Protection sequences

The procedure that is followed for the triggering of context containment and the clearing of context containment in a scenario consisting of span failure in a closed ring followed by span recovery is illustrated in Figure 10.5 and described below.

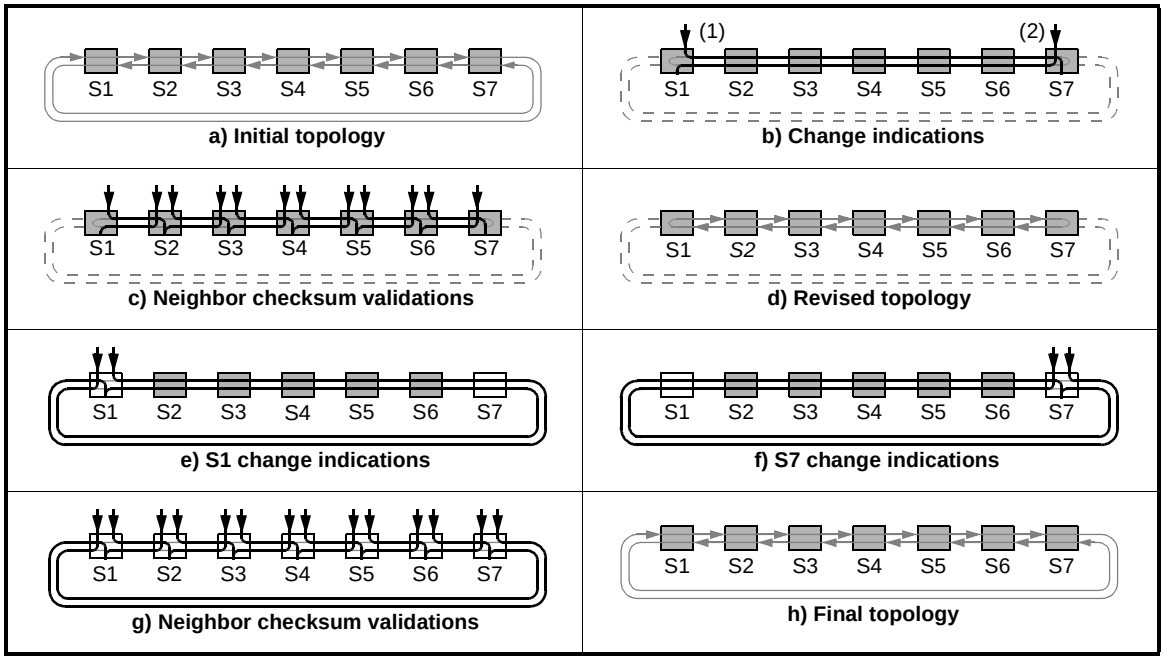


Figure 10.5—Span failure and recovery

The initial closed-ring topology of Figure-Cell 10.5-a is disrupted by a fault on the stations S7-to-S1 span. Stations S1 and S7 broadcast TP frames indicating a new edge, as illustrated in Figure-Cell 10.5-b. These TP frames trigger context containment at all steering-ring stations (context containment is not triggered at any wrapping-ring station).

After the new topology is determined to be stable and valid, each station compares its locally computed topology checksum with those received from its neighbor stations in the TC frame, as illustrated in Figure-Cell 10.5-c. Edge stations S1 and S7 only compare their topology checksum with the neighbor station on the non-edge span. At every station where the relevant neighbor checksums match the locally computed checksum, context containment is cleared. A revised open-ring topology is formed, as illustrated in Figure-Cell 10.5-d.

The span between stations S1 and S7 is restored. Station S1 broadcasts TP frames indicating the removal of an edge, as illustrated in Figure-Cell 10.5-e. Station S7 broadcasts TP frames indicating the removal of an edge, as illustrated in Figure-Cell 10.5-f. These TP-frame transmissions trigger context containment at all stations of a steering ring or wrapping ring.

After the new topology is found to be valid and stable, stations compare their locally computed topology checksum with those supplied by their neighbor's TC frame, as illustrated in Figure-Cell 10.5-g. As neighbor checksums are found to match the locally computed checksum, context containment is cleared. The original open-ring topology is formed, as illustrated in Figure-Cell 10.5-h.

10.1.5.6 Keepalive check behaviors

Each link has keepalive checks that report an SF link condition, based on the lack of receipt of expected fairness frames on each of the two receive links. These keepalive checks have the following properties:

a) A configurable *holdOffTimerValue* can suppress spurious responses to expected link-status glitches, by extending the time between detection and reporting of a physical SF condition. The glitches could be due to protection switching of RPR traffic by underlying SONET infrastructure.
b) The transmission of RPR keepalives should occur only if all implemented checks of normal MAC operation are fulfilled.
c) An RPR keepalive failure is triggered by the absence of received fairness frames for a *keepaliveDelay*-specified interval. (The default *keepaliveDelay* value triggers an RPR keepalive failure after a loss of four consecutive SCFF frames). The larger configurable *keepaliveDelay* values facilitate the delay of layer-2 failure detection, so that layer-1 equipment redundancy can take effect before protection is initiated.
d) Upon receipt of a single keepalive, a link in SF transitions to WTR state. The link is then activated upon the receipt of a TP frame from a new neighbor, or the expiration of the WTR timeout interval.

10.1.6 Defect conditions

A variety of protection/discovery defects can be detected and reported, as follows:

a) Another station's *protConfig* value (protection configuration) value differs from that of this station. (All stations are expected to be configured to use the same protection mechanism.)
   A ring that has steering-configured and wrapping-configured stations will unable to wrap on spans connected to steering-configured stations. Traffic sourced by wrapping stations on the ring, that is intended to traverse the edge span to reach its destination(s), is dropped at this edge rather than being wrapped. Thus, traffic from wrapping-configured stations is not fully protected.
b) A TP frame is received and its *frame.ri* value differs from the station's *ri* assignment to this span. (The miscabled span assumes an SF error condition)
   The SF condition is cleared when an expected the station's expected *frame.ri* value is received.

c) Topology inconsistency. The topology database contains inconsistent information, as follows:
   1) Closed ring: The MAC address of a ringlet0 station *k* hops away matches the MAC address of the ringlet1 station *numStations* - *k* hops away.
   2) Open ring: A station with the same MAC address is located before the apparent edge station.

   (These checks are only applied after the topology has apparently stabilized.)
   If an inconsistency is found based on the above criteria, a defect is triggered to the operator for this topology inconsistency. If the topology inconsistency persists for 10 seconds, then an implementation may choose to stop adding or dropping all data frames from the ring.

d) Excess stations. The total number of stations appears to exceed MAX_STATIONS, because:
   1) Closed ring: A TP frame is received from a far distant station, with *frame.ttl* equal to one.
   2) Closed ring: The sum of the hop-count distances to each edge exceeds MAX_STATIONS-1.

   (These checks are only applied after the topology has apparently stabilized.)

e) Topology unstable. The topology was unstable for longer than the *instabilityDelay* value.
   This condition is cleared when both of the following stability conditions are true:
   1) No stability related database updates occurred within a *stabilityDelay* specified interval.
   2) All reachable stations on either ringlet have also been marked valid.

10.1.7 Adaptive behaviors

Some adaptive behaviors are based on knowledge of others' behaviors, communicated through TP frames or components of ATT frames, as follows:

a) If any non-jumbo frame station is reported through the *jp* (jumbo frame preferred) field of a TP frame, a jumbo-frame preferred station defaults to a non jumbo-frame preferred behavior.

b) Other stations' behaviors are affected by the contents of the *type*=ATT_STATION_SET entry within an ATT frame, as follows:
   1) If any station desires to receive data frames with invalid *fcs* fields, as indicated in the *bf* field of of this entry, other stations allow such frames to transit through them.
   2) If any station so desires to receive multi-choke fairness messages, as indicated in the *mu* field of this entry, multi-choke fairness frames (MCFF) are sent periodically.

10.2 Terminology and variables

10.2.1 Terminology

For clarity and conciseness of state-machine notation and routine definitions, multiple span-related variables are parameterized by ringlet identifier *ri*. The value of *ri* is mapped to the incoming ringlet associated with that span, so that information associated with west and east spans of a station correspond to ri values of 0 and 1, respectively, as illustrated in Figure 10.6.

As examples: *myTopoInfo.spanProtState[0]* contains the protection state of the west span of the station, which receives ringlet0 frames; *myTopoInfo.spanProtState[1]* contains the protection state of the east span of the station, which receives ringlet1 frames. This mapping is used uniformly for variables parameterized by ringlet identifier.

10.2.2 Common state machine definitions

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

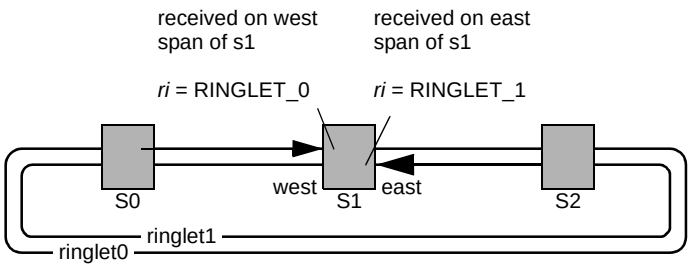


Figure 10.6—Relationship between *ri* value and east/west

CLOSED_RING
: Indicates that the topology is a closed ring, as defined in 3.2. When used to represent a binary value, the value represented is 1.

FS
: A (forced switch) administrative request condition configured by the operator (see 10.1.5.3).

IDLE
: The normal (idle) operating protection state of a span (see 10.1.5.3).

JUMBO
: Indicates that a ring supports jumbo frames, or that a station is configured to support jumbo frames if all other stations on the ring are configured to support jumbo frames (see 8.2). When representing a binary value, its value is 1.

MAX_STATIONS
: The maximum number of stations allowed on a ring. This is 255.

MS
: A (manual switch) administrative request condition configured by the operator (see 10.1.5.3).

OPEN_RING
: The topology is an open ring. When used to represent a binary value, the value represented is 0.

REGULAR
: Indicates that a ring does not support jumbo frames, or that a station is not configured to support jumbo frames (see 8.2). When representing a binary value, its value is 0.

SD
: A (signal degrade) protection status condition of a span (see 10.1.5.3).

SF
: A (signal fail) protection status condition of a span (see 10.1.5.3).

STABLE
: Indicates that the topology is stable.

STEERING
: Indicates that the protection type of the ring or the protection configuration of a station is steering, as defined in 3.2. When used to represent a binary value, the value represented is 0.

SHAKY
: Indicates that the topology is unstable.

WRAPPING
: Indicates that the protection type of the ring or the protection configuration of a station is wrapping, as defined in 3.2. When used to represent a binary value, the value represented is 1.

WTR
: A (wait to restore) protection state that occurs upon clearing of SF and SD (see 10.1.5.3).

10.2.3 Common state machine variables

atdTimerDelay

The configured value of the timer used for ATD frame transmission.

Range—[1 sec, 10 sec]

Resolution—1 sec

Default—1 sec

checksumMatch[*ri*]

Indicates whether this station's checksum is valid, its neighbor's checksum is valid, and this station's checksum matches that of its neighbor.

TRUE—This checksums are valid and match.

FALSE—(Otherwise).

containmentActive

Indication that context containment is active. This can be set by both stages of the topology database update. This is used by Clause 6.

TRUE—Context containment is active.

FALSE—(Otherwise.)

edgeChange

Indication that the edge state of a span of the station has changed in the protection state machine. This is an input to the second stage of the topology database update from the protection state machine.

TRUE—There is a change in edge state.

FALSE—(Otherwise.)

txFastDelay

The configured value of the timer used for fast TP and TC frame transmission.

Range—[1 ms, 20 ms]

Resolution—1 ms

Default—10 ms

lrttRequest

Indication from the topology validation state machine in Table 10.17 that the topology has become stable, and therefore that an LRTT measurement can commence (see Table 10.26) if the station is using conservative mode fairness.

TRUE—A new LRTT measurement can start.

FALSE—(Otherwise.)

needSecMacValidation

Indication from the topology validation state machine in Table 10.17 that the topology has just become valid, and that SecondaryValidation execution is required.

TRUE—The condition described above is in effect.

FALSE—(Otherwise.)

newNeighbor[*ri*]

Indication of a new neighbor on the span of the station with receive link on ringlet *ri*. This is an input to the protection state machine from the first stage of the topology database update.

TRUE—There is a new neighbor.

FALSE—(Otherwise.)

reversionMode

Configured to indicate if the station is in revertive or non-revertive protection mode.

TRUE—The station is in revertive mode.

FALSE—(Otherwise.)

(default): Shall behave as though set to FALSE.

ri

Receive ringlet identifier corresponding to a physical span of a station. For state machines that run on one span of a station, *ri* maps to that span of the station.

rxTpFrame

The contents of a TP frame received from the ring, and fields within the frame further defined in 10.3.1.

txSlowDelay

The configured value of the timer used for slow TP and TC frame transmission.

Range—[100 ms, 1 sec]
Resolution—100 ms
Default—100 ms

spanProtAdmin[*ri*]

Administrative state of the span of the station with receive link on ringlet *ri*.

IDLE—Normal operation, span in use.
MS—Indicates a desire to deactivate the span.
FS—Indicates a mandate to deactivate the span.

spanProtStatus[*ri*]

Protection status of the span of the station with receive link on ringlet *ri*.

IDLE—None or an insignificant error rate.
SD—A significant error rate indicating a degraded span (signal degrade).
SF—A unacceptable error rate indicating a span has failed (signal fail).

tempHopsRx[*ri*]

Temporary value of hops in the receive direction on ringlet *ri* before reaching oneself or before reaching an edge. This variable is updated during the topology validation process.

keepaliveEvent[*ri*]

An indication that a validated keepalive (a received fairness frame) has been received.

topoChanged

Indication from the topology database update state machines to the topology validation state machine in Table 10.17 of whether stability-impacting portions of the topology have changed.

TRUE—Station positions and edge information in the topology database have changed.
FALSE—(Otherwise.)

topoStability

Indication from the topology validation state machine in Table 10.17 of whether the topology is stable.

STABLE—The topology is stable.
SHAKY—(Otherwise.)

topoType

Variable that stores a computed value of ring topology type before topology is validated. At that point, the value of *topoType* is set into the topology database.

CLOSED_RING—The topology is a closed ring.
OPEN_RING—(Otherwise.)

topoValid

Indication from the topology validation state machine in Table 10.17 of whether the topology is valid.

TRUE—The topology is valid.
FALSE—(Otherwise.)

totalHopsRx[*ri*]

Hops in the receive direction on ringlet *ri* before reaching oneself or before reaching an edge.

tpContentChange

Indication from the second stage of the topology database update to the TP frame transmit state machine in Table 10.18 of whether the sequence number in transmitted TP frames should be incremented due to a change in jumbo preference or wrap configuration information.

TRUE—The sequence number should be incremented.
FALSE—(Otherwise.)

transmitTcFrame

Indication from the topology validation state machine in Table 10.17 that the topology is valid, and therefore that the topology checksum can be computed at a station and communicated to its neighbor stations using the TC frame.

TRUE—The topology is valid.

FALSE—(Otherwise.)

transmitTpFrame

The trigger for immediately transmitting TP frames, followed by periodic transmission using the fast protection timer. The *transmitTpFrame* value can be set TRUE by TopologyUpdate1 and ProtectionUpdate state machines, and (after processing) cleared to FALSE by the TransmitTpFrame state machine.

TRUE—Immediate transmission of TP frames is triggered.

FALSE—Transmission of TP frames is not triggered.

10.2.4 *ringInfo* fields

The topology database is divided into *ringInfo*, *myTopoInfo*, and *topoEntry* portions as categorized in 10.2.4, 10.2.5, and 10.2.6, respectively, as discussed in 10.5. The *ringInfo* portion of that database includes the following field values:

badFcsUser

Indicates whether any station on the ring copies data frames with invalid *fcs* fields to the client.

TRUE—At least one station that copies data frames with invalid *fcs* fields to the client.

FALSE—(Otherwise.)

jumboType

The indication whether a ring supports jumbo frames.

JUMBO—The ring supports jumbo frames.

REGULAR—(Otherwise.)

multichokeUser[*ri*]

Indication whether any station expects to receive multichoke fairness frames on ringlet *ri*.

TRUE—At least one station that expects to receive multichoke fairness frames on ringlet *ri*.

FALSE—(Otherwise.)

mtuSize

An indication of the MTU size on the ring.

JUMBO_MAX—The MTU size on the ring is JUMBO_MAX.

REGULAR_MAX—The MTU size on the ring is REGULAR_MAX.

numStations

Total number of stations on the ring.

topoType

The indication of the ring topology type.

OPEN_RING—Indicates an open ring topology.

CLOSED_RING—Indicates a closed ring topology.

totalHopsTx[*ri*]

Hops in the transmit direction on ringlet *ri* before reaching oneself or before reaching an edge.

unreservedRate[*ri*]

The difference between LINK_RATE and the total reserved bandwidth on ringlet *ri*, as computed in 10.4.2.2.1.

10.2.5 *myTopoInfo* fields

The topology database is divided into *ringInfo*, *myTopoInfo*, and *topoEntry* portions as categorized in 10.2.4, 10.2.5, and 10.2.6, respectively, as discussed in 10.5. The *myTopoInfo* portion of that database includes the following field values:

badFcsUser

Indicates whether the station copies data frames with invalid *fcs* fields to the client.

TRUE—The station copies data frames with invalid *fcs* fields to the client.

FALSE—(Otherwise.)

checksum

Topology checksum.

checksumValid

Indicates whether the topology checksum of the station is valid.

TRUE—The topology checksum is valid.

FALSE—(Otherwise.)

conservativeMode

Indicates whether this station is configured to run conservative mode.

TRUE—The station is configured to run conservative mode.

FALSE—The station is configured to run aggressive mode.

interfaceIndex

Interface index.

jumboPrefer

Indicates whether this station prefers to support jumbo frames. Jumbo frames are defined in 8.2.1.

JUMBO—The station prefers to support jumbo frames.

REGULAR—(Otherwise.)

lastNeighborMac[*ri*]

MAC address of the last known neighbor station on the span of the station with receive link on ringlet *ri*.

macAddress

MAC address of the station.

managementAddressType

Management IP address type, as defined in 10.4.5.

managementIpAddr

Management IP address.

multichokeUser

Indicates whether this station expects to receive multichoke fairness frames.

TRUE—This station expects to receive multichoke fairness frames.

FALSE—(Otherwise.)

neighborChecksum[*ri*]

Topology checksum of the neighbor station connected on the span of the station with receive link on ringlet *ri*.

neighborCheckValid[*ri*]

Indicates whether this station's neighbor (on the receive link of ringlet) has a valid topology checksum (contained in *neighborChecksum[ri]*).

TRUE—The checksum is valid.

FALSE—(Otherwise.)

protConfig

Type of protection configuration.

STEERING—Indicates the station configured for steering.

WRAPPING—Indicates the station is configured for wrapping.

reservedRate[*ri*]

Reserved (subclassA0) bandwidth on ringlet *ri*.

secMacAddress1

First secondary MAC address.

secMacAddress2

Second secondary MAC address.

secMacAddr1Used
Indicates whether the first secondary MAC address of this station is in use.
TRUE—The address is in use.
FALSE—(Otherwise.)
secMacAddr2Used
Indicates whether the second secondary MAC address of this station is in use.
TRUE—The address is in use.
FALSE—(Otherwise.)
sequenceNumber
Sequence number used in transmitted TP frames.
spanEdgeState[*ri*]
Edge state on the span of the station with receive link on ringlet *ri.*
TRUE—An edge is present on the span of the station with receive link on ringlet *ri*.
FALSE—(Otherwise.)
spanProtState[*ri*]
Protection state on the span of the station with receive link on ringlet *ri*.
stationName
Station name.
weight[*ri*]
Station weight on ringlet *ri*.

10.2.6 *topoEntry*[*ri*][*hops*] fields

The topology database is divided into *ringInfo*, *myTopoInfo*, and *topoEntry* portions as categorized in 10.2.4, 10.2.5, and 10.2.6, respectively, as discussed in 10.5. Each *myTopoInfo*[*ri*][*hops*] entry within that database includes the following field values:

badFcsUser
Indicates whether a station copies data frames with invalid *fcs* fields to the client.
TRUE—Station copies data frames with invalid *fcs* fields to the client.
FALSE—(Otherwise.)
conservativeMode
Indicates whether a station is configured to run conservative mode.
TRUE—Station is configured to run conservative mode.
FALSE—The station is configured to run aggressive mode.
holdOffTimerValue
A configurable value for each per span that delays the generation of protection triggers.
Range—[200 ms, 10 ms]. A zero value is also allowed.
Resolution—10 ms
Default—zero
hopsRx
Hop count separating the source of the TP frames and the station on which the topology database is stored.
interfaceIndex
Interface index.
jumboPrefer
Indicates a station that prefers to support jumbo frames.
JUMBO—Indicates the station prefers to support jumbo frames.
REGULAR—(Otherwise.)
lrtt
Loop round trip time measured to/from the station at location *index*.
macAddress
MAC address of the station.

managementAddressType
Management IP address type, as defined in 10.4.5.
managementIpAddr
Management IP address.
multichokeUser
Indicates whether a station expects to receive multichoke fairness frames.
TRUE—Station expects to receive multichoke fairness frames.
FALSE—(Otherwise.).
protConfig
Type of protection configuration.
STEERING—Indicates a steering configured station.
WRAPPING—Indicates a wrapping configured station.
reachable
Reachability of the station at location *index* from the station on which the topology database is stored on ringlet *Other*(*ri*).
TRUE—Station is reachable.
FALSE—(Otherwise.)
reservedRate[*ri*]
Reserved (subclassA0) bandwidth on ringlet *ri*.
secMacAddress1
First secondary MAC address.
secMacAddress2
Second secondary MAC address.
secMacAddr1Used
Indicates whether the first secondary MAC address of the station is in use.
TRUE—The address is in use.
FALSE—(Otherwise.)
secMacAddr2Used
Indicates whether the second secondary MAC address of the station is in use.
TRUE—The address is in use.
FALSE—(Otherwise.)
sequenceNumber
Sequence number contained in TP frame received on ringlet *ri*.
spanEdgeState[*ri*]
Indicates whether an edge is present on the span of the station with receive link on ringlet *ri*.
TRUE—Edge is present on the span of the station with receive link on ringlet *ri*.
FALSE—(Otherwise).
spanProtState[*ri*]
Protection state on the span of the station with receive link on ringlet *ri*.
stationName
Station name.
valid
Provides an indication of whether this entry is valid.
TRUE—This entry is valid.
FALSE—(Otherwise.)
weight[*ri*]
Station weight on ringlet *ri*.

10.2.7 Common state machine routines

ClearAtdInfo(*ri, hop*)
Clears all ATD information associated with this entry in the topology database.

FindIndex(*frame*)

Determines the topology database index value corresponding to the received TP frame, as defined by Equation 10.1.

```
(MAX_STATIONS - frame.ttl + 1)
```
(10.1)

JumboAdminRequest( )

Provides the value of a new administrative jumbo preference request.

JUMBO—Indicates a station that prefers to support jumbo frames.

REGULAR—Indicates a station that does not support jumbo frames.

(null)—No jumbo administrative request is available.

MatchAllStationMacToSecMac()

Indicates that a secondary MAC address of the local station matches a station MAC address of a valid and reachable station in the topology database, as defined by Equation 10.2.

TRUE—A station MAC address duplicates a secondary MAC address of the local station.

FALSE—(Otherwise.)

(10.2)

```
Boolean
MatchAllStationMacToSecMac()
{
    int ringlet, hop;

    if (myTopoInfo.secMacAddress1 == myTopoInfo.macAddress)
        return(TRUE);
    if (myTopoInfo.secMacAddress2 == myTopoInfo.macAddress)
        return(TRUE);
    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hop = 1; hop <= ringInfo.totalHopsTx[Other(ringlet)]; hop += 1)
        {
            if (!topoEntry[ringlet][hop].valid)
                continue;
            if (topoEntry[ringlet][hop].macAddress == myTopoInfo.secMacAddress1)
                return(TRUE);
            if (topoEntry[ringlet][hop].macAddress == myTopoInfo.secMacAddress2)
                return(TRUE);
        }
    return(FALSE);
}
```

MismatchedProtection()

Indicates a presence of mismatched protection configuration stations, as defined by Equation 10.3.

TRUE—There is a station with a mismatched protection configuration on the ring.

FALSE—(Otherwise.)

(10.3)

```
Boolean
MismatchedProtection()
{
    int ringlet, hop, protConfig;

    protConfig = myTopoInfo.protConfig;
    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hop = 1; hop <= MAX_STATIONS; hop += 1)
        {
            if (!topoEntry[ringlet][hop].valid)
                continue;
            if (topoEntry[ringlet][hop].protConfig != protConfig)
                return(TRUE);
        }
    return(FALSE);
}
```

ProtAdminRequestCheck( )
Indicates whether *ProtAdminRequest*( ) would return a non-null value.
TRUE—A non-null value would be returned.
FALSE—(Otherwise.)

TimerDone(*timeout*)
Indicates whether the timeout time has been reached, as specified by Equation 10.4.

```
((currentTime - timeout) >= 0)
```
(10.4)

TimerBad(*timeout*, *delay*)
Indicates whether the timeout value has been incorrectly initialized, as specified by Equation 10.5.

```
((timeout - currentTime) > delay)
```
(10.5)

TopoStateMachineStage2(*ri*)
Executes the second stage of the topology database update based on updates from the protection state machine running on the span of the station with receive link on ringlet *ri*.

10.2.8 Variables and routines defined in other clauses

This clause references the following variables and routines defined in Clause 6.

currentTime
Other(*ri*)

This clause references the following variables defined in Clause 7.

LINK_STATUS

This clause references the following literals defined in Clause 8.

JUMBO_MAX
REGULAR_MAX

This clause references the following literals defined in Clause 9.

LINK_RATE

This clause references the following variables defined in Clause 9.

advertisingInterval

10.2.9 Defect indications

The following defect indications shall be supported, as described in the subclauses listed below.

duplicateSecMacAddressDefect
The duplicate secondary MAC address defect (local station only) is specified in 10.5.2. It is of critical severity.
TRUE—A duplicate secondary MAC address defect is present.
FALSE—(Otherwise.)

excessReservedRateDefect

This defect indicates that the amount of reserved bandwidth on a ringlet exceeds the available LINK_RATE.

TRUE—An excess reserved rate defect is present.

FALSE—(Otherwise.)

lrttIncompleteDefect

A critical severity defect that indicates the LRTT measurement for the ring has not been completed within the *lrttDelay* specified time, as specified in 10.6.15.4.

TRUE—An LRTT incomplete defect is present.

FALSE—(Otherwise.)

maxSecMacAddressDefect

The maximum number of secondary MAC addresses on the ring exceeded defect is specified in 10.5.2. It is of critical severity.

maxStationsDefect

The maximum number of stations exceeded critical severity defect.

TRUE—A maximum stations defect is present.

FALSE—(Otherwise.)

miscablingDefect[*ri*]

The critical-severity miscabling defect associated with the *ri*-selected span.

TRUE—A miscabling defect is present on the *ri*-selected span.

FALSE—(Otherwise.)

protMisconfigDefect

A critical severity defect that indicates the presence of mismatched-protection-configuration stations, based on the value returned by *MismatchedProtection*().

TRUE—A protection configuration defect is present on the station.

FALSE—(Otherwise.)

topoInconsistencyDefect

The inconsistent topology defect is specified in 10.7.3. It is of critical severity.

TRUE—An inconsistent topology defect is present.

FALSE—(Otherwise.)

topoInstabilityDefect

The critical-severity topology instability defect.

TRUE—A topology instability defect is present.

FALSE—(Otherwise.)

10.3 Frame formats

10.3.1 Topology and protection (TP) frame format

The fixed-length TP frame is identified by the *controlType*=CT_TOPO_PROT field and is broadcast to minimize transmission delays. The TP frame contents include information for signaling link protection state, for discovery of the physical topology of the ring, and for reporting station preferences information, as illustrated in Figure 10.7.

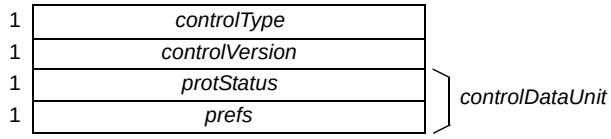


Figure 10.7—TP payload format

The header fields (specified in 8.3, but not illustrated here) are additionally constrained as specified in Table 10.2.

Table 10.2—TP frame header field-value restrictions

Field	Subfield	Value	Description
ttl	—	MAX_STATIONS	Allows TP frames to propagate through all stations
baseControl	*sc*	CLASS_A0	Sent as subclassA0
	fe	FALSE	Not fairness eligible
	we	0	These frames are never wrapped
da	—	FF-FF-FF-FF-FF-FF	Allows TP frames to be received by any station

10.3.1.1 *protStatus:* An 8-bit field specified in 10.3.1.3.

10.3.1.2 *prefs:* An 8-bit field specified 10.3.1.4.

10.3.1.3 protStatus

The subfields of the 8-bit *protStatus* (protection status) field are shown in Figure 10.8 and described thereafter.

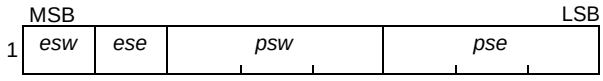


Figure 10.8—*protStatus* field format

10.3.1.3.1 *esw*: A (edge state, west) bit that indicates whether an edge is present on the west span of a station. A value of 0 indicates that there is no edge, while a value of 1 indicates that an edge is present.

10.3.1.3.2 *ese*: A (edge state, east) bit that indicates whether an edge is present on the east span of a station. A value of 0 indicates that there is no edge, while a value of 1 indicates that an edge is present.

10.3.1.3.3 *psw*: A 3-bit (protection state, west) field indicates the protection state on the west span of the station, as defined in the right columns of Table 10.3.

10.3.1.3.4 *pse*: A 3-bit (protection state, east) field field indicates the protection state on the east span of the station, as defined in the right columns of Table 10.3.

Table 10.3—*psw* and *pse* values

spanProtAdmin	*spanProtStatus*	*currentTime* < *wrtTimeout*	Value	Name	Description
IDLE	IDLE	TRUE	000_2	IDLE	No request
		FALSE	001_2	WTR	Wait-to-restore
MS	IDLE	—	010_2	MS	Manual switch
MS, IDLE	SD	—	011_2	SD	Signal degrade
	SF	—	100_2	SF	Signal fail
FS	—	—	101_2	FS	Forced switch
—	—	—	110_2	—	Reserved
—	—	—	111_2	—	Reserved

The *spanProtAdmin* administrative state field is set by client directives. The *spanProtStatus* protection status field is derived from physical layer status (reported via PHY_LINK_STATUS.indication as defined in 7.2.3.2), loss of keepalives, and miscabling status. These values and the state of the wait to restore (WTR) timer are combined to form the *psw* and *pse* field values, as specified in Table 10.3.

10.3.1.4 *prefs*

The subfields of the 8-bit *prefs* (preferences) field are illustrated in Figure 10.9 and described thereafter.

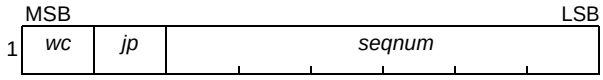


Figure 10.9—*prefs* field format

10.3.1.4.1 *wc*: A (wrap protection configured) bit that is set based on the operator configured protection configuration stored in *myTopoInfo.protConfig*, as defined in 10.2.5.

10.3.1.4.2 *jp*: A (jumbo frame preferred) bit that is set based on the operator specified jumbo preference stored in *myTopoInfo.jumboPrefer*, as defined in 10.2.5.

10.3.1.4.3 *seqnum*: A 6-bit (sequence number) field that is incremented each time other portions of the *controlDataUnit* have changed, as specified in 10.6.7.

10.3.2 Topology checksum (TC) frame format

The fixed-length broadcast TC frames are identified by the *controlType*=CT_TOPO_CHKSUM field in the frame. These TC frames communicate the topology checksum (defined by Equation 10.19) to neighbor stations so that the stations can determine whether the relevant portions of their topology databases match, and therefore, whether they can exit context containment, as illustrated in Figure 10.10

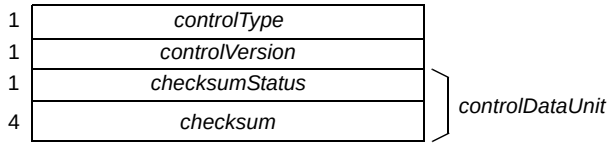


Figure 10.10—TC frame format

The header fields (specified in 8.3, but not illustrated here) are additionally constrained as specified in Table 10.4. A *ttl=1* field value forces downstream stripping, so these frames never travel beyond the neighbor station.

Table 10.4—TC frame header field-value restrictions

Field	Subfield	Value	Description
ttl	—	1	Ensures TC frames do not propagate beyond neighbor station
baseControl	*sc*	CLASS_A0	Sent as subclassA0
	fe	FALSE	Not fairness eligible
	we	0	These frames are never wrapped
da	—	FF-FF-FF-FF-FF-FF	Allows TC frames to be received by neighbor station

10.3.2.1 *checksumStatus*: An 8-bit field defined in 10.3.2.3.

10.3.2.2 *checksum*: An 32-bit field shown in Figure 10.10. The value in this field is a checksum computed from a subset of fields in the topology database, as described in 10.5.

10.3.2.3 *checksumStatus*

The 8-bit *checksumStatus* (checksum status) field is shown in Figure 10.11. The sub-fields of the *checksumStatus* field are described in this subclause.

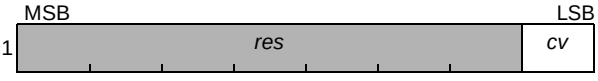


Figure 10.11—*checksumStatus* field format

10.3.2.4 *res*: A 7-bit (reserved) field.

10.3.2.5 *cv*: A (checksum valid) bit that indicates that the value in the *checksum* field is valid. The value in this field is set to 1 if the value in the *checksum* field is valid, and is set to 0 otherwise.

10.3.3 Loop round trip time request frame format

The fixed-length loop round trip time (LRTT) request control frames are identified by the *controlType*=CT_LRTT_REQ field in the frame. Stations using conservative rate adjustment shall generate loop round trip time request frames. All stations shall perform loop round trip time request frame receipt processing. The loop round trip time request frame has a 14-byte payload, as illustrated in Figure 10.12.

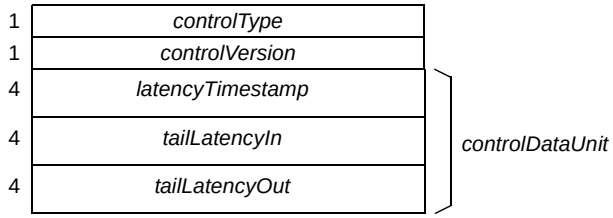


Figure 10.12—LRTT request frame payload

The header fields (specified in 8.3, but not illustrated here) are additionally constrained as specified in Table 10.5.

Table 10.5—LRTT request frame header field-value restrictions

Field	Subfield	Value	Description
baseControl	*sc*	CLASS_A0	Sent as subclass A0
	fe	FALSE	Not fairness eligible
	we	0	These frames are never wrapped
da	—	(depends)	Unicast address of the station whose LRTT is being measured

10.3.3.1 *latencyTimestamp*: A 32-bit field that contains a local timestamp generated by the requestor at the time of transmitting this frame. The interpretation of this field is local to the requestor and has no meaning to other stations.

10.3.3.2 *tailLatencyIn*: A 32-bit field provided for the responder's use when reflecting the frame as a response frame. The requestor shall clear this field to zero.

10.3.3.3 *tailLatencyOut*: A 32-bit field provided for the responder's use when reflecting the frame as a response frame. The requestor shall clear this field to zero.

10.3.4 Loop round trip time response frame format

The fixed-length loop round trip time (LRTT) response control frames are identified by the *controlType*=CT_LRTT_RSP field in the frame. All stations shall perform loop round trip time response frame generation. The loop round trip time response frame has a 14-byte payload, as illustrated in Figure 10.13.

The header fields (specified in 8.3, but not illustrated here) are additionally constrained as specified in Table 10.5.

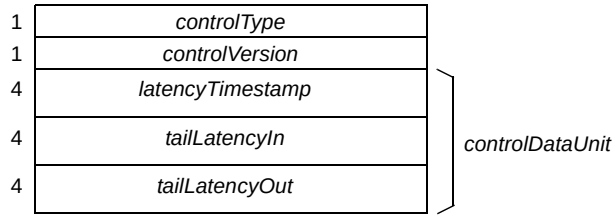


Figure 10.13—LRTT response frame payload

Table 10.6—LRTT response frame header field-value restrictions

Field	Subfield	Value	Description
baseControl	*sc*	CLASS_A0	Sent as subclassA0
	fe	FALSE	Not fairness eligible
	we	0	These frames are never wrapped
da	—	*sa* from request	The unicast address of the station measuring the LRTT

10.3.4.1 *latencyTimestamp*: A copy of the like-named 32-bit field from the last received loop round trip time request frame.

10.3.4.2 *tailLatencyIn*: A 32-bit field provided to allow a responding station to mark the time at which it received the loop round trip time request frame for which this frame is a response, in order to improve the estimate of the amount of latency incurred in processing the frame at the responding station. If the responding station has the ability to place a timestamp in the frame at the time of receipt, then this field contains a timestamp expressed in microseconds from an epoch local to the responding station. If the tail station is unable to place a timestamp in the frame on receipt, then it shall fill this field with a value of all zeros.

10.3.4.3 *tailLatencyOut*: A 32-bit field provided to allow a responding station to mark the time at which it transmitted the loop round trip time response frame, in order to improve the estimate of the amount of latency incurred in processing the frame at the responding station. If the responder placed a non-zero timestamp in the *tailLatencyIn* field at the time of receipt, then this field contains a timestamp at the time of transmission, expressed in microseconds from an epoch local to the responding station. If the responder placed a value of zero in the *tailLatencyIn* field at the time of receipt, then this field contains an estimate of the total latency incurred by this frame at the responder, expressed in microseconds. If the responder is unable to estimate or measure the latency incurred, it shall clear this field to zero.

NOTE—While the latency is best kept as close to 0 as possible, and the measurement or estimate of the latency is best kept as accurate as possible, estimations that overestimate the latency yield better fairness response than estimations that underestimate the latency.

10.3.5 ATD frame format

The variable-length ATD control frames are identified by the *controlType*=CT_STATION_ATD field in the frame. In comparison to TP frames, these ATD frames contain less time-critical information and are sent less often. The ATD frame's payload provides type-length-value (TLV) encoded parameters to other stations, as illustrated in Figure 10.14.

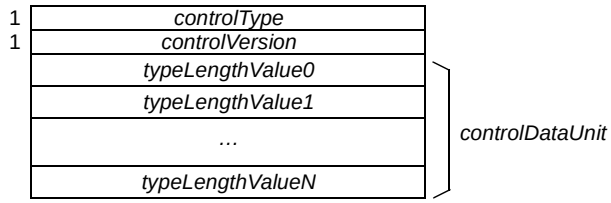


Figure 10.14—Attribute discovery frame format

The header fields (specified in 8.3, but not illustrated here) are additionally constrained as specified in Table 10.7.

Table 10.7—ATD frame header field-value restrictions

Field	Subfield	Value	Description
ttl	—	MAX_STATIONS	Allows ATD frames to propagate through all stations
baseControl	*sc*	CLASS_A0	Sent as subclassA0
	fe	FALSE	Not fairness eligible
	we	0	These frames are never wrapped
da	—	FF-FF-FF-FF-FF-FF	Allows ATD frames to be received by any station

A type-length-value encoding scheme is used to encode additional station attributes (ATTs) in ATD frames, as illustrated in Figure 10.15. Within an ATD frame, the ATTs shown in Table 10.8 can appear in any order.

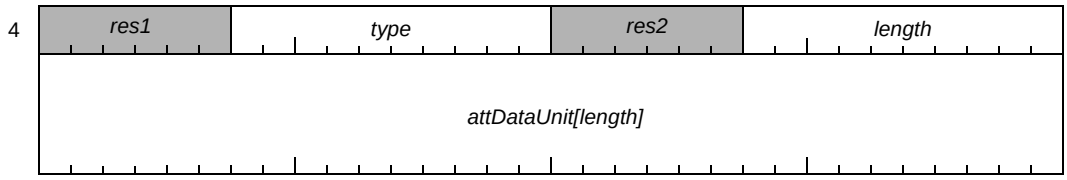


Figure 10.15—Type-length-value format

10.3.5.1 *res1*: A 6-bit reserved field.

10.3.5.2 *type*: A 10-bit field that identifies the ATT type, as specified in Table 10.8. All ATT types shown in Table 10.8 shall be usable within the ATD frame.

10.3.5.3 *res2*: A 6-bit reserved field.

10.3.5.4 *length*: The length, in bytes, of the following *typeDependent* value information. The value of *length* shall be less than 1024.

10.3.5.5 *attDataUnit*: Data whose format and function is specified by the preceding *type* field. The length of this field is less than 1024 bytes.

Table 10.8—*type* encoding for ATD frame

Value	Name	Size	Reference	Support	Description
0	—	—	—	—	Reserved
1	ATT_WEIGHT	2	10.4.1	C	Fairness weights of stations
2	ATT_BANDWIDTH	4	10.4.2	C	Reserved classA0 bandwidth
3	ATT_STATION_SET	1	10.4.3	C	Station settings
4	ATT_STATION_NAME	0-127	10.4.4	O	Station name (an ASCII string)
5	ATT_MANAGE_ADDRESS	5 or 17	10.4.5	O	Management IP address
6	ATT_INTERFACE_INDEX	4	10.4.6	O	*ifIndex* of the ring interface
7	ATT_SECONDARY_MACS	12	10.4.7	C	Secondary MAC addresses
8-1022	—	—	—	—	Reserved
1023	ATT_ORG_SPECIFIC	8-1023	10.4.8	O	Nonstandard (organization specific) values

Legend:
O—Optional
C—Conditional (mandatory if value is different from default value, otherwise optional)

10.4 Defined ATT encodings

10.4.1 Weight ATT

The weight ATT is identified by its distinctive *type*=ATT_WEIGHT field value (see 10.3.5). The weight ATT encodes the station configurable fairness weight in each ringlet (see 9.3.1), as illustrated in Figure 10.16. The consumer of the information contained in the weight ATT is the fairness module, which needs this information to compute the sum of the weights of active stations in 9.4.

Figure 10.16—Weight *attDataUnit* format

10.4.1.1 *ringlet0Weight*: Specify the fairness weight of the station on ringlet0.

10.4.1.2 *ringlet1Weight*: Specify the fairness weight of the station on ringlet1.

If a station has a weight not equal to 1 on either ringlet, it shall advertise the weight via the weight ATT. If a station has a weight equal to 1 on both ringlets, it may advertise the weight via the weight ATT. If a weight ATT is not included in the ATD frames sent by a given source station, then the station's weight on each ringlet is assumed to have a default value of 1 by receiving stations.

10.4.2 Station bandwidth ATT

The station bandwidth ATT is identified by its distinctive *type*=ATT_BANDWIDTH field value (see 10.3.5). The station bandwidth ATT encodes the source station's reserved subclassA0 transmit bandwidth on each ringlet, as illustrated in Figure 10.17.

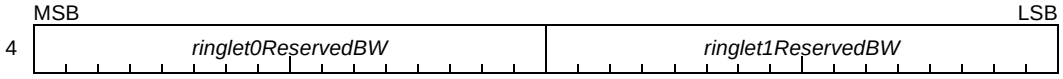


Figure 10.17—Station bandwidth *attDataUnit* format

10.4.2.1 *ringlet0ReservedBW*: A 16-bit field that specifies the station's reserved subclassA0 bandwidth allocation on ringlet0.

10.4.2.2 *ringlet0ReservedBW*: A 16-bit field that specifies the station's reserved subclassA0 bandwidth allocation on ringlet1.

The station bandwidth ATT shall be included in each ATD frame for non-zero values of reserved bandwidth. It may be included in each ATD frame for a zero value of station reserved bandwidth on both ringlets. If the station bandwidth ATT is not included in the ATD frame sent by a given source, then the station reserved bandwidth on both ringlets is assumed to have a default value of zero by receiving stations.

Each station then sums up the reserved bandwidth reported by each station to determine the total reserved subclassA0 bandwidth on each ringlet. The knowledge of the per-ringlet total reserved bandwidth is used by the fairness module (see 9.4).

The reserved bandwidth shall use the same normalization as the *fairRate* field in fairness frames defined in 9.3, with WEIGHT equal to 1. The purpose of this normalization is so that reserved bandwidth values are made available to the fairness module in the most convenient form possible.

10.4.2.2.1 Computation of total unreserved bandwidth

The total unreserved bandwidth on a given ringlet (used to fill in *ringInfo.unreservedRate* for each ringlet) is computed as defined by Equation 10.6:

```
void                                                                    (10.6)
ComputeTotalUnreservedRate()
{
    int ringlet, hop, resRate[2];

    resRate[0] = myTopoInfo.reservedRate[0];
    resRate[1] = myTopoInfo.reservedRate[1];
    switch(ringInfo.topoType)
    {
    case CLOSED_RING:
        for (ringlet = 0; ringlet < 2; ringlet += 1)
          for (hop = 1; hop <= ringInfo.totalHopsTx[Other(ringlet)]; hop += 1)
            resRate[Other(ringlet)] +=
                 topoEntry[ringlet][hop].reservedRate[Other(ringlet)];
        break;
    case OPEN_RING:
        for (ringlet = 0; ringlet < 2; ringlet += 1)
          for (hop = 1; hop <= ringInfo.totalHopsTx[Other(ringlet)]; hop += 1)
          {
            resRate[Other(ringlet)] +=
                 topoEntry[ringlet][hop].reservedRate[Other(ringlet)];
            resRate[ringlet] += topoEntry[ringlet][hop].reservedRate[ringlet];
          }
        break;
    }
    ringInfo.unreservedRate[0] = LINK_RATE - resRate[0];
    ringInfo.unreservedRate[1] = LINK_RATE - resRate[1];
}
```

In the event that the total unreserved bandwidth is less than zero on either ringlet, an excess reserved rate defect shall be declared by setting *excessReservedRateDefect* to TRUE. This defect is cleared when the excess reserved bandwidth is removed.

10.4.3 Station settings ATT

The station settings ATT is identified by its distinctive *type*=ATT_STATION_SET field value (see 10.3.5). The station settings ATT encodes several station settings, as illustrated in Figure 10.18.

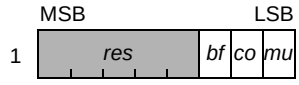


Figure 10.18—Station settings *attDataUnit* format

10.4.3.1 *res*: A 6-bit reserved field.

10.4.3.2 *bf*: A (bad frame) bit that is set to 1 if the station sends data frames with invalid *fcs* fields to the client, based on the value of *myTopoInfo.badFcsUser* defined in 10.2.5. Otherwise, this bit is cleared to 0.

10.4.3.3 *co*: A (conservative option) bit that is set to 1 if the station is configured to use conservative mode fairness, based on the value of *conservativeMode* defined in 9.2.2.2. Otherwise, this bit is cleared to 0.

10.4.3.4 *mu*: A (multichoke update) bit that is set to 1 if the station expects to receive multi-choke fairness frames from all other stations on the ring, based on the value of *multiChokeIndOption* defined in 9.2.2.2. Otherwise, this bit is cleared to 0.

The station settings ATT shall be included in each ATD frame, if any of the *bf*, *co*, or *mu* fields would be set to 1. It may be included in each ATD frame if the *bf*, *co* and *mu* fields would be set to 0. If the station settings ATT is not included in the ATD frame, then these fields are assumed to have a default value of zero by receiving stations.

For the local station, this information is stored in the topology database in the fields *myTopoInfo.badFcsUser*, *myTopoInfo.conservativeMode*, and *myTopoInfo.multichokeUser*, as defined in 10.2.5. Upon receipt of this ATT from another station on the ring, this information is stored in the corresponding fields of the topology database within *topoEntry.*

As defined in 10.2.4, *ringInfo.multichokeUser*[*ri*] indicates if any station expects to receive multi-choke fairness frames on ringlet *ri*, and therefore that every station transmits multi-choke fairness frames onto ringlet *ri*. The *ringInfo.badFcsUser* bit indicates if any station on the ring copies data frames with invalid *fcs* fields to the client, and therefore that every station transits data frames with invalid *fcs*.

10.4.4 Station name ATT

The variable-length station name ATT is identified by its distinctive *type*=ATT_STATION_NAME value (see 10.3.5). The station name ATT encodes the first 127 characters of the operator assigned *stationName value*, as illustrated in Figure 10.19.

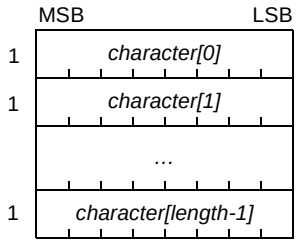


Figure 10.19—Station name *attDataUnit* format

***character[n]*:** A sequence of 8-bit fields that identify the first through last ASCII characters corresponding to the station's name (a NULL character termination is not provided).

The *stationName* characters follow the definition of the RFC 3418 sysName object.

The station name ATT may be included in any ATD frame. If a station name ATT is not included in the ATD frames sent by a given source station, then the station name is assumed to have a default value of an empty string by receiving stations.

The topology module uses this information to fill in the *stationName* field in the topology and status database (see 10.5).

10.4.5 Management address ATT

The variable-length management address ATT is identified by its distinctive *type*=ATT_MANAGE_ADDRESS value (see 10.3.5). The management address ATT encodes the internet protocol (IP) address of the management interface of the station, as illustrated in Figure 10.20.

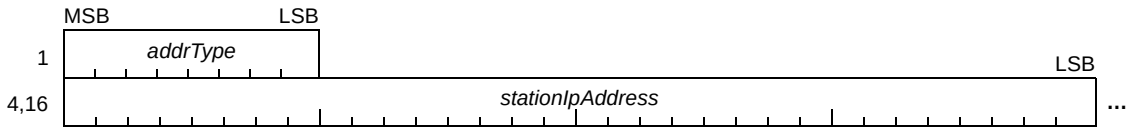


Figure 10.20—Management address ATT *attDataUnit* format

10.4.5.1 *addrType*: An 8-bit field that carries the internet address type identifier of the following IP address, as defined in RFC 3291. The mapping of these values to supported address types is shown in Table 10.9.

10.4.5.2 *stationIpAddress*: The 32-bit or 128-bit *stationIpAddress* field carries the IP address of the management interface of the station. If the *managementAddressType* field indicates an IPv4 address, *stationIpAddress* is a 32-bit field. If the *managementAddressType* field indicates an IPv6 address, *stationIpAddress* is a 128-bit field.

Table 10.9—Mapping of address type values to supported address types

Value	Name	Address length	Description
1	*ipv4*	4	Internet Protocol version 4 address
2	*ipv6*	16	Internet Protocol version 6 address

The station management address ATT may be included in each ATD frame. If a station management address ATT is not included in the ATD frames sent by a given source station, then the station management address is assumed to have a default value of all zero by receiving stations.

10.4.6 Station interface index ATT

The station interface index ATT is identified by its distinctive *type*=ATT_INTERFACE_INDEX value (see 10.3.5). The station interface index ATT encodes the *rprIfIndex* attribute from the RPR MIB, and is illustrated in Figure 10.21. The station interface index may be included in any ATD frame.

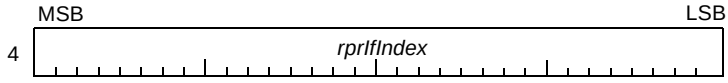


Figure 10.21—Station interface index ATT *attDataUnit* format

10.4.6.1 *rprIfIndex*: A 32-bit field that contains the RFC 2863 *ifIndex* value of the RPR interface of a station.

If a station interface index ATT is included in the ATD frames sent by a given source station, then the station interface index is assumed to have a default value of 0 by receiving stations.

10.4.7 Secondary MAC ATT

The secondary MAC ATT is identified by its distinctive *type*=ATT_SECONDARY_MACS value (see 10.22). This ATT supplies two 48-bit secondary MAC addresses, as illustrated in Figure 10.22.

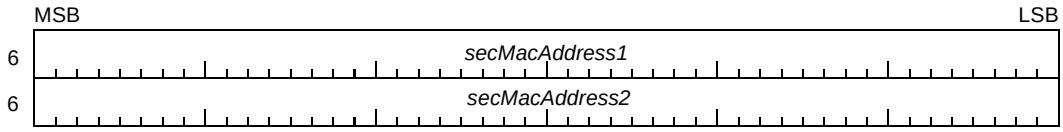


Figure 10.22—Secondary MAC ATT *attDataUnit* format

10.4.7.1 *secMacAddress1*: A 48-bit field that identifies the first secondary MAC address used by a station. If the station does not have any secondary MAC address configured, *secMacAddress1* shall be zero.

10.4.7.2 *secMacAddress2*: A 48-bit field that identifies the second secondary MAC address used by a station. If the station does not have any secondary MAC address configured, *secMacAddress2* shall be zero.

Secondary MAC addresses are reported in ATD frames in the same manner that they are stored locally in the topology database. The secondary MAC ATT shall be included in each ATD frame if any secondary MAC address is registered on the station. It may be included in each ATD frame otherwise. If the secondary MAC ATT is not included in the ATD frame sent by a given source, then *secMacAddress1* and *secMacAddress2* values are assumed to have default values of all zero by receiving stations.

10.4.8 Organization-specific ATT

The variable-length organization-specific ATT is identified by its distinctive *type*=ATT_ORG_SPECIFIC value (see 10.3.5). The organization-specific ATT encodes a collection of organization-specific parameters, as illustrated in Figure 10.23.

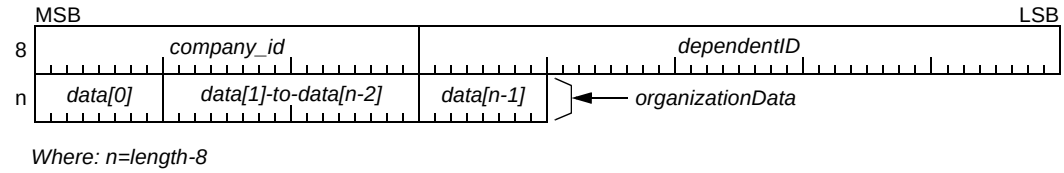


Figure 10.23—Organization-specific *attDataUnit* format

10.4.8.1 *organizationEUI*: An EUI-64 comprised of the *company_id* and *dependentID* fields shown in Figure 10.23. The *company_id* is a 24-bit OUI field supplied by the IEEE/RAC for the purpose of identifying the organization supplying the (unique within the organization) 40-bit *dependentID.*

10.4.8.2 *data[n]*: A sequence of bytes whose format and function is dependent on the initial 64-bit EUI-64 identifier. The exact definition of the function of this organization-specific data is outside of the scope of this standard.

The organization-specific ATT may be included in the ATD frame, allowing this information to be reported to all other stations on the RPR ring. The actions taken as a result of receiving or not receiving the information contained in the organization-specific ATT is outside the scope of this standard.

10.5 Topology database updates

10.5.1 Topology database structure

10.5.1.1 Logical representation of topology database

The topology database is partitioned into multiple components, as shown in Figure 10.24. The different portions of the topology database are referenced as follows:

a) Ring-level information: *ringInfo.fieldname*.
b) Information for the station itself: *myTopoInfo.fieldname*.
c) Information received from stations that are *n* hops away:
   topoEntry[0][hops].fieldname, for information received from ringlet0
   topoEntry[1][hops].fieldname, for information received from ringlet1.

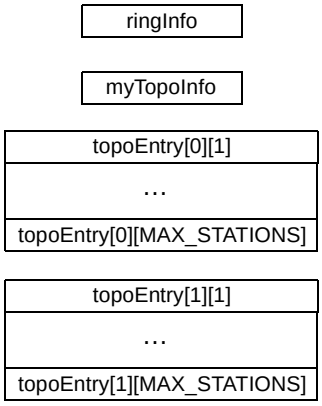


Figure 10.24—Topology database structure

The correspondence between indexing per ringlet and the east/west span nomenclature used in the MIB is described in 10.2.3.

Received TP frames update station MAC address, edge state, and protection information, on the source station's ringlet and hop count distance Received TC frames are used to update topology checksum related information. Received LRTT frames update loop round-trip time measurement information. The remainder of the information in the database is obtained from ATD frames.

The topology database contains ring-level information (see Table 10.10), station-local information (see Table 10.11), and information associated with each of the other stations (see Table 10.12).

Table 10.10—Ring level topology database: *ringInfo*

Description	Variable	Example value
TP derived information		
ring jumbo type	*jumboType*	REGULAR
MTU size	*mtuSize*	REGULAR_MAX
number of stations on ring	*numStations*	4
ring topology type	*topoType*	CLOSED_RING
transmit hops on ringlet0	*totalHopsTx[0]*	3
transmit hops on ringlet1	*totalHopsTx[1]*	3
ATD derived information		
Station receiving frames with bad fcs	*badFcsUser*	FALSE
Station using multichoke fairness on ringlet0	*multichokeUser*[0]	FALSE
Station using multichoke fairness on ringlet1	*multichokeUser*[1]	FALSE
Available bandwidth on ringlet0	*unreservedRate*[0]	800
Available bandwidth on ringlet1	*unreservedRate*[1]	700

Table 10.11—Topology database for local station: *myTopoInfo*

Description	Variable	Example value
TP derived information		
local station MAC address	*macAddress*	00-10-A4-97-B7-DE
station jumbo preference	*jumboPrefer*	REGULAR
station protection configuration	*protConfig*	WRAPPING
west span protection state	*spanProtState*[0]	IDLE
east span protection state	*spanProtState*[1]	IDLE
west span edge state	*spanEdgeState*[0]	FALSE
east span edge state	*spanEdgeState*[1]	FALSE
sequence number	*sequenceNumber*	3
west last known neighbor MAC address	*lastNeighborMac*[0]	00-10-A4-97-A8-DE
east last known neighbor MAC address	*lastNeighborMac*[1]	00-10-A4-97-A8-BD
topology checksum	*checksum*	AA-11-BB-22
topology checksum valid	*checksumValid*	TRUE
west neighbor topology checksum	*neighborChecksum*[0]	AA-11-BB-22
east neighbor topology checksum	*neighborChecksum*[1]	AA-11-BB-22
west neighbor topology checksum valid	*neighborCheckValid*[0]	TRUE
east neighbor topology checksum valid	*neighborCheckValid*[1]	TRUE

Table 10.11—Topology database for local station: *myTopoInfo* *(continued)*

ATD derived information		
station MAC address	*macAddress*	00-10-A4-97-B7-DE
station name	*stationName*	San Francisco
secondary MAC address 1	*secMacAddress1*	None
secondary MAC address 2	*secMacAddress2*	None
secondary MAC address 1 in use	*secMacAddress1Used*	FALSE
secondary MAC address 2 in use	*secMacAddress2Used*	FALSE
weight on ringlet0	*weight*[0]	1
weight on ringlet1	*weight*[1]	5
reserved bandwidth on ringlet0	*reservedRate*[0]	200
reserved bandwidth on ringlet1	*reservedRate*[1]	160
receives frames with bad fcs	*badFcsUser*	FALSE
multichoke fairness frames expected	*multichokeUser*	FALSE
uses conservative mode	*conservativeMode*	TRUE
management address type	*managementAddressType*	IPV4
management IP address	*managementIpAddr*	192.168.1.50
interface index	*interfaceIndex*	2

Table 10.12—Topology database for one ringlet: *topoEntry*[*ringlet*][*hops*]

Variable	hops=1	hops=2	hops=3
TP derived information			
macAddress	00-10-A4-97-A8-DE	00-10-A4-97-A8-EF	00-10-A4-97-A8-BD
valid	TRUE	TRUE	TRUE
hopsRx	1	2	3
jumboPrefer	REGULAR	JUMBO	JUMBO
protConfig	WRAPPING	WRAPPING	WRAPPING
spanProtState[0]	IDLE	IDLE	IDLE
spanProtState[1]	IDLE	IDLE	IDLE
spanEdgeState[0]	FALSE	FALSE	FALSE
spanEdgeState[1]	FALSE	FALSE	FALSE
reachable	TRUE	TRUE	TRUE
sequenceNumber	3	15	5
lrtt	10	20	30
ATT derived information			
macAddress	00-10-A4-97-A8-DE	00-10-A4-97-A8-EF	00-10-A4-97-A8-BD
stationName	Los Angeles	Denver	Portland
secMacAddress1	00-10-A4-97-B7-AB	00-10-A4-97-D4-DE	None
secMacAddress2	00-10-A4-97-B7-CD	00-10-A4-97-B7-EF	None
secMacAddress1Used	TRUE	TRUE	FALSE
secMacAddress2Used	TRUE	TRUE	FALSE
weight[0]	1	3	1
weight[1]	5	3	5
reservedRate[0]	50	100	200
reservedRate[1]	40	80	160
badFcsUser	FALSE	FALSE	FALSE
multichokeUser	FALSE	FALSE	FALSE
conservativeMode	TRUE	FALSE	TRUE
managementAddressType	IPV4	IPV4	IPV4
managementIpAddr	10.0.0.1	10.0.0.2	192.168.1.100
interfaceIndex	12	3	2

10.5.2 Attribute updates

In order to allow other stations time to react to changes in advertised station attributes, stations shall advertise changed values for attributes that use resources (i.e., ATT_STATION_BW, ATT_STATION_NAME, ATT_STATION_SEC_MAC) as follows:

a) Before adding a resource (e.g., more reserved bandwidth or secondary MAC address), advertise the resource usage, then start using it after at least RRTT + 10 ms.

b) After removing a resource usage (e.g., station name), wait at least RRTT + 10 ms before removing advertisement of it.

For attributes for which there is a resource total that should not be exceeded (e.g., reserved bandwidth), there is no control in the MAC to prevent this condition from occurring.

NOTES
1—An implementation can issue an alarm on passing the threshold(s), if it so desires.
2—The means of determining RRTT are implementation specific. Either of the following are known to be sufficient:
a) Transmit an advertisement and wait for that advertisement frame to return to this source station.
b) Transmit a flush and wait for that flush frame to return to this source station.

For attributes for which there are one or two items of each resource value (e.g., station name or secondary MAC address), no station shall claim ownership if another station already claims ownership. Race conditions that allow two stations to advertise the same value shall force both to remove their claim and issue an alarm.

Some rules related to updates of the topology database resulting from ATD frames are as follows:

a) An ATD is ignored if received from a station not in the topology database.

b) An ATD is ignored if received from a station whose topology database entry is not marked as valid.

c) An ATD is ignored if received from a station not at the same location in the topology database as indicated by the *ttl* value of the received ATD frame.

The following additional rules apply to updates of the topology database due to receipt of ATTs other than organization-specific ATTs.

a) The ATD-frame information is retained until replaced by new ATD frame information, or cleared.

b) When a ATD frame is received, stored information contained in this type of ATD frame is replaced.

c) All ATT information received from a station is included in that stations's topology database entry.

The topology database is modified upon receipt of TP, TC, LRTT, or ATD frames that result in a change in information contained in the database, by the protection state machine, or based on input from the protection state machine that results in further updates to the database. Changes in TC, LRTT, or ATD frame parameters, or by the protection state machine, modify fields within the database, but not the apparent position of stations in the topology.

10.5.3 Lower level representation of topology database

The MAC has topology database components illustrated in Figure 10.25. The index associated with this database represents the distance-to-source hops on ringlet0 or on ringlet1, for the entries are derived from TP frames received on ringlet1 and ringlet0, respectively. The index value is derived from the *frame.ttl* field value, as specified in Equation 10.1.

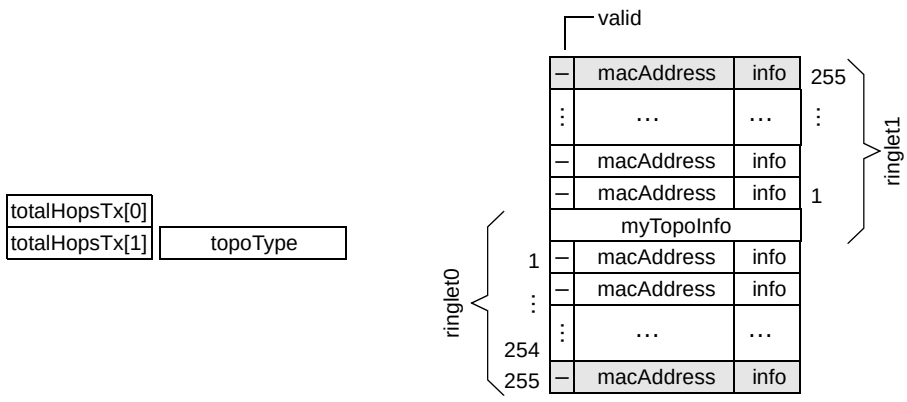


Figure 10.25—Topology database parameters

NOTE—Figure 10.25 is provided strictly for explanatory, illustrative, and representational purposes. It is not intended to depict a specific implementation or to constrain implementers. In particular, the widths of the fields in the figure are not shown to scale.

The *totalHopsTx[0]* and *totalHopsTx[1]* fields indicate the number of destination stations in the transmit direction on ringlet0 and ringlet1. The *topoType* field indicates whether the topology is a closed ring or an open ring.

Within each array, the *valid* field indicates whether a topology database entry is considered valid. The *valid* entry is nominally cleared for all stations downstream of a detected edge; the *valid* bit entry is also set when a TP frame is received from the station across a detected edge in an open ring topology. The *macAddress* field is the source station MAC address within the received TP frame. The *info* field refers to the remainder of the information contained in the topology database.

10.5.4 Topology database updates

10.5.4.1 Stable database structures

Stable database structures can reflect closed ring or open ring topologies, as illustrated in the left and right sides of Figure 10.26, respectively. In a closed ring topology with no more than MAX_STATIONS, the local station's entries are repeated on the two ends of the closed ring. This facilitates verification that the topology is a closed ring. In an open ring topology, each side of the open ring terminates where an edge is detected.

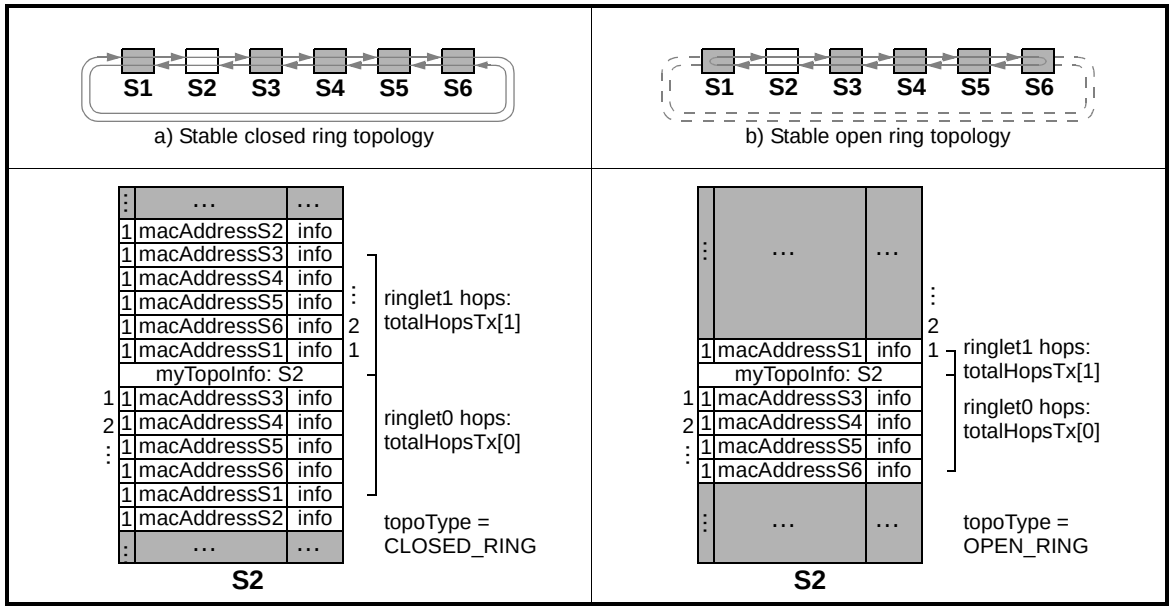


Figure 10.26—Stable database structures

10.5.4.2 Topology change sequences

A sequence of topology changes is illustrated in Figure 10.27. A stable closed ring topology (Figure-Cell 10.27-a) has valid index values and redundant entries. A single severed link (Figure-Cell 10.27-b) restricts the database knowledge to an open ring array, based on a pair of reports from the two edge stations. Two severed links (Figure-Cell 10.27-c) further restricts the database knowledge to a smaller open ring array, based on a pair of reports from the two edge stations. Joining additional stations (Figure-Cell 10.27-d) requires more extensive database updates, with span extensions triggered by the remote edge-station reports.

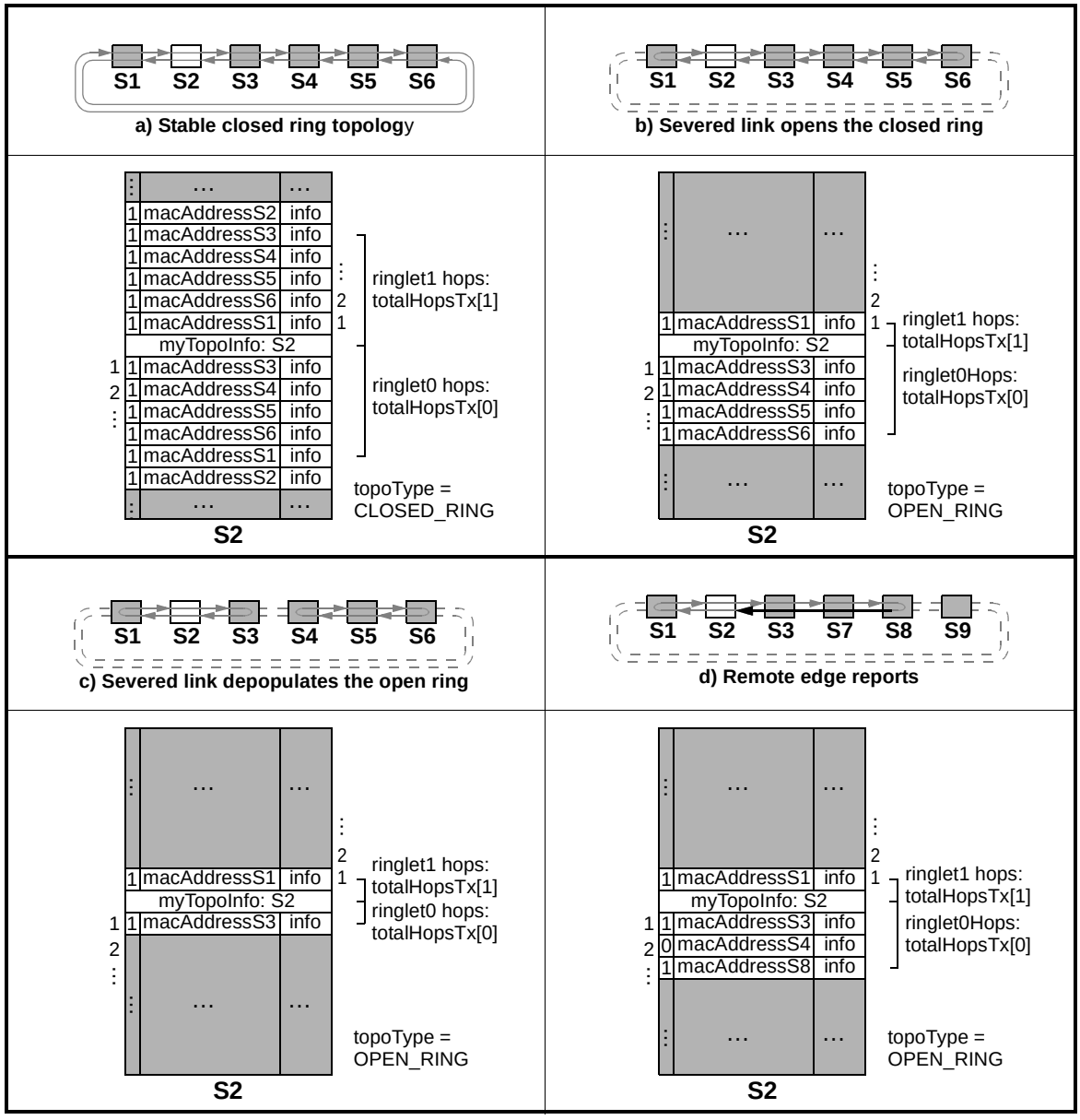


Figure 10.27—Topology change sequences

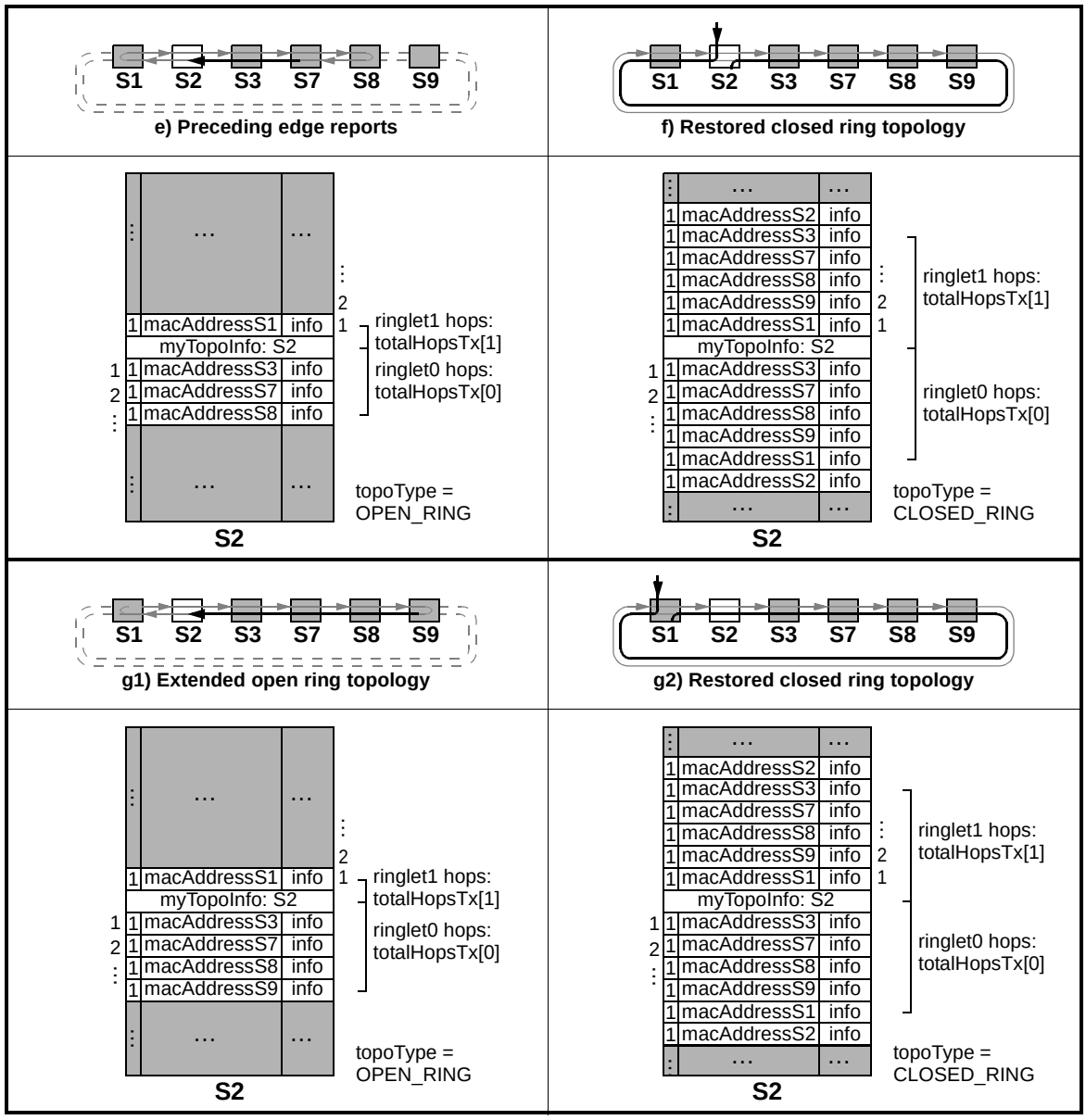


Figure 10.27—Topology change sequences *(continued)*

The remote edge report is followed by TP frames (Figure-Cell 10.27-e) from closer stations, filling in the invalid database entries. The appearance of a connecting station (Figure-Cell 10.27-f) eliminates the possibility of edge-point reports, forcing the report of a self-generated TP frame to confirm the closed ring topology.

Alternatively, the addition of a connecting station can happen to (Figure-Cell 10.27-g1) extend the segment before (Figure-Cell 10.27-g2) converting between open ring and closed ring topologies. This has the advantage that the database entries are implied by the consistent incremental nature of the restoration.

10.6 State machines

10.6.1 State machine functions

A station's topology and protection state machines manage the transmission/reception of related frame, the reception of administrative requests, and the processing associated with these functions, as illustrated in Figure 10.28. Within this figure, he shaded ReceiveTpFrame, ReceiveAtdFrame, and ReceiveMonitor state machines are the only state machines that have no direct access to the shared topology database.

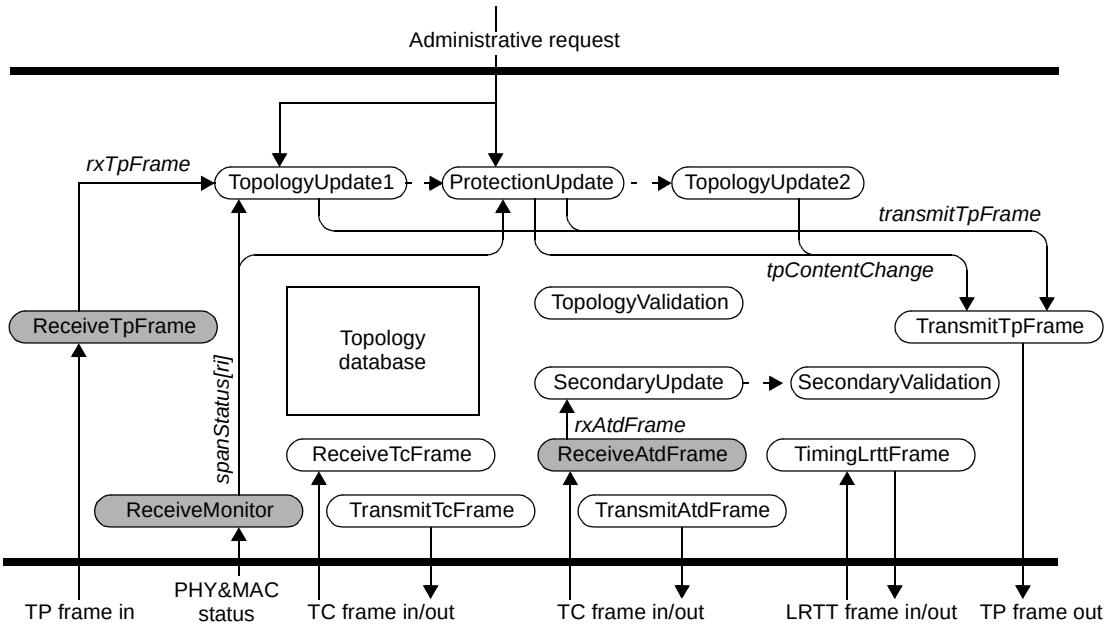


Figure 10.28—Topology and protection relationships

10.6.1.1 ProtectionUpdate: The ProtectionUpdate (see 10.6.5) state machine is called by TopologyUpdate1 and executes on an *ri*-specified span. The protection state machine uses information contained in the topology database to determine its span's protection state and edge status, before updating the topology database. The TopologyUpdate2 state machine is called immediately before ProtectionUpdate completion.

10.6.1.2 ReceiveAtdFrame: The ReceiveAtdFrame (see 10.6.12) state machines receive ATD frames.

10.6.1.3 ReceiveTcFrame: The one-per span TransmitTcFrame (see 10.6.10) state machines receive TC frames.

10.6.1.4 ReceiveTpFrame: Two one-per span ReceiveTpFrame (see 10.6.8) state machines provide *rxTpFrame* (validated TP control frames) for the TopologyUpdate1 state machine.

10.6.1.5 SecondaryUpdate: The SecondaryUpdate (see 10.6.13) state machine updates secondary MAC addresses, based on received ATT frames or administrative directives. SecondaryValidate is then called to detect excess or duplicate secondary MAC address defects.

10.6.1.6 SecondaryValidate: The SecondaryValidate (see 10.6.14) state machine ensures that new secondary MAC addresses do not duplicate station MAC addresses or other secondary MAC addresses, and that no more than a fixed maximum number of secondary MAC addresses are in use on the ring at any time. The SecondaryValidate state machine is called by the SecondaryUpdate state machine.

10.6.1.7 TimingLrttFrame: The TimingLrttFrame (see 10.6.15.1) state machine generates LRTT request frames and processes the returned response frames, for the purpose of calibrating the ring's LRTT.

10.6.1.8 TopologyUpdate1: One TopologyUpdate1 (see 10.6.3) state machine processes TP frames from both spans of the station. Context containment is triggered (as necessary) and the ProtectionUpdate state machine is called when necessary.

10.6.1.9 TopologyUpdate2: The TopologyUpdate2 (see 10.6.3) state machine is called by ProtectionUpdate and executes on an *ri*-specified span. The database is updated (to account for ProtectionUpdate changes) and additional checks are performed based on the ProtectionUpdate-modified topology database, as well as configuration information for jumbo preference and wrap configuration.

10.6.1.10 TopologyValidation: The TopologyValidation (see 10.6.6) state machine determines when the topology is stable and valid.

10.6.1.11 TransmitAtdFrame: The TransmitTcFrame (see 10.6.11) state machine transmits ATD frames.

10.6.1.12 TransmitTcFrame: The TransmitTcFrame (see 10.6.9) state machine transmits TC frames.

10.6.1.13 TransmitTpFrame: The TransmitTpFrame (see 10.6.3) state machine transmits TP frames, based on *transmitTpFrame* and *transmitTpFrame* triggers.

NOTE—This clause defines logical behavior of state tables rather than specifying the number of instances.

10.6.2 ReceiveMonitor state machine

The ReceiveMonitor state machine is responsible for updating the topology database based on receipt of TP frames from the ring or a change in local protection information.

10.6.2.1 ReceiveMonitor state machine variables

currentTime
See 10.2.9.
keepaliveTime[*ri*]
The time at which a keepalive defect is reported (in the absence of continued fairness frames).
keepaliveDelay
If no keepalive is received within this time, a signal fail condition is declared on a span.
Range—[2 ms, 50 ms]
Resolution—1 ms
Default—3 ms
LINK_STATUS
See 10.2.8.
miscablingDefect[*ri*]
See 10.2.9.
noDefectDelay
The time delay after which a persistent error (not NO_DEFECT) condition is allowed.
noDefectTime
The time of the last observed NO_DEFECT receive-link condition.
ri
See 10.2.3.
spanStatus[*ri*]
The link status for the span, as reported to the TopoUpdate1 and ProtectionUpdate state machines.

10.6.2.2 ReceiveMonitor state machine routines

The routines below are used in the topology database update state machines.

TimerDone(*timeout*)
TimerBad(*timeout, delay*)
See 10.2.7.

10.6.2.3 ReceiveMonitor state tables

The ReceiveMonitor state machine generates the *spanStatus*[*ri*] indication based on receive-link status, as specified in Table 10.13. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 10.13—ReceiveMonitor state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	keepAliveEvent	1	keepaliveTime = currentTime + keepaliveDelay; keepAliveEvent = FALSE;	START
	TimerBad(noDefectTime, noDefectDelay);	2	noDefectTime = currentTime + noDefectDelay;	FIRST
	TimerDone(noDefectTime);	3	noDefectTime = currentTime;	
	—	4	—	
FIRST	TimerBad(keepaliveTime, keepaliveDelay);	5	keepaliveTime = currentTime + keepaliveDelay;	NEXT
	TimerDone(keepaliveTime);	6	keepaliveTime = currentTime;	
	—	7	—	
NEXT	TimerDone(keepaliveTime)	8	spanStatus[ri] = SF;	FINAL
	miscablingDefect[ri]	9		
	!TimerDone(noDefectDelay)	10	spanStatus[ri] = IDLE;	FINAL
	LINK_STATUS == NO_DEFECT	11		
	LINK_STATUS == DEGRADE_FAIL	12	spanStatus[ri] = SF;	
	LINK_STATUS == SIGNAL_FAIL	13		
	—	14	spanStatus[ri] = SD;	
FINAL	LINK_STATUS == NO_DEFECT	15	noDefectTime = currentTime + noDefectDelay;	RETN
	—	16	—	

Row 10.13-1: A recorded keepalive event extends the *keepaliveTime* timeout.
(Another state machine sets *keepaliveEvent*=TRUE when validated fairness frames are received).
Row 10.13-2: An incorrectly initialized *keepaliveTime* value is properly initialized.
Row 10.13-3: An expired *keepaliveTime* is adjusted to avoid arithmetic overflows.
Row 10.13-4: Otherwise, no keepalive consistency corrections are necessary.

Row 10.13-5: An incorrectly initialized *noDefectTime* value is properly initialized.
Row 10.13-6: An expired *noDefectTime* timer is set to a just-expired state, to avoid arithmetic overflows.
Row 10.13-7: Otherwise, no defect-delay timer corrections are necessary.

Row 10.13-8: If the keepalive timer has expired, a SF condition is asserted.
Row 10.13-9: If a miscabling condition exists, a SF condition is asserted.
Row 10.13-10: Reporting of SF and SD conditions is delayed for a *noDefectDelay*-specified interval.
Row 10.13-11: During transient signal-degrade/signal-fail conditions, a IDLE condition is reported.
Row 10.13-12: After a persistent signal-degrade&signal-fail conditions, a SF condition is reported.
Row 10.13-13: After a persistent signal-fail link-status condition, a SF condition is reported.
Row 10.13-14: After other persistent signal-degrade conditions, a SD condition is reported.

Row 10.13-15: In the absence of signal-degrade&signal-fail conditions, restart the *noDefectDelay* timer.
Row 10.13-16: Otherwise, no actions are performed.

10.6.3 TopologyDbUpdate state machine

The TopologyDbUpdate state machine is responsible for updating the topology database based on receipt of TP frames from the ring or a change in local protection information.

10.6.3.1 TopologyDbUpdate1 state machine definitions

MAX_STATIONS
See 10.2.2.
SF
STEERING
WRAPPING
See 10.2.2.

10.6.3.2 TopologyDbUpdate1 state machine variables

containmentActive
See 10.2.3.
edgeChange
See 10.2.3.
hops
Index of topology database entry that is updated by the received TP frame.
myTopoInfo
See 10.2.5.
newNeighbor[*ri*]
See 10.2.3.
protMisconfigDefect
See 10.2.9.
ri
See 10.2.3.
ringInfo
See 10.2.4.
rxTpFrame
See 10.2.3.
spanProtStatus[*ri*]
See 10.2.3.
timeSinceFailDegrade
The time that has elapsed since the initial detection of a physical signal fail or signal degrade. If this time exceeds *holdoffTimerValue*, then this signal fail or signal degrade is made visible to the topology database update state machine.
timeSinceKeepalive[*ri*]
See 10.2.3.

topoChanged
See 10.2.3.
topoEntry
See 10.2.6.
transmitTpFrame
See 10.2.3.
wtrTimeout[*ri*]
The time at which a wait to restore (WTR) timeout is set to expire.

10.6.3.3 TopologyDbUpdate1 state machine routines

The routines below are used in the topology database update state machines.

ClearAtdInfo(*ri, hop*)
See 10.2.7.
ContainOnEdgeChange(*ri, frame, protConfig*)
Checks whether context containment needs to be activated for a change in edge state reported in a received TP frame, as defined by Equation 10.7.
TRUE—Context containment needs to be activated.
FALSE—(Otherwise.)

(10.7)

```
Boolean
ContainOnEdgeChange(old, new, protConfig)
{
    Boolean check;

    switch (protConfig)
    {
    case STEERING:
        check = (old.spanEdgeState[0] != new.spanEdgeState[0]);
        check ||= (old.spanEdgeState[1] != new.spanEdgeState[1]);
        break;
    case WRAPPING:
        check = (old.spanEdgeState[0] && !new.spanEdgeState[0]);
        check ||= (old.spanEdgeState[1] && !new.spanEdgeState[1]);
        break;
    }
    return(check);
}
```

FindIndex(*frame*)
JumboAdminRequestCheck( )
See 10.2.7.
new
A transient copy of *topoEntry* data, with speculative changes based on received TP frame contents.
old
A snapshot copy of *topoEntry* data, provided for convenient test-specification purposes.
Other(*ri*)
See 10.2.8.
ProtectionCallNeeded()
Indicates whether the protection state machine should be called.

```
Boolean                                                                    (10.8)
ProtectionCallNeeded(ri)
{
    int status, ringlet, hop;

    if (AdminRequestCheck(ri) != NULL)
        return(TRUE);
    ReceiveMonitor();
    if (spanStatus[ri] != spanProtStatus[ri])
        return(TRUE);
    if (JumboAdminRequestCheck() != NULL)
        return(TRUE);
    if (ProtAdminRequestCheck() != NULL)
        return(TRUE);
    if (wtrTimeout[ri] != NULL && currentTime >= wtrTimeout[ri])
        return(TRUE);
    return(FALSE);
}
```

ProtAdminRequestCheck( )

See 10.2.7.

ProtectionUpdate(*ri*)

Executes the protection state machine running on the span with a receive link on ringlet *ri*.

TopologyChanged(*ri*, *frame*)

Checks for a difference between the edge state and MAC address information contained in the received TP frame and the topology database entry at the index corresponding to that TP frame, as defined by Equation 10.9. This indicates that the topology has changed.

TRUE—Information in TP frame differs from corresponding topology database entry.

FALSE—(Otherwise.)

```
Boolean                                                                    (10.9)
TopologyChanged(old, new)
{
    Boolean check;

    check = (!old.valid);
    check ||= (old.macAddress != old.macAddress);
    check ||= (old.spanEdgeState[0] != new.spanEdgeState[0]);
    check ||= (old.spanEdgeState[1] != new.spanEdgeState[1]);
    return(check);
}
```

10.6.3.4 TopologyDbUpdate1 state tables

The TopologyDbUpdate1 state machine updates the topology database based on receipt of TP frames from the ring. This state machine is also responsible for detecting when ProtectionUpdate should be called. Topology database updates due to changes in local protection status or local administrative requests are handled by the TopologyDbUpdate1 state machine, which is called at the completion of the protection state machine, as specified in Table 10.14.

Table 10.14—TopologyDbUpdate1 state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	—	1	ri = Other(ri);	FIRST

FIRST	rxTpFrame != NULL &&
	rxTpFrame.ri == ri
		2	new = old= topoEntry[ri][hops];
			hops = FindIndex(new);
			rxTpFrame = NULL;
			new.macAddress = rxTpFrame.sa;
			new.protConfig = rxTpFrame.wc;
			new.jumboPrefer = rxTpFrame.jp;
			new.spanProtState[0] = rxTpFrame.psw;
			new.spanProtState[1] = rxTpFrame.pse;
			new.spanEdgeState[0] = rxTpFrame.esw;
			new.spanEdgeState[1] = rxTpFrame.ese;
			new.sequenceNumber = rxTpFrame.seqnum;
				TOPO

	ProtectionCallNeeded(ri)	3	—	FINAL
	—	4	—	START
TOPO	myTopoInfo.spanProtState != SF && TopologyChanged(old, new)	5	topoChanged = TRUE	EDGE
	—	6	—	EXEC1

EDGE	**!**old.valid &&
	!new.spanEdgeState[Other(ri)]
		7	containmentActive = TRUE;
			transmitTpFrame = TRUE;
			ClearAtdInfo(ri, hops);
				NEIGH

	old.macAddress != new.macAddress	8		
	ContainOnEdgeChange(old, new, myTopoInfo.protConfig)	9	containmentActive = TRUE;	
	—	10	—	
NEIGH	hops == 1 && myTopoInfo.spanProtState != SF	11	newNeighbor[ri] = (new.macAddress != myTopoInfo.lastNeighborMac[ri]); myTopoInfo.lastNeighborMac[ri] = new.macAddress	EXEC1
	—	12	—	

EXEC1	myTopoInfo.spanProtState == SF	13	new.valid = FALSE;
			new.reachable = FALSE;
			ClearAtdInfo(ri, hops);
				EXEC2

	hops!= 1 && new.spanEdgeState[Other(ri)]	14		

	hops== 1 &&
	new.spanEdgeState[Other(ri)]
		15	new.valid = TRUE;
			new.reachable = FALSE;
			ClearAtdInfo(ri, hops);	

	new.spanEdgeState[ri]	16	new.valid = TRUE; new.reachable = TRUE;	
	—	17	—	NEAR

Table 10.14—TopologyDbUpdate1 state table *(continued)*

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

EXEC2	—	18	for (j= hops+1; j<=MAX_STATIONS; j+= 1)
			{
			topoEntry[ri][j].valid = FALSE;
			topoEntry[ri][j].reachable = FALSE;
			ClearAtdInfo(ri, hops);
			}
				NEAR

NEAR	—	19	topoEntry[ri][hops]= new;	FINAL
FINAL	—	20	ProtectionUpdate(ri); ProtectionUpdate(Other(ri));	START

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 10.14-1: Process the TP frames from the datapaths in alternating order.

Row 10.14-2: Create a speculative *new* copy by merging the received TP-frame data parameters with the old copy, so that old and new copies are available for the detection of changed values.
Row 10.14-3: If protection conditions have changed, the protection state machines should be executed.
Row 10.14-4: Otherwise, continue checking for following changes.

Row 10.14-5: If information within the received TP frame does not match the corresponding fields of the topology database entry, then set *topologyChanged* to TRUE. Continue with the context containment check.
Row 10.14-6: Otherwise, continue with the topology database update.

Row 10.14-7: If the previous entry was invalid, a new stations was added, and containment is required.
Row 10.14-8: If the previous entry was from a different source, containment is also required.
Row 10.14-9: The change of a station's edge condition sometimes forces a containment.
Row 10.14-10: Otherwise, containment is not invoked.

Row 10.14-11: The detection of a valid changed neighbor sets condition codes for other state machines.
Row 10.14-12: Otherwise, the neighbor-changed condition codes are not set.

Row 10.14-13: Parameters from across a SF span are saved (for diagnostics) but are not marked valid.
Row 10.14-14: Parameters from across a remote edge are saved (for diagnostics) but are not marked valid.
Row 10.14-15: Parameters coming across a failed span are saved (for diagnostics) but are not marked valid.
Row 10.14-16: Otherwise, parameters from across a non-SF span are assumed to be valid and are saved.
Row 10.14-17: Remote parameters from across a SF span are discarded.

Row 10.14-18: The beyond-the-edge entries are marked invalid.
Row 10.14-19: The speculative new topology database entry is saved.
Row 10.14-20: The protection state machines resolve TP-frame invoked topology database changes.

10.6.4 TopologyDbUpdate2 state machine

The TopologyDbUpdate2 state machine sets *edgeChange* to indicate whether the edge state has changed for either span; this information is used to activate context containment. The TopologyDbUpdate2 state machine also handles setting of the jumbo preferences type for the ring (*jumboType*), checks for a protection configuration mismatch on the ring, and updates validity bits in the topology database. The topology database is also updated based on changes of local protection state information (already updated in the database by Table 10.16).

10.6.4.1 TopologyDbUpdate2 state machine definitions

JUMBO
    See 10.2.2.
JUMBO_MAX
    See 10.2.8.
MAX_STATIONS
    See 10.2.2.
REGULAR
    See 10.2.2.
REGULAR_MAX
    See 10.2.8.
SF
STEERING
WRAPPING
    See 10.2.2.

10.6.4.2 TopologyDbUpdate2 state machine variables

containmentActive
    See 10.2.3.
edgeChange
    See 10.2.3.
hops
    Index of topology database entry that is updated by the received TP frame.
myTopoInfo
    See 10.2.5.
ri
    See 10.2.3.
ringInfo
    See 10.2.4.
spanProtStatus[*ri*]
    See 10.2.3.
topoChanged
    See 10.2.3.
topoEntry
    See 10.2.6.
tpContentChange
    See 10.2.3.

10.6.4.3 TopologyDbUpdate2 state machine routines

The routines below are used in the topology database update state machines.

ClearAtdInfo(*ri*, *hop*)

See 10.2.7.

JumboAdminRequest( )

See 10.2.7.

MismatchedProtection()

See 10.2.7.

Other(*ri*)

See 10.2.8.

ProtAdminRequest()

Provides the value of a new administrative protection configuration request.

STEERING—Indicates a station that is configured to use steering protection.

WRAPPING—Indicates a station that is configured to use wrapping protection.

(null)—No administrative protection configuration request is available.

RegularStationExists()

Determines if there is a regular (non-jumbo preferred) station on the ring, as defined by Equation 10.9.

TRUE—There is a regular (non-jumbo preferred) station on the ring.

FALSE—(Otherwise.)

(10.10)

```
Boolean
RegularStationExists()
{
    int ringlet, hop;

    if (myTopoInfo.jumboPrefer == REGULAR)
        return(TRUE);
    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hop = 1; hop <= MAX_STATIONS; hop += 1)
       {
            if (!topoEntry[ringlet][hop].valid)
                continue;
            if (topoEntry[ringlet][hop].jumboPrefer == REGULAR)
                return(TRUE);
       }
    return(FALSE);
}
```

10.6.4.4 TopologyDbUpdate2 state table

The TopologyDbUpdate2 state machine is called from the ProtectionUpdate state machine, after that state machine has updated span protection state and edge state for the station, as specified in Table 10.15.

Table 10.15—TopologyDbUpdate2 state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	(jumbo = JumboAdminRequest()) != NULL	1	tpContentChange ||= (jumbo != myTopoInfo.jumboPrefer); myTopoInfo.jumboPrefer= jumbo;	PROT
	—	2	—	
PROT	(protect= ProtAdminRequest()) != NULL	3	tpContentChange ||= (protect != myTopoInfo.protConfig); myTopoInfo.protConfig= protect;	EDGE
	—	4	—	
EDGE	myTopoInfo.protConfig == STEERING && edgeChange	5	containmentActive = TRUE;	TEST
	myTopoInfo.protConfig == WRAPPING && !myTopoInfo.spanEdgeState[ri] && edgeChange	6		
	—	7	—	
TEST	!myTopoInfo.spanEdgeState[ri]	8	—	SIZE
	myTopoInfo.spanProtState[ri] == SF	9	topoEntry[ri][j].valid = FALSE;	INVAL
	—	10	—	

INVAL	—	11	topoEntry[ri][1].reachable = FALSE;
			for (j=2; j <= MAX_STATIONS; j+= 1) {
			topoEntry[ri][j].valid = FALSE;
			topoEntry[ri][j].reachable =FALSE;
			ClearAtdInfo(ri, j);
			}
				SIZE

SIZE	RegularStationExists()	12	ringInfo.jumboType = REGULAR; ringInfo.mtuSize = REGULAR_MAX;	PROT
	—	13	ringInfo.jumboType = JUMBO; ringInfo.mtuSize = JUMBO_MAX;	
PROT	MismatchedProtection()	14	protMisconfigDefect = TRUE;	RETN
	—	15	—	

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4. Upon reaching the RETN state, execution of this state machine stops until it is next called by the protection state machine.

Row 10.15-1: The jumbo setting for the station is updated; changes (if any) are noted.
Row 10.15-2: Otherwise do nothing.

Row 10.15-3: The protection setting for the station is updated; changes (if any) are noted.
Row 10.15-4: Otherwise do nothing.

Row 10.15-5: If a steering-protected station's edge state has changed, context containment is activated.
Row 10.15-6: If a wrapping-protected station's edge state has cleared, context containment is activated.
Row 10.15-7: Otherwise do not activate context containment due to edge changes at the station.
(The *containmentActive* variable can also be set by the first stage of the topology database update.)

Row 10.15-8: If no edge is present, edge processing is skipped.
Row 10.15-9: Update validity and reachability fields in the topology database for an edge on the ringlet0 (west) side of the station. Clear ATD information in the topology database associated with this entry.
Row 10.15-10: Update validity and reachability fields in the topology database for an edge on the ringlet1 (east) side of the station. The validity is not cleared for a neighboring station directly on the other side of an edge span if the protection state of the span is not SF, but that station is not reachable.
Row 10.15-11: Clear the cross-edge reachability indication, and entry information associated with beyond the neighbor entries.

Row 10.15-12: If a regular station is present on the ring, the ring jumbo preference type is set to non-jumbo. This setting is done immediately upon detection of the station, not after topology validation has completed.
Row 10.15-13: If all stations present on the ring are jumbo-preferred, then the ring jumbo preference type is set to jumbo

Row 10.15-14: If there is are wrap configured and steering configured stations on the same ring, the protection configuration defect is set to TRUE. The clearing of this defect is done only when topology validation has completed.
Row 10.15-15: Otherwise do nothing.

10.6.5 ProtectionUpdate state machine

This subclause specifies the unified state machine for both steering and wrapping protection. The protection state machine runs on the span (east and west) of a station for which it is called.

10.6.5.1 ProtectionUpdate state machine definitions

FS
IDLE
MAX_STATIONS
MS
SD
SF
WTR
See 10.2.2.

10.6.5.2 ProtectionUpdate state machine variables

adminMod
: An input trigger to the state machine representing a modified administrative state of this span.
  TRUE—The input trigger to the state machine is a modified administrative state of this span.
  FALSE—(Otherwise.)

currentTime
: See 10.2.9.

edgeChange
: See 10.2.3.

myTopoInfo
: See 10.2.5.

newNeighbor[*ri*]
: See 10.2.3.

ri
: See 10.2.3.

saved
: A variable that saves the edge state and protection state information of the local station at entry to the protection state machine. This variable has subfields as categorized below.
  edge—Edge state on the span of the station with receive link on ringlet *ri*.
  prot—Protection state on the span of the station with receive link on ringlet *ri*.

spanProtAdmin[*ri*]
: See 10.2.3.

spanProtStatus[*ri*]
: See 10.2.3.

statusMod
: The input trigger to the state machine representing the operational status of this span.
  TRUE—The input trigger to the state machine is a modified operational status of this span.
  FALSE—(Otherwise.)

topoEntry
: See 10.2.6.

tpContentChange
: See 10.2.3.

transmitTpFrame
: See 10.2.3.

10.6.5.3 ProtectionUpdate state machine routines

The routines below are used in the protection state machine or are called within routines in this subclause.

CheckIdleEdges(*ri*)
: Determines if the ring has idle edges on both ends, as defined by Equation 10.11. A station's entry in the topology database must be valid for that station's protection state and edge values to be considered.
  TRUE—There are idle edges on both ends of the ring.
  FALSE—(Otherwise.)

(10.11)

```
Boolean
CheckIdleEdges(ri)
{
    int ringlet, hop;
    Boolean idleEdge;

// Check in topology database containing TP frame information
//  from both ringlets.
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        idleEdge = (myTopoInfo.spanEdgeState[ringlet] &&
          myTopoInfo.spanProtState[ringlet] == NULL);
        for (hop = 1; idleEdge == FALSE && hop <= MAX_STATIONS; hop += 1)
        {
            if (!topoEntry[ringlet][hop].valid)
                continue;
            idleEdge ||= (topoEntry[ringlet][hop].spanEdgeState[ringlet] &&
              topoEntry[ringlet][hop].spanProtState[ringlet] == IDLE);
        }
        if (idleEdge == FALSE)
           return(FALSE);
    }
    return(TRUE);
}
```

CheckPreempt(state, ri)

Determines whether there is any other span in the ring (other than that connecting the station and its neighbor, where the station receives from the span on ringlet *ri*) that contains the protection state of *protState* or a protection state higher than *protState* in the hierarchy, as defined by Equation 10.12. A station's entry in the topology database must be valid for that station's protection state values to be considered.

TRUE—There is another span in the ring with the aforementioned properties.

FALSE—(Otherwise.)

(10.12)

```
Boolean
CheckPreempt(state, ri)
{
    int ringlet, hop;

    if (myTopoInfo.spanProtState[Other(ri)] >= state)
        return(TRUE);

// Check in topology database containing TP frame information
//  from both ringlets.
    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hop = 1; hop <= MAX_STATIONS; hop += 1)
        {
            if (!topoEntry[ringlet][hop].valid)
                continue;
            if ((hop == 1) &&
              topoEntry[ringlet][hop].spanProtState[ri] >= state)
                return(TRUE);
            if (hop > 1 &&
             (topoEntry[ringlet][hop].spanProtState[Other(ri)] >= state ||
            topoEntry[ringlet][hop].spanProtState[ri] >= state))
                return(TRUE);
        }
    return(FALSE);
}
```

DistantPreempt(state, ri)

Determines whether the locally visible protection state of the receive link of this span is preempted by protection elsewhere on the ring, not including this span, as defined by Equation 10.13.

(The routines *CheckPreempt* and *CheckIdleEdges* are also defined in this subclause.)

TRUE—The protection state of this span is preempted.

FALSE—(Otherwise.)

```
Boolean                                                          (10.13)
DistantPreempt(state, ri)
{
    Boolean check, checkPreemption, checkIdleEdges;

    checkPreemption= CheckPreempt(state, ri);
    checkIdleEdges= CheckIdleEdges(ri);
    check = (state < SF) && (checkPreempt || checkIdleEdges);
    return(check);
}
```

EdgeChanged(ri, saved)

Compares *saved.edge* to the edge state contained in the topology database of the span that the station receives from on ringlet *ri*, as defined by Equation 10.14.

TRUE—The edge state contained in the topology database is different from *saved.edge*.

FALSE—(Otherwise.)

```
(saved.edge != myTopoInfo.spanEdgeState[ri]);                    (10.14)
```

GetAdminRequest(*ri*)

Provides the value of a new administrative protection command on the span receiving on ringlet *ri*.

FS—Indicates a forced switch command.

MS—Indicates a manual switch command.

IDLE—Indicates the clearing of a forced switch or manual switch command.

IndicateEdgeChangeAndValue( )

Indicates to the LME that edge state has changed for this station, and the value that the edge state has changed to. The LME can use this information to compute protection duration, last activation time of protection, and number of times protection has been initiated on that span.

NeighborOverride(state, ri)

Determines whether the locally visible protection state of the receive link of this span is overridden by the protection state of the other link of this span, as defined by Equation 10.15.

TRUE—The protection state is overridden.

FALSE—(Otherwise.)

```
(state < SF && topoEntry[ri][1].valid &&
  state < topoEntry[ri][1].spanProtState[Other(ri)])             (10.15)
```

Other(*ri*)

See 10.2.8.

ProtChanged(ri, saved)

Compares *saved.prot* to the protection state contained in the topology database of the span that the station receives from on ringlet *ri*, as defined by Equation 10.16.

TRUE—The protection state contained in the topology database is different from *saved.prot*.

FALSE—(Otherwise.)

```
(saved.prot != myTopoInfo.spanProtState[ri]);                    (10.16)
```

TimerDone(*timeout*)

TopoStateMachineStage2(*ri*)

See 10.2.7

10.6.5.4 ProtectionUpdate state table

The ProtectionUpdate state table, as specified in Table 10.16.

Table 10.16—ProtectionUpdate state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	—	1	saved.edge=myTopoInfo.spanEdgeState[ri]; saved.prot= myTopoInfo.spanProtState[ri];	FIRST
FIRST	(admin = GetAdminRequest(ri)) != NULL	2	—	PARSE
	spanProtAdmin[ri] == MS && NeighborOverride(MS, ri)	3	spanProtAdmin[ri] == IDLE;	FIRST
	spanProtAdmin[ri] == MS && DistantPreempt(MS, ri)	4		
	spanProtStatus[ri] == IDLE && wtrTimeout[ri] != NULL	5	spanProtStatus[ri] = WTR	MAIN
	—	6	otherProtState= topoEntry[ri][1].spanProtState[Other(ri)] ReceiveMonitor(); statusMod= (spanStatus[ri] != spanProtStatus[ri]); spanProtStatus[ri] = statusMod;	
PARSE	admin == FS	7	spanProtAdmin[ri] = admin;	FIRST
	admin == IDLE	8		
	NeighborOverride(MS, ri)	9	—	
	DistantPreempt(MS, ri)	10		
	—	11	spanProtAdmin[ri] = MS;	

Table 10.16—ProtectionUpdate state table

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

MAIN	spanProtAdmin[ri] == FS	12	myTopoInfo.spanProtState[ri] = FS;
			myTopoInfo.spanEdgeState[ri] = TRUE;
			wtrTimeout[ri] = currentTime;
				POST

	spanProtStatus[ri] == SF	13	topoEntry[ri][1].valid = FALSE;
			myTopoInfo.spanProtState[ri] = SF;
			myTopoInfo.spanEdgeState[ri] = TRUE;
			wtrTimeout[ri] = currentTime + wtrDelay;	

	spanProtStatus[ri] == SD &&
	NeighborOverride(SD, ri)
		14	myTopoInfo.spanProtState[ri] = SD;
			myTopoInfo.spanEdgeState[ri] = TRUE;
			wtrTimeout[ri] = currentTime;	

	spanProtStatus[ri] == SD &&
	DistantPreempt(SD, ri)
		15	myTopoInfo.spanProtState[ri] = SD;
			myTopoInfo.spanEdgeState[ri] = FALSE;
			wtrTimeout[ri] = currentTime;	

	spanProtStatus[ri] == SD	16	myTopoInfo.spanProtState[ri] = SD;
			myTopoInfo.spanEdgeState[ri] = TRUE;
			wtrTimeout[ri] = currentTime + wtrDelay;	

	spanProtAdmin[ri] == MS	17	myTopoInfo.spanProtState[ri] = MS;
			myTopoInfo.spanEdgeState[ri] = TRUE;
			wtrTimeout[ri] = currentTime;
				POST

MAIN	myTopoInfo.spanProtState[ri] == WTR	18	—	WTRS
	—	19	—	IDLED
WTRS	NeighborOverride(WTR, ri)	20	wtrTime = NULL;	IDLED
	DistantPreempt(WTR, ri)	21		
	newNeighbor[ri]	22		
	wtrTimeout[ri] == NULL	23		
	TimerDone(wtrTimeout[ri])	24		
	—	25	myTopoInfo.spanProtState[ri] = WTR; myTopoInfo.spanEdgeState[ri] = TRUE;	POST
IDLED	topoEntry[ri][1].valid == TRUE && otherProtState != IDLE && !DistantPreempt(otherProtState, ri)	26	myTopoInfo.spanProtState[ri] = IDLE; myTopoInfo.spanEdgeState[ri] = TRUE;	POST
	—	27	myTopoInfo.spanProtState[ri] = IDLE; myTopoInfo.spanEdgeState[ri] = FALSE;	

POST	—	28	edgeChange = EdgeChanged(ri, saved);
			transmitTpFrame ||= ProtChanged(ri,saved);
			tpContentChange ||= transmitTpFrame;
			newNeighbor[ri] = FALSE;
				NEAR

NEAR	edgeChange	29	IndicateEdgeChangeAndValue();
			transmitTpFrame = TRUE;
			tpContentChange = TRUE;
			topoChanged = TRUE;
				FINAL

	—	30	—	

Table 10.16—ProtectionUpdate state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
FINAL	—	31	TopoStateMachineStage2(ri); edgeChange = FALSE;	RETN

Row 10.16-1: Save the edge and protection state on the span of the station with receive link on ringlet *ri*.

Row 10.16-2: Check for queued administrative requests.
Row 10.16-3: An accepted MS administrative request is overridden by the protection state of the neighbor.
Row 10.16-4: An accepted MS administrative request is preempted by the protection state on another span.
Row 10.16-5: A changed-to-idle span sometimes waits for the wait-to-restore (WTR) timer.
Row 10.16-6: Otherwise, other protection state is checked.

Row 10.16-7: A FS administrative request is retained.
Row 10.16-8: An IDLE administrative request is retained.
Row 10.16-9: A pending MS administrative request is overridden by the protection state of the neighbor.
Row 10.16-10: A pending MS administrative request is preempted by the protection state on another span.
Row 10.16-11: A pending MS administrative request is otherwise accepted.

Row 10.16-12: An FS administrative request forces an edge and disables the WTR.
Row 10.16-13: An SF span protection state forces an edge, clears the neighbor's validity bit in this station's database, and enables WTR.
Row 10.16-14: An SD span protection state with neighbor override keeps an edge and disables the WTR.
Row 10.16-15: An SD span protection state with a preemptive other-span protection condition removes the edge and disables the WTR timer.
Row 10.16-16: Otherwise, a SD span protection state forces an edge and the WTR timer is enabled.
Row 10.16-17: An FS administrative request forces an edge and disables the WTR.
Row 10.16-18: The WTR protection states are processed separately.
Row 10.16-19: The IDLE protection states are processed separately.

Row 10.16-20: A neighbor override terminates the WTR timer, changing to the IDLE protection state.
Row 10.16-21: An other-span preemptive protection condition terminates the WTR timer, changing to the IDLE protection state.
Row 10.16-22: A new neighbor terminates the WTR timer, changing to the IDLE protection state.
Row 10.16-23: An inactive timeout terminates the WTR timer, changing to the IDLE protection state. (This is never believed to occur, but is provided to simplify validation of state-machine correctness.)
Row 10.16-24: A timeout condition terminates the WTR timer, changing to the IDLE protection state.
Row 10.16-25: Otherwise, the WTR condition remains and the edge condition is forced.

Row 10.16-26: An IDLE protection state retains an edge in the presence of a not-preempted neighbor.
Row 10.16-27: Otherwise, the IDLE protection state release the edge.

Row 10.16-28: The *edgeChange* value is set for use as an input to Table 10.15. *transmitTpFrame* (an input to 10.18) is set so that fast TP frame transmission is triggered as per the rules in 10.6.7. In this state machine, *tpContentChange* (also an input to 10.18) is set to *transmitTpFrame* if *transmitTpFrame* has been set to TRUE. Variables are also cleared for the next pass through this state machine. Proceed to FINAL state.
Row 10.16-29: An edge change and the new value of edge state is indicated to the LME so that protection-related statistics can be computed; *topoChanged* (an input to Table 10.17) is also set to TRUE.
Row 10.16-30: Otherwise, no action is performed.

Row 10.16-31: The second stage of the topology database update (see Table 10.15) is invoked.

10.6.6 TopologyValidation state machine

This subclause specifies the state machine for validating the topology database. At startup this state machine most likely begins with topology in an unstable state. If the topology stabilizes, the topology moves into the stable state. If all validation checks then pass, the topology moves into the valid state. If any validation check fails, the topology moves into the invalid state and then directly into the unstable state so that the validation process can be repeated.

10.6.6.1 TopologyValidation state machine definitions

CLOSED_RING
OPEN_RING
See 10.2.2.
STABLE
SHAKY
See 10.2.2.

10.6.6.2 TopologyValidation state machine variables

checksumMatch[*ri*]
See 10.2.3.
containmentActive
See 10.2.3.
currentTime
See 10.2.9.
instabilityDelay
The time duration after which the instability defect timer expires.
Value—10 seconds
instabilityTimeout
The time at which the instability defect timer is due to expire.
lrttRequest
See 10.2.3.
maxStationsDefect
See 10.2.9.
myTopoInfo
See 10.2.5.
needSecMacValidation
See 10.2.3.
protMisconfigDefect
See 10.2.9.
ri
See 10.2.3.
ringInfo
See 10.2.4.
stabilityDelay
The time duration after which the instability defect timer expires.
Range—[10 ms, 100 ms]
Default—40 ms.
stabilityTimeout
The time at which the topology stability timer is due to expire.
tempHopsRx[*ri*]
See 10.2.3.

topoChanged
See 10.2.3.
topoEntry
See 10.2.6.
topoInconsistencyDefect
See 10.2.9.
topoInstabilityDefect
See 10.2.9.
topoStability
topoType
topoValid
See 10.2.3.
totalHopsRx[*ri*]
See 10.2.3.
transmitTcFrame
See 10.2.3.

10.6.6.3 TopologyValidation state machine routines

ChecksumMatch( )
Compares the checksums received from the valid (not across edge-span) neighbor station(s), as defined by Equation 10.17. If the checksums are valid and match the valid checksum at this station (indicating that the topology is both valid and stable), then context containment can be cleared.
TRUE—There are checksum matches with each of the valid neighbor station(s)
FALSE—(Otherwise.)

(10.17)

```
Boolean
NeighborChecksumMatch()
{
    int ringlet;
    Boolean neighborMatch;

    for (ringlet = 0; ringlet < 2 && myTopoInfo.checksumValid; ringlet += 1)
        if (myTopoInfo.neighborCheckValid[ringlet])
            checksumMatch[ringlet] =
              (myTopoInfo.checksum == myTopoInfo.neighborChecksum[ringlet]);
        else
            checksumMatch[ringlet] = FALSE;
    neighborMatch = (checksumMatch[0] || myTopoInfo.spanEdgeState[0]);
    neighborMatch &&= (checksumMatch[1] || myTopoInfo.spanEdgeState[1]);
    if (myTopoInfo.checksumValid)
        return(neighborMatch);
    checksumMatch[0] = FALSE;
    checksumMatch[1] = FALSE;
    return(FALSE);
}
```

ChecksumStep(*checksum, macAddress, sequenceNumber*)
Performs the addition to the topology checksum of the value for a single topology database entry, as defined by Equation 10.18. All values in this routine are viewed as big Endian. Appropriate translations need to be made for little Endian implementations.

```
uint32_t                                                                    (10.18)
ChecksumStep(checksum, macAddress, sequenceNumber)
{
    uint64_t sum;

    sum = checksum & 0x00000000FFFFFFFF;
    sum += (macAddress >> 16);
    sum += ((macAddress & 0xFFFF) << 16) | sequenceNumber;
    sum = (sum & 0x00000000FFFFFFFF) + ((sum & 0xFFFFFFFF00000000) >> 32);
    sum = (sum & 0x00000000FFFFFFFF) + ((sum & 0xFFFFFFFF00000000) >> 32);
    return(sum);
}
```

ClearInstabilityDefectTimer( )

Clears the instability defect timer, and sets *instabilityTimer* to zero.

ClearStabilityTimer( )

Clears the stability defect timer, and sets *stabilityTimer* to zero.

ComputeTc( )

Computes the topology checksum of the local station and sets the value into the topology database, as defined by Equation 10.19.

```
void                                                                        (10.19)
ComputeTc()
{
    int ringlet, numHops;
    int64_t checksum = 0x0000000000000000;

    switch(ringInfo.topoType)
    {
    case CLOSED_RING:
        for (ringlet = 0; ringlet < 2; ringlet += 1)
        {
           checksum = ChecksumStep(checksum, myTopoInfo.macAddress,
             myTopoInfo.sequenceNumber);
            for (numHops = 1; numHops < totalHopsRx[ringlet] + 1; numHops += 1)
            {
                if (!topoEntry[ringlet][numHops].valid)
                    continue
                checksum = ChecksumStep(checksum,
                 topoEntry[ringlet][numHops].macAddress,
                   topoEntry[ringlet][numHops].sequenceNumber);
           }
        }
        break;
    case OPEN_RING:
        checksum = ChecksumStep(checksum, myTopoInfo.macAddress,
           myTopoInfo.sequenceNumber);
        for (ringlet = 0; ringlet < 2; ringlet += 1)
            for (numHops = 1; numHops < totalHopsRx[ringlet] + 1; numHops += 1)
            {
                if (!topoEntry[ringlet][numHops].valid)
                    continue
                checksum = ChecksumStep(checksum,
                  topoEntry[ringlet][numHops].macAddress,
                  topoEntry[ringlet][numHops].sequenceNumber);
            }
        break;
    }
    return(FALSE);
}
```

ExceedsMaxStations()

Checks if the number of stations on the ring exceeds MAX_STATIONS, as defined by Equation 10.20.

TRUE—The number of stations on the ring exceeds MAX_STATIONS.

FALSE—(Otherwise.)

(10.20)

```
Boolean
ExceedsMaxStations()
{
    switch(topoType)
    {
    case CLOSED_RING:
        if (tempHopsRx[0] >= MAX_STATIONS || tempHopsRx[1] >= MAX_STATIONS)
            return(TRUE);
        break;
    case OPEN_RING:
        if ((tempHopsRx[0] + tempHopsRx[1] + 1) > MAX_STATIONS)
            return(TRUE);
        break;
    }
    return(FALSE);
}
```

InconsistencyCheck()

Checks for topology inconsistency, as defined by Equation 10.21.

TRUE—Topology found to be inconsistent.

FALSE—(Otherwise.)

(10.21)

```
Boolean
InconsistencyCheck()
{
    int hop, ringlet;

    switch(topoType)
    {
    case CLOSED_RING:
        if(tempHopsRx[0] != tempHopsRx[1])
            return(TRUE);
        for (hop = 1; hop <= tempHopsRx[0]; hop += 1)
            if (topoEntry[0][hop].macAddress !=
             topoEntry[1][tempHopsRx[1]+1-i].macAddress)
                return(TRUE);
        break;
    case OPEN_RING:
        for (ringlet = 0; ringlet < 2; ringlet += 1)
        {
            if (myTopoInfo.spanEdgeState[ringlet])
                continue
            for (hop = 1; hop <= MAX_STATIONS; hop += 1)
           {
                if (!topoEntry[ringlet][hop].valid)
                    continue
                if (topoEntry[ringlet][hop].spanEdgeState[ringlet])
                    break;
                if (myTopoInfo.macAddress == topoEntry[ringlet][hop].macAddress)
                    return(TRUE);
            }
        }
    }
    return(FALSE);
}
```

MarkReachableOrInvalidEntries()

Sets the reachability fields in the topology database after the database is determined to be stable and valid, as defined by Equation 10.22. Also clears validity bits for all stations beyond an edge in an open ring, or beyond the scope of the topology in a closed ring.

(10.22)

```
void
MarkReachableOrInvalidEntries()
{
    int ringlet, hop;

    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hop = 1; hop <= MAX_STATIONS; hop += 1)
        {
            if (hop > totalHopsRx[ringlet] + 1)
            {
                topoEntry[ringlet][hop].reachable = FALSE;
                topoEntry[ringlet][hop].valid = FALSE;
                continue
            }
            if (hop == totalHopsRx[ringlet] + 1)
            {
                topoEntry[ringlet][hop].reachable =
                  (ringInfo.topoType == CLOSED_RING)
                continue
            }
            topoEntry[r][j].reachable = TRUE;
        }
}
```

MismatchedProtection( )

See 10.2.7.

Other(*ri*)

See 10.2.8.

TimerDone(*timeout*)

TimerBad(*timeout*, *delay*)

See 10.2.7.

ValidityCheck()

Walks through the topology to check if there is an invalid entry where a valid entry would be expected, as defined by Equation 10.23. In addition, the hop count for each ringlet and the ring topology type is determined.

TRUE—An invalid entry is detected where a valid entry would be expected.

FALSE—(Otherwise.)

(10.23)

```
Boolean
ValidityCheck()
{
    int ringlet, hop;
    Boolean validityChk = FALSE;
    int topoType = CLOSED_RING;

    tempHopsRx[0] = 0;
    tempHopsRx[1] = 0;
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        if (myTopoInfo.spanEdgeState[ringlet])
        {
            topoType = OPEN_RING;
            continue;
        }
        for (hop = 1; hop <= MAX_STATIONS; hop += 1)
        {
            if (!topoEntry[ringlet][hop].valid)
            {
                 validityChk = TRUE;
                 topoType = OPEN_RING;
                 continue;
            }
            if (topoEntry[ringlet][hop].spanEdgeState[ringlet])
            {
                 topoType = OPEN_RING;
                 tempHopsRx[ringlet] = hop;
                 break;
            }
            if (myTopoInfo.macAddress == topoEntry[ringlet][hop].macAddress)
            {
                tempHopsRx[ringlet] = hop - 1;
                break;
            }
            if (hop == MAX_STATIONS)
                topoType = OPEN_RING;
            tempHopsRx[ringlet] = hop;
       }
    }
    return(validityChk);
}
```

10.6.6.4 TopologyValidation state table

The topology validation state machine runs in parallel with the topology database update and protection state machines, as specified in Table 10.17. The input variable *topoChanged* (from the topology database update state machine) is set to TRUE if a topology stability-impacting change has occurred (in source MAC address or edge state) for any topology database entry due to a topology database update. The topology validation state machine determine whether any topology stability-impacting changes have occurred during the period of the stability timer.

Table 10.17—TopologyValidation state table

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

START	topoChanged	1	topoStability = SHAKY;
			lrttRequest = FALSE;
			myTopoInfo.neighborCheckValid[0]=FALSE;
			myTopoInfo.neighborCheckValid[1]=FALSE;
			transmitTcFrame = TRUE;
				SHAKY

	—	2	—	START
FIRST	—	3	instabilityTimeout= currentTime+defectDelay;	AFTER
AFTER	—	4	stabilityTimeout = currentTime + stableDelay; topoChanged = FALSE;	SHAKY
SHAKY	TimerDone(stabilityTimeout) && !topoChanged	5	topoStability == STABLE; lrttRequest = TRUE;	STABLE
	TimerDone(instabilityTimeout)	6	topoInstabilityDefect = TRUE;	FIRST
	—	7	—	AFTER
STABLE	topoChanged	8	topoStability == SHAKY; lrttRequest = FALSE;	FIRST
	(topoInstabilityDefect = ValidityCheck()) == TRUE	9	—	INVAL
	(maxStationsDefect = ExceedsMaxStations())==TRUE	10		
	(topoInconsistencyDefect = InconsistencyCheck())==TRUE	11		

	—	12	protMisconfigDefect &&=
			MismatchedProtection();
			ringInfo.topoType = topoType;
			totalHopsRx[0] = tempHopsRx[0];
			totalHopsRx[1] = tempHopsRx[1];
			ringInfo.totalHopsTx[0] = totalHopsRx[1];
			ringInfo.totalHopsTx[1] = totalHopsRx[0];
				TYPES

TYPES	topoType == CLOSED_RING	13	ringInfo.numStations = totalHopsRx[0]+1;	DBASE
	—	14	ringInfo.numStations = totalHopsRx[0] + totalHopsRx[1] +1;	

Table 10.17—TopologyValidation state table *(continued)*

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

DBASE	—	15	MarkReachableOrInvalidEntries();
			ComputeTc( );
			myTopoInfo.neighborCheckValid[0]=FALSE;
			myTopoInfo.neighborCheckValid[1]=FALSE;
			myTopoInfo.checksumValid = TRUE;
			transmitTcFrame = TRUE;
			topoValid = TRUE;
			needSecMacValidation = TRUE;
				VALID

INVAL	topoChanged	16	topoStability = SHAKY;	FIRST
	—	17	—	INVAL

VALID	topoChanged	18	topoStability = SHAKY;
			myTopoInfo.checksumValid = FALSE;
			topoValid = FALSE;
				FIRST

	ChecksumMatch()	19	containmentActive = FALSE;	VALID
	—	20	—	

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 10.17-1: Upon startup, if the topology is unstable proceed directly to the SHAKY state. Set *lrttRequest* to FALSE so Table 10.26 is not entered. Set *transmitTcFrame* so that TC frame transmission is triggered in Table 10.20. Initialize the status of the topology checksums received from neighbor stations to FALSE.
Row 10.17-2: Otherwise remain in the START state.

Row 10.17-3: Restart the instability timer before continuing with other checks.
Row 10.17-4: Restart the stability timer before continuing with other checks.

Row 10.17-5: The topology is stable if the stability timer has expired and no topology stability-impacting change has occurred. Clear stability and instability timers and set *lrttRequest* to invoke TimingLrttFrame.
Row 10.17-7: If a topology stability-impacting change has occurred but the instability timer has not yet expired, restart the stability timer and set *topoChanged* to TRUE.
Row 10.17-6: Otherwise remain in SHAKY state.

Row 10.17-8: If there is a change in the topology database, move from STABLE-to-SHAKY states, restart the stability and instability timers, and set *lrttRequest* to FALSE (so that Table 10.26 is not entered).
Row 10.17-9: If the validity check fails, declare a topology instability defect and go to the INVALID state.
Row 10.17-10: If excess stations appear, declare an excess stations defect and go to the INVALID state.
Row 10.17-11: If the topology is inconsistent, declare an inconsistency defect and go to the INVALID state.
Row 10.17-12: Otherwise, clear the topology inconsistency defect. Set appropriate ring-wide values within the topology database that are set only based on a valid topology. If there is no protection configuration mismatch on the ring, clear any existing protection configuration defect.

Row 10.17-15: Reachability is updated and invalid entries are cleared in the topology database. Compute the topology checksum and mark that topology checksum valid. Clear the validity of the neighbor checksums, so that neighbor stations must reconfirm their valid checksums to clear context containment at this station. Trigger transmission of TC frames and validation of secondary MAC addresses.

Row 10.17-16: If there is a change in the topology database, move from the INVALID state to the UNSTABLE state. Both the stability and instability timers are restarted.
Row 10.17-17: Otherwise remain in INVALID state.

Row 10.17-18: If there is a change in the topology database, move from VALID-to-UNSTABLE states. An indication that the topology checksum of this station is now invalid is set. Both the stability and instability timers are restarted.
Row 10.17-19: If the neighbor checksums both match, then clear context containment.
Row 10.17-20: Otherwise remain in VALID state.

10.6.7 TransmitTpFrame state machine

The TP frame contains the ringlet identifier (*rxTpFrame.ri*) on which it is sent. When a station receives a TP frame with a *ttl* set to MAX_STATIONS (indicating the frame has traveled exactly one hop), it verifies that the ringlets of both stations are connected with the same ringlet identifier value, as indicated in the received frame and by the locally configured value of *ri* associated with the span on which the frame is received.

The TP frame, like all RPR frames, contains the source MAC address of the station from which it is sent as part of the RPR header. A TP frame with *frame.ttl*=MAX_STATIONS (indicating the frame has traveled exactly one hop), thus verifies the neighbor's MAC address.

The TP frames are sent on a bi-level periodic timer. There is a configurable fast transmission period and a configurable slow transmission period (*txFastDelay* and *txSlowDelay*), with configurable ranges as defined in 10.2.3. There is also a fixed number of frames that are sent on the fast timer after any of the above triggers, as defined in 10.6.7.1. Each time a TP frame is triggered by a change in the information contained in the frame (except for a change in the *wc* or *jp* bits) or upon initialization, the fast TP frame period is used, followed by the slow TP frame period until a new frame is triggered from the station.

10.6.7.1 TransmitTpFrame state machine constants

FAST_TP_COUNT
: The number of frames transmitted on the fast TP timer, before using the slow TP timer.
  Value—8

MAX_SEQNUM
: The maximum value of the 6-bit *seqnum* field.
  Value—63

10.6.7.2 TransmitTpFrame state machine variables

currentTime
: See 10.2.9.

txFastDelay
: See 10.2.3.

fastTpFrameCount
: The number of TP frames remaining to be transmitted on the fast TP timer.

myTopoInfo
: See 10.2.5.

txSlowDelay
: See 10.2.3.

txTpTime
: The time at which the next TP frame should be sent.

tpContentChange
: See 10.2.3.

transmitTpFrame
See 10.2.3.

10.6.7.3 TransmitTpFrame state machine routines

SendTpFrame( )
Inserts a copy of the TP frame into both MAC control queues (one per ringlet). The MAC control queue is defined in Clause 6.
TimerDone(timeout)
TimerBad(timeout, delay)
See 10.2.7.

10.6.7.4 TransmitTpFrame state table

The TP frame transmit state machine specified in Table 10.18 determines when TP frames are sent on the ring from a station. This state machine uses variables *transmitTpFrame* and *tpContentChange* as inputs. The *transmitTpFrame* value indicates when to trigger fast timer based TP frame transmission. The *tpContentChange* value indicates when a change has occurred in the TP frame content, including in the jumbo preferences or protection configuration settings of the station; it also results in the increment of the sequence number sent in the TP frame.

Table 10.18—TransmitTpFrame state table

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

START	transmitTpFrame	1	transmitTpFrame = FALSE;
			fastTpFrameCount= FAST_TC_COUNT;
			txTpTime =
			currentTime + txFastDelay;
				FIRST

	fastTpFrameCount >= FAST_TC_COUNT	2		
	fastTpFrameCount >= 0 && TimerBad(txTpTime, txFastDelay)	3		
	fastTpFrameCount == 0 && TimerBad(txTpTime, txSlowDelay)	4		
	—	5	—	START
FIRST	fastTpFrameCount > 1 && TimerDone(txTpTime)	6	fastTcFrameCount -= 1; txTpFastTime = currentTime + txFastDelay;	NEAR
	TimerDone(txTpTime)	7	fastTpFrameCount = 0; txTpTime = currentTime + txSlowDelay;	
	—	8	—	START
NEAR	tpContentChange	9	myTopoInfo.sequenceNumber = ((myTopoInfo.sequenceNumber +1) % MAX_SEQNUM); tpContentChange = FALSE;	FINAL
	—	10	—	
FINAL	—	11	SendTpFrame();	START

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 10.18-1: If *transmitTpFrame* has been set, send a fixed number of fast-rate TP frames.
(The *transmitTpFrame* bit can be set by the TopologyUpdate1 or ProtectonUpdate state machines.)
Row 10.18-2: Incorrectly initialized counters behave as though *transmitTpFrame* were set TRUE.
Row 10.18-3: Incorrectly initialized *txFastDelay*-based timers behave identically.
Row 10.18-4: Incorrectly initialized *txSlowDelay*-based timers behave identically.
Row 10.18-5: Otherwise, do no initialization adjustments are necessary.

Row 10.18-6: When the initial fast timers expires, update counter and timer, then do fast rate transmissions.
Row 10.18-7: After the fast timers complete, update counter and timer, then do slow rate transmissions
Row 10.18-8: Otherwise, continue checking until a timer expires.

Row 10.18-9: If the content has changed, the sequence number is incremented before frame transmission.
Row 10.18-10: Otherwise, the sequence number is not incremented before frame transmission.
Row 10.18-11: After the timer expires, a TP frame is sent on each ringlet.

10.6.8 ReceiveTpFrame state machine

The receipt of a valid TP frame on either ringlet from any station causes the MAC control sublayer to update its current local topology image. Miscabling detection is done for each TP frame received from a neighbor station, allowing early detection of miscabling errors. These miscabling errors set the link state to SF and generate a miscabling defect indication.

10.6.8.1 ReceiveTpFrame state machine definitions

MAX_STATIONS
: See 10.2.2.

10.6.8.2 ReceiveTpFrame state machine variables

duplicateSecMacAddressDefect
: See 10.2.9.

index
: Index of topology database entry that is updated by the received TP frame.

miscabled
: An intermediate variable that indicates the presence or absence of a miscabled condition.

miscablingDefect[*ri*]
: See 10.2.9.

ri

rxTpFrame
: See 10.2.3.

tempTpFrame
: Received TP frame that is being validated prior to being passed to the topology database update state machine in Table 10.14. When validation is completed, *tempTpFrame* is set into *rxTpFrame*.

10.6.8.3 ReceiveTpFrame state machine routines

The routines below are used in the TP frame receive state machine.

AcceptTpFrame(*frame, index*)

Checks if a received TP frame passes the sequence number check, as defined by Equation 10.24. If this check is not passed, then the frame is not processed.

TRUE—The received TP frame passes the sequence number check.

FALSE—(Otherwise.)

```
Boolean                                                                    (10.24)
AcceptTpFrame(frame, index)
{
    Boolean check;

    check = (topoEntry[frame.ri][index].sequenceNumber == frame.seqnum);
    check &&= (topoEntry[frame.ri][index].valid);
    check &&= (topoEntry[frame.ri][index].macAddress == frame.sa);
    return(!check);
}
```

MatchAllStationMacToSecMac( )

TBD.

TpFrameReceived( )

This routine provides a newly received TP frame.

10.6.8.4 ReceiveTpFrame state table

This state machine handles the reception of TP frames. It runs in parallel with the other state machines defined in this clause. Upon receipt of a TP frame, it checks whether that frame passes the miscabling and sequence number checks. It sets and clears the miscabling defect on the two spans of the station.

Table 10.19—ReceiveTpFrame state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	(tempTpFrame = TpFrameReceived()) == NULL;	1	—	START
	frame.ttl != MAX_STATIONS	2	index= FindIndex(tempTpFrame);	GOOD
	frame.ri == myRI	3	miscablingDefect[ri] = FALSE;	
	—	4	miscablingDefect[ri] = TRUE;	START
GOOD	**!**AcceptTpFrame(tempTpFrame, index)	5	—	START
	MatchAllStationMacToSecMac()	6	duplicateSecMacAddressDefect = TRUE; needSecMacValidation = TRUE;	FINAL
	—	7	—	
FINAL	rxTpFrame == NULL	8	rxTpFrame = tempTpFrame;	START
	—	9	—	FINAL

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 10.19-1: Wait until a TP frame is received.
Row 10.19-2: For non-neighbor frames, generate the database entry index and continue processing.
Row 10.19-3: If the neighbor is correctly cabled, clear the miscabling defect.
Row 10.19-4: If the neighbor is incorrectly cabled, set the miscabling defect.

Row 10.19-5: If the sequence number check fails, based on a comparison of the sequence number contained in the TP frame and the corresponding entry in the topology database, the TP frame is discarded.
Row 10.19-6: If the MAC address matches one of the secondary MAC addresses. A duplicate secondary MAC address defect is set; secondary MAC address validation occurs after setting *needSecMacValidation*.
Row 10.19-7: Otherwise, no special processing occurs.

Note—The MAC address in the TP frame is compared to the secondary MAC addresses in the topology database in this state machine, rather than in the TopologyValidation state machine because secondary MAC addresses can be configured without impacting the validity of the discovered topology.

Row 10.19-8: When the topology database update (Table 10.15) is done with the received TP frame, it nulls out *rxTpFrame*. At this point, the received TP frame (stored in *tempTpFrame*) is set into *rxTpFrame* so that it is picked up by the topology database update.
Row 10.19-9: Otherwise wait until the *rxTpFrame* is released.

10.6.9 TransmitTcFrame state machine

This subclause specifies the state machine for determining when TC frames are sent on the ring from a station, and related requirements.

10.6.9.1 TransmitTcFrame state machine constants

FAST_TC_COUNT
The number of frames transmitted on the fast TC timer, before using the slow TC timer.
Value—4

10.6.9.2 TransmitTcFrame state machine variables

currentTime
See 10.2.9.
fastTcFrameCount
The number of TC frames remaining to be transmitted on the fast TC timer.
txFastDelay
txSlowDelay
See 10.2.3.
transmitTcFrame
See 10.2.3.
txTcFrame
The contents of a TC frame transmitted to a neighbor station, as defined in 10.3.2.
txTcTime
The time at which the next TC frame should be sent.

10.6.9.3 TransmitTcFrame state machine routines

SendTcFrame(*txTcFrame*)
Inserts a copy of the *txFrame* into both MAC control queues (one per ringlet). The MAC control queue is defined in Clause 6. In addition, this routine ensures that fields in the TC frame *controlDataUnit* are set as defined by Equation 10.25.

```
txTcFrame.cv = myTopoInfo.checksumValid;
txTcFrame.checksum = myTopoInfo.checksum;
```
(10.25)

TimerDone(*timeout*)
TimerBad(*timeout*, *delay*)
See 10.2.7.

10.6.9.4 TransmitTcFrame state table

The Transmit TC frame state machine specified in Table 10.20 determines when TC frames are sent on the ring from a station. This state machine handles the transmission of TC frames. It runs in parallel with the other state machines defined in this clause. It uses the input variable *transmitTcFrame* to trigger fast timer based TC frame transmission.

Table 10.20—TransmitTcFrame state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	transmitTcFrame	1	transmitTpFrame = FALSE; fastTpFrameCount = FAST_TC_COUNT; txTcTime = currentTime + txFastDelay;	FIRST
	fastTcFrameCount >= FAST_TC_COUNT	2		
	fastTcFrameCount != 0 && TimerBad(txTcTime, txFastDelay)	3		
	fastTcFrameCount == 0 && TimerBad(txTcTime, txSlowDelay)	4		
	—	5	—	START
START	fastTcFrameCount > 1 && TimerDone(txTcTime)	6	fastTcFrameCount -= 1; txTcTime = currentTime + txFastDelay;	FINAL
	TimerDone(txTcTime)	7	fastTcFrameCount = 0; txTcTime = currentTime + txSlowDelay;	
	—	8	—	START
FINAL	—	9	SendTcFrame(txTcFrame);	START

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 10.20-1: If *transmitTcFrame* has been set, send a fixed number of fast-rate TC frames.
(The *transmitTcFrame* bit can be set by the TopologyValidation state machines after the topology is valid).
Row 10.20-2: Incorrectly initialized timers behave as though *transmitTcFrame* were set TRUE.
Row 10.20-3: Incorrectly initialized *txTcFastDelay*-based timers behave identically.
Row 10.20-4: Incorrectly initialized *txTcSlowDelay*-based timers behave identically.
Row 10.20-5: Otherwise, do no initialization adjustments are necessary.

Row 10.20-6: When the initial fast timers expires, update counter and timer, then do fast rate transmissions.
Row 10.20-7: After the fast timers complete, update counter and timer, then do slow rate transmissions.
Row 10.20-8: Otherwise, continue checking until a timer expires.

Row 10.20-9: After the timer expires, a TC frame is sent on each ringlet.

10.6.10 ReceiveTcFrame state machine

This subclause specifies the state machine for handling the receipt of TC frames at a station, and related requirements.

The receipt of a TC frame on a ringlet causes the MAC control sublayer to update its current local topology image, provided an SF condition is not present on the receiving span.

10.6.10.1 ReceiveTcFrame state machine variables

miscablingDefect[*ri*]
See 10.2.9.
myTopoInfo
See 10.2.5.
ri
See 10.2.3.
rxFrame
Received TC frame that is being validated prior to being used to update the topology database.
topoValid
See 10.2.3.
transmitTcFrame
See 10.2.3.

10.6.10.2 ReceiveTcFrame state machine routines

TcFrameReceived( )
This routine provides a newly received TC frame.

10.6.10.3 ReceiveTcFrame state table

This state machine handles the reception of TC frames. It runs in parallel with the other state machines defined in this clause. Upon receipt of a TC frame on a ringlet, it checks whether a miscabling defect is in effect on the span receiving from that ringlet. If there is no miscabling defect, the contents of the TC frame are used to update the topology database.

Table 10.21—ReceiveTcFrame state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	(rxFrame = TcFrameReceived()) != NULL && !miscablingDefect[rxFrame.ri]	1	ri = rxFrame.ri;	CHECK
	—	2	—	START
NEXT	!topoValid	3	—	FINAL
	myTopoInfo.neighborCheckValid[ri] != rxFrame.checksumValid	4	transmitTcFrame = TRUE;	
	myTopoInfo.neighborChecksum[ri] != rxFrame.checksum	5		
	—	6	—	
FINAL	—	7	myTopoInfo.neighborChecksum[ri] = rxFrame.checksum; myTopoInfo.neighborCheckValid[ri] = rxFrame.checksumValid;	START

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 10.21-1: A TC frame is received and there is no miscabling defect; continue checking.
Row 10.21-2: Otherwise, remain in START.

Row 10.21-3: If the topology is not valid, save (but otherwise ignore) the checksum update values.
Row 10.21-4: If the neighbor's checksum is invalid, trigger fast-timer TC frame transmissions.
Row 10.21-5: If the neighbor's checksum is different, trigger fast-timer TC frame transmissions.
Row 10.21-6: Otherwise, save (but otherwise ignore) the checksum update values.

Row 10.21-7: The contents of the TC frame update the topology database; return to START.

10.6.11 TransmitAtdFrame state machine

Stations send ATD frames periodically, based on the configurable *atdTimerDelay* transmission-interval parameter.

10.6.11.1 TransmitAtdFrame state machine variables

atdTimerDelay
See 10.2.3.
currentTime
See 10.2.8.
txAtdTimeout
The time at which the next ATD frame should be transmitted.

10.6.11.2 TransmitAtdFrame state machine routines

TimerDone(*timeout*)
TimerBad(*timeout*, *delay*)
See 10.2.7.
SendAtdFrame( )
Inserts a copy of the ATD frame into both MAC control queues (one per ringlet).
SendAtdFrame( )
Inserts a copy of the ATD frame into both MAC control queues (one per ringlet).

10.6.11.3 TransmitAtdFrame state table

The ATD frame transmit state machine specified in Table 10.22 determines when ATD frames are sent on the ring from a station.

Table 10.22—TransmitAtdFrame state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	TimerDone(txAtdTimeout)	1	txAtdTimeout = currentTime + atdTimerDelay;	FINAL
	TimerBad(txAtdTimeout, atdTimerDelay)	2		
	—	3	—	START
FINAL	—	4	SendAtdFrame();	START

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 10.22-1: If the current timer has expired, update restart the timer perform timeout operations.
Row 10.22-2: If the timer is incorrectly initialized, update restart the timer perform timeout operations.
Row 10.22-3: Otherwise, wait for the transmission timer timeout.

Row 10.22-4: Send an ATD frame, before waiting for the next timeout occurrence.

10.6.12 ReceiveAtdFrame state machine

The receipt of a ATD frame on either ringlet from any station causes the MAC control sublayer to update its current local topology image.

10.6.12.1 ReceiveAtdFrame state machine variables

hops
ri
rxFrame
rxAtdFrame
See 10.2.3.

10.6.12.2 ReceiveAtdFrame state machine routines

The routines below are used in the TP frame receive state machine.

AtdFrameReceived( )
This routine provides a newly received ATD frame.

10.6.12.3 ReceiveAtdFrame state table

The reception of ATD frames involves updates to the topology database, as specified in Table 10.23.

Table 10.23—ReceiveAtdFrame state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	(rxFrame = AtdFrameReceived()) != NULL;	1	—	EXEC
	—	2	—	START
FINAL	rxAtdFrame == NULL	3	rxAtdFrame = rxFrame;	START
	—	4	—	FINAL

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 10.23-1: An ATD frame is received and the index values for the topology database entry are computed.
Row 10.23-2: Otherwise, remain in START.

Row 10.23-3: Wait for the last ATD frame to be processed.
Row 10.23-4: Then process the recently received ATD frame.

10.6.13 SecondaryUpdate state machine

This subclause specifies the state machine for determining when the topology database is updated with secondary MAC addresses, and when defects are set.

10.6.13.1 SecondaryUpdate state machine constants

MAX_SEC_MAC
: The maximum number of secondary MAC addresses allowed on a ring. This equals 32.

10.6.13.2 SecondaryUpdate state machine variables

index
: Index of topology database entry that is updated by the received ATD frame.

myTopoInfo
: See 10.2.5.

ri
: See 10.2.3.

rxAtdFrame
: The contents of an ATD frame received from the ring (see 10.3.5 and 10.4).

mac
: A copy of a requested secondary MAC address, used to detect secondary MAC address changes.

topoEntry
: See 10.2.6.

totalHopsRx[ri]
: See 10.2.3.

10.6.13.3 SecondaryUpdate state machine routines

FindIndex(*frame*)
: See 10.2.7.

SecMacAttPresent( )
: Indicates if a secondary MAC address ATT is contained in the received ATD frame.
  TRUE—A secondary MAC address ATT is contained in the received ATD frame.
  FALSE—(Otherwise.)

SecMac1ConfigRequest( )
: Provides the value of a secondary MAC address 1 to be configured on the local station.

SecMac2ConfigRequest( )
: Provides the value of a secondary MAC address 2 to be configured on the local station.

10.6.13.4 SecondaryUpdate state table

The SecondaryUpdate state machine is specified in Table 10.24 and the secondary MAC ATT format is specified in 10.4.7. This state machine handles the topology database update resulting from a received ATD frame or secondary MAC address configuration at the local station. (A distinct SecondaryValidation state machine is called to validate the secondary MAC address assignments.)

Table 10.24—SecondaryUpdate processing state table

Current state		Row	Next state	
state	condition		action	state

START	rxAtdFrame != NULL	1	index = FindIndex(rxAtdFrame);
			miscabled = (frame.ri != myRI &&
			frame.ttl == MAX_STATIONS);
				NEXT

	(mac = SecMac1ConfigRequest()) != NULL	2	myTopoInfo.secMacAddr1Used &&= (mac == myTopoInfo.secMacAddress1); myTopoInfo.secMacAddress1 = mac; SecondaryValidation();	START
	(mac = SecMac2ConfigRequest()) != NULL	3	myTopoInfo.secMacAddr2Used = (mac == myTopoInfo.secMacAddress2); myTopoInfo.secMacAddress2 = mac; SecondaryValidation();	
	—	4	—	
NEXT	miscabled	5	—	START
	SecMacAttPresent() && topoEntry[ri][index].valid	6	—	TEST1
	—	7	—	FINAL
TEST1	rxAtdFrame.secMacAddress1 != topoEntry[ri][index].secMacAddress1	8	topoEntry[ri][index].secMacAddress1 = rxAtdFrame.secMacAddress1; topoEntry[ri][index].secMacAddr1Used = FALSE; SecondaryValidation();	TEST2
	—	9	—	
TEST2	rxAtdFrame.secMacAddress2 != topoEntry[ri][index].secMacAddress2	10	topoEntry[ri][index].secMacAddress2 = rxAtdFrame.secMacAddress2; topoEntry[ri][index].secMacAddr2Used = FALSE; SecondaryValidation();	FINAL
	—	11	—	
FINAL	—	12	rxAtdFrame = NULL	START

In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Row 10.24-1: When an ATD frame is received, proceed to the NEXT state.
Row 10.25-2: When configuring a new *secMacAddress1*, clear *secMacAddr1Used* to FALSE.

Row 10.25-3: When configuring a new *secMacAddress2*, clear *secMacAddr2Used* to FALSE.
Row 10.24-4: Otherwise, return to START state.

Row 10.24-6: Check more if a valid secondary MAC address ATT is present in the received ATD frame.
Row 10.24-7: Otherwise, terminate ATD frame checking.

Row 10.24-8: If the ATT's *secMacAddress1* differs from the database, clear *secMacAddr1Used* to FALSE.
Row 10.24-9: Otherwise, continue ATD frame checking.

Row 10.24-10: If the ATT's *secMacAddress2* differs from the database, clear *secMacAddr2Used* to FALSE.
Row 10.24-11: Otherwise, continue ATD frame checking.

Row 10.24-12: Discard the processed *rxAtdFrame* and return to the START state.

10.6.14 SecondaryValidation state machine

The SecondaryValidation state machine ensures that:

a) A critical severity *duplicateSecMacAddressDefect* defect is generated by the following errors:
   1) The ATD's MAC address duplicates a secondary MAC address of the receiving station.
   2) An ATD's secondary MAC address duplicates the MAC address of the receiving station.
   3) An ATD's secondary MAC addresses duplicates a station's secondary MAC address.
   4) An ATD's secondary MAC addresses would configure a duplicate secondary MAC address.
b) A critical severity *maxSecMacAddressDefect* defect is generated if more than MAX_SEC_MAC (a value of at least 32 for any station) secondary MAC addresses are present on the ring.

10.6.14.1 SecondaryValidation state machine constants

MAX_SEC_MAC
The maximum number of secondary MAC addresses allowed on a ring. This equals 32.

10.6.14.2 SecondaryValidation state machine variables

duplicateSecMacAddressDefect
See 10.2.9.
maxSecMacAddressDefect
See 10.2.9.
myTopoInfo
See 10.2.5.
needSecMacValidation
See 10.2.3.
ringInfo
See 10.2.4.
mac
A copy of a requested secondary MAC address, used to detect secondary MAC address changes.
topoEntry
See 10.2.6.
topoValid
See 10.2.3.
totalHopsRx[ri]
See 10.2.3.

10.6.14.3 SecondaryValidation state machine routines

ExceedsMaxSecMac()

Indicates whether the secondary MAC addresses in the topology database exceeds MAX_SEC_MAC, as defined by Equation 10.26.

TRUE—The number of secondary MAC addresses exceeds MAX_SEC_MAC.

FALSE—(Otherwise.)

(10.26)

```
Boolean
ExceedsMaxSecMac()
{
    int ringlet, hop, mySec;
    int secMacTotal = 0;

    if (myTopoInfo.secMacAddress1 != 0)
      secMacTotal += 1;
    if (myTopoInfo.secMacAddress2 != 0)
      secMacTotal += 1;
    mySec = secMacTotal;
    for (ringlet = 0; ringlet < 2; ringlet += 1)
    {
        if (ringInfo.topoType == CLOSED_RING)
            secMacTotal = mySec;
        for (hop = 1; hop <= totalHopsRx[ringlet]; hop += 1)
        {
            if (!topoEntry[ringlet][hop].valid)
                continue
            secMacTotal += (topoEntry[ringlet][hop].secMacAddress1 != 0);
            secMacTotal += (topoEntry[ringlet][hop].secMacAddress2 != 0);
        }
        if (secMacTotal > MAX_SEC_MAC)
            return(TRUE);
    }
    return(FALSE);
}
```

MarkAllReachableSecMacsInUse()

Marks all valid and reachable non-zero secondary MAC addresses in the topology database in use, as defined by Equation 10.27.

(10.27)

```
void
MarkAllReachableSecMacsInUse()
{
    int ringlet, hop;

    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hop = 1; hop <= totalHopsRx[ringlet]; hop += 1)
        {
            if (!topoEntry[ringlet][hop].valid ||
             !topoEntry[ringlet][hop].reachable)
                continuel
            if (topoEntry[ringlet][hop].secMacAddress1 != 0)
                 topoEntry[ringlet][hop].secMacAddr1Used = TRUE;
            if (topoEntry[ringlet][hop].secMacAddress2 != 0)
                 topoEntry[ringlet][hop].secMacAddr2Used = TRUE;
        }
    return(FALSE);
}
```

MarkSomeSecMacsInUse()

Marks a subset of the secondary MAC addresses in the topology database in use in the event of a secondary MAC validation defect. In the event of a *maxSecMacAddressDefect*, the algorithm used to determine which secondary MAC addresses are marked in use is implementation dependent and beyond the scope of this standard. In the event of a *duplicateSecMacAddressDefect*, secondary MAC addresses that duplicate a valid and reachable primary MAC address or secondary MAC address should not be marked in use.

MatchAllSecMacToSecMac()

Indicates that a secondary MAC address of the local station matches a secondary MAC address contained in some valid and reachable entry in the topology database, as defined by Equation 10.28.

TRUE—A secondary MAC address duplicates a secondary MAC address of the local station.

FALSE—(Otherwise.)

(10.28)

```
Boolean
MatchAllSecMacToSecMac()
{
    int ringlet, hop;
    TopoEntry entry;

    if ((myTopoInfo.secMacAddress1 == myTopoInfo.secMacAddress2) &&
        (myTopoInfo.secMacAddress1 != 0))
      return(TRUE);
    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hop = 1; hop <= totalHopsRx[ringlet]; hop += 1)
        {
            entry= topoEntry[ringlet][hop];
            if (!entry.valid)
                continue;
            if (myTopoInfo.secMacAddress1 != 0)
               if (entry.secMacAddress1 == myTopoInfo.secMacAddress1 ||
                 entry.secMacAddress2 == myTopoInfo.secMacAddress1)
                     return(TRUE);
             if (myTopoInfo.secMacAddress2 != 0)
                if (entry.secMacAddress1 == myTopoInfo.secMacAddress2 ||
                  entry.secMacAddress2 == myTopoInfo.secMacAddress1)
                      return(TRUE);
        }
    return(FALSE);
}
```

MatchAllSecMacToStationMac()

Indicates that the MAC address of the local station matches a secondary MAC address contained in some valid and reachable entry in the topology database, as defined by Equation 10.29.

TRUE—A secondary MAC address duplicates the station MAC address.

FALSE—(Otherwise.)

(10.29)

```
Boolean
MatchAllSecMacToStationMac()
{
    int ringlet, hop;

    if (myTopoInfo.secMacAddress1 == myTopoInfo.macAddress)
        return(TRUE);
    if (myTopoInfo.secMacAddress2 == myTopoInfo.macAddress)
        return(TRUE);
    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hop = 1; hop <= totalHopsRx[ringlet]; hop += 1)
        {
            if (!topoEntry[ringlet][hop].valid)
                continue
            if (topoEntry[ringlet][hop].secMacAddress1 == myTopoInfo.macAddress)
                return(TRUE);
            if (topoEntry[ringlet][hop].secMacAddress2 == myTopoInfo.macAddress)
                return(TRUE);
      }
    return(FALSE);
}
```

MatchAllStationMacToSecMac()

See 10.2.7.

SecMacAttPresent( )

Indicates if a secondary MAC address ATT is contained in the received ATD frame.

TRUE—A secondary MAC address ATT is contained in the received ATD frame.

FALSE—(Otherwise.)

10.6.14.4 SecondaryValidation state table

The SecondaryValidation state machine validates secondary MAC addresses, as specified in Table 10.25.

Table 10.25—SecondaryValidation state table

Current state		Row	Next state	
state	**condition**		**action**	**state**
START	!topoValid	1	—	FINAL
	(maxSecMacAddressDefect = ExceedsMaxSecMac()) == TRUE	2		
	MatchAllSecMacToStationMac()	3	duplicateSecMacAddressDefect = TRUE;	
	MatchAllStationMacToSecMac()	4		
	—	5	duplicateSecMacAddressDefect = MatchAllSecMacToSecMac();	
FINAL	maxSecMacAddressDefect	6	MarkSomeSecMacsInUse();	RETN
	duplicateSecMacAddressDefect	7		
	topoValid	8	MarkAllReachableSecMacsInUse();	
	—	9	—	

Row 10.25-1: If the topology is not valid, proceed to FINAL state.
Row 10.25-2: Record a secondary MAC addresses defect and proceed to the FINAL state.
Row 10.25-3: Record a defect if the secondary MAC address matches the local station MAC address.
Row 10.25-4: Record a defect if a station MAC address matches a local secondary MAC address.
Row 10.25-5: Record a defect if a secondary MAC address matches a local secondary MAC address.

Row 10.25-6: With a *maxSecMacAddressDefect*, mark a subset of in-use secondary MAC addresses.
Row 10.25-7: With a *duplicateSecMacAddressDefect*, mark a subset of in-use secondary MAC addresses.
Row 10.25-8: If a valid-topology validation passes, mark reachable secondary MAC addresses.
Row 10.25-9: Otherwise, return to caller.

10.6.15 TimingLrttFrame state machine

A station deploying the conservative method of rate computation sends unicast loop round trip time (LRTT) request frames individually on each ringlet to all other stations on the ring. This is done when a new topology is validated, and may optionally be done at any other time with at least 10 seconds between successive measurement cycles. A station receiving an LRTT request sends a response via the opposing ringlet. Both LRTT request and LRTT response frames are sent as subclassA0. All stations, no matter which fairness mode is used, respond to an LRTT request by sending a LRTT response via the opposing ringlet. The sending station is known as the requestor and the receiving station is known as the responder. On receiving an LRTT response, the requestor computes the LRTT and maintains this value in the topology database. The LRTT value is computed as defined by Equation 10.30.

```
lastRcvdLrtt = (currentTime - latencyTimestamp) +                              (10.30)
  (hopsToResponder * advertisementInterval) - (tailLatencyOut - tailLatencyIn)
```

The LRTT value is used by Clause 9 in the computation of the FRTT. The format of the LRTT frames are specified in Figure 10.12 and Figure 10.13.

The responder may measure or estimate the amount of latency added between reception of the LRTT request frame and transmission of the LRTT response frame, and return that latency amount with the response frame.

The LRTT fields in the topology database are not cleared when the topology is unstable. The LRTT fields are expected to be updated within one second after topology stabilizes. If the measurement is not completed within one second, a LRTT defect is triggered. Subsequent measurements are optional. A station shall respond to a LRTT request within 100 ms. If a LRTT response is received after 100 ms, the station shall ignore it. The requestor shall send another LRTT request to a station that does not respond at 200 ms intervals until a response is received, or until a LRTT incomplete defect is triggered.

If topology becomes unstable while the LRTT measurement is taking place, the LRTT measurement is aborted.

10.6.15.1 TimingLrttFrame state machine definitions

COMPLETION_TIME
: Time allowed for completion of LRTT measurement.
  Value—100 ms.

10.6.15.2 TimingLrttFrame state machine variables

advertisingInterval
currentTime
: See 10.2.8.

lrttComplete
: Indication if LRTT measurement is completed.

lrttreTime
: The time at which the *lrttreDelay* is due to expire.

lrttreDelay
: Retransmit timer for LRTT before LRTT measurement is completed.
  Value—200 ms

lrttTime
: The time at which the *lrttDelay*-specified delay is due to expire.

lrttDelay
The delay time to claim LRTT measurement defect.
Value—1 second
lrttFrame
A LRTT request or repsonse frame.
lrttIncompleteDefect
See 10.2.9.
lrttRequest
See 10.2.3.
myTopoInfo
See 10.2.5.
ri
See 10.2.3.
topoEntry
See 10.2.6.
topoStability
See 10.2.3.
totalHopsRx[*ri*]
See 10.2.3.

10.6.15.3 TimingLrttFrame state machine routines

FindIndex(*frame*)
See 10.2.7.
LrttFrameReceived( )
This routine provides a newly received LRTT frame.
LrttRequestHandle(*frame*)
Handle LRTT request frame, as defined by Equation 10.31.

```
void                                                    (10.31)
LrttRequestHandle(frame)
{
    int txRinglet;
    Frame txFrame;

    txFrame.controlType = CT_LRTT_RSP;
    txFrame.da = frame.sa;
    txFrame.tailLatencyIn = frame.tailLatencyIn;
    txFrame.tailLatencyOut = currentTime;
    txRinglet = Other(frame.ri);
    SendFrame(select, txFrame);
}
```

LrttRequestTransmit( )
Transmit LRTT request frame to start LRTT measurement, as defined by Equation 10.32.

```
void                                                                    (10.32)
LrttRequestTransmit(newRequest)
{
    int ringlet, txRinglet, hop;
    Frame frame;

    for(ringlet = 0, ringlet < 2; ringlet += 1)
    {
        for (hop = 1; hop <= totalHopsRx[ringlet]; hop += 1)
       {
            if(topoStability == UNSTABLE)
               return;
            if((newRequest || topoEntry[ringlet][hop].lrtt == NULL))
            {
                frame.controlType = CT_LRTT_REQ;
                frame.da = topoEntry[ringlet][hop].macAddress;
                frame.latencyTimestamp = currentTime;
                frame.tailLatencyIn = 0;
                frame.tailLatencyOut = 0;.
                if(newRequest)
                    topoEntry[ringlet][hop].lrtt = NULL;
                txRinglet = Other(ringlet);
                SendFrame(txRinglet,frame);
            }
        }
    }
    return;
}
```

LrttResponseHandle(*frame*)

Handle LRTT response frame, as defined by Equation 10.33. The LRTT measurement to a destination station from a source station on ringlet *ri* (returning on ringlet *Other*(*ri*)) is stored in the entry of the topology database on corresponding to the ringlet from which the LRTT response frame is received (*Other*(*ri*)), and to the number of hops separating the destination station from the source station.

```
void                                                                    (10.33)
LrttResponseHandle(frame)
{
int hop, index, ringlet, latency;

    index = FindIndex(frame);

    latency = currentTime - frame.latencyTimestamp;
    if (latency > COMPLETION_TIME)
        return;

    topoEntry[ringlet][index].lrtt = latency + (index * advertisementInterval) -
     (frame.tailLatencyOut - frame.tailLatencyIn);

    for (ringlet = 0; ringlet < 2; ringlet += 1)
        for (hop = 1; hop <= totalHopsRx[ringlet]; hop += 1)
            if(topoEntry[ringlet][hop].lrtt == NULL)
               return;
    lrttComplete = TRUE;
    return;
}
```

SendFrame(*ri*, *frame*)

Send a frame to ringlet *ri*.

StartLrttreTimer( )
Start *lrttreDelay* timer.
TimerDone(*timeout*)
See 10.2.7.

10.6.15.4 TimingLrttFrame state table

The LRTT frame transmit state machine specified in Table 10.26 handles transmission and reception of LRTT frames. In the case of any ambiguity between the text and the state machine, the state machine shall take precedence. The notation used in the state table is described in 3.4.

Table 10.26—TimingLrttFrame state table

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

START	lrttRequest && myTopoInfo.conservativeMode	1	lrttTime = currentTime + lrttDelay;
			lrttreTime = currentTime + lrttreDelay;
			lrttComplete = FALSE;
			lrttRequest = FALSE;
			LrttRequestTransmit(TRUE);
				START

	(lrttFrame=LrttFrameReceived()) !=NULL;	2	—	NEAR
	!lrttComplete && TimerDone(lrttreTime);	3	LrttRequestTransmit(FALSE); lrttreTime = currentTime + lrttreDelay;	START
	!lrttComplete && TimerDone(lrttTime);	4	lrttIncompleteDefect=TRUE;	
	—	5	—	
NEAR	lrttFrame.controlType==CT_LRTT_REQ	6	lrttFrame.tailLatencyIn= currentTime; LrttRequestHandle(lrttFrame);	START
	lrttFrame.controlType == CT_LRTT_RES	7	LrttResponseHandle(lrttFrame);	FINAL
FINAL	lrttComplete	8	lrttIncompleteDefect=FALSE;	START
	—	9	—	

Row 10.26-1: If conservative rate adjustment is deployed in the station, then LRTT measurement will be triggered after *lrttRequest* is set to TRUE.
Row 10.26-2: If a LRTT request/response frame is received, proceed to RECEIVE state to process it.
Row 10.26-3: If *lrttreDelay* is expired and LRTT measurement is not complete, a LRTT request is retransmitted. The parameter of *LrttRequestTransmit* is set to FALSE because this is not a new request.
Row 10.26-4: If LRTT measurement is not complete and the LRTT timer has expired, a LRTT incomplete defect is triggered.
Row 10.26-5: Otherwise do nothing.

Row 10.26-6: If a a LRTT request frame is received, the current time stamp is placed into *tailLatencyIn* and processed in the *lrttRequestHandle*() routine.
Row 10.26-7: If a LRTT response frame is received, the frame is processed.

Row 10.26-8: If the LRTT measurement is complete, the LRTT timers are cleared.
Row 10.26-9: Otherwise, continue LRTT processing.

10.7 Protection special cases

10.7.1 Edge report cases

A few protection scenarios are worthy of special consideration, as illustrated in Figure 10.29. Within this figure, station S1 updates its database based on information received from the east-side stations S2, S3, and S4. Any TP frames received over a SF link are saved (for diagnostic purposes), but are not marked valid or reachable, as illustrated in Figure-Cell 10.29-a. Any TP frames received over a non-SF link, reporting a near-side neighbor station edge, are also saved (for diagnostic purposes) but are marked valid and not reachable, as illustrated in Figure-Cell 10.29-b.

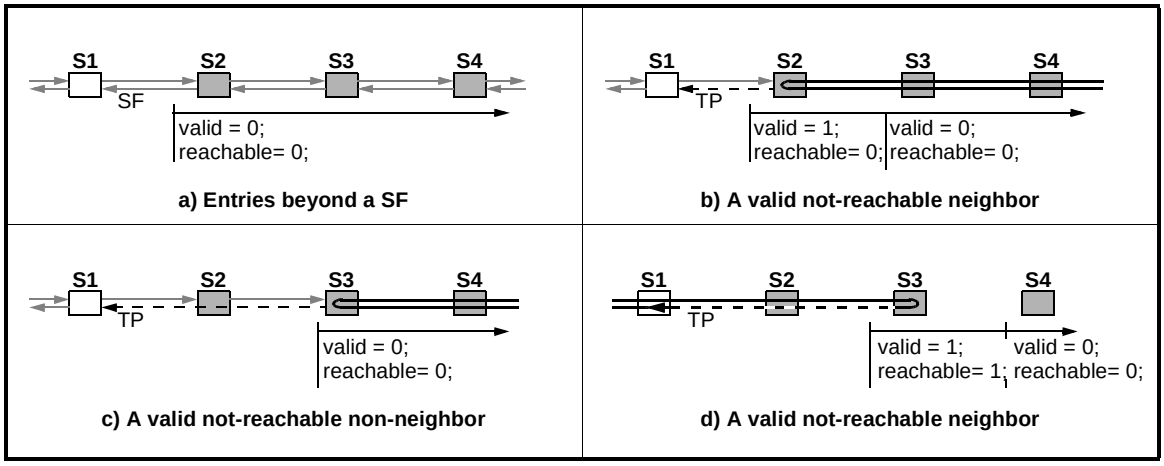


Figure 10.29—Edge condition updates

If a near-side station S3 edge is reported, station S3 (and beyond) is marked not-valid and not-reachable, as illustrated in Figure-Cell 10.29-c. If far-side station S3 edge is reported, then stations S4 (and beyond) is marked not-valid and not-reachable, as illustrated in Figure-Cell 10.29-d.

10.7.2 Station connectivity failures

A station can lose the internal connections between east-side and west-side components, as illustrated in Figure 10.30. An example is a center-wrap station includes with separate hardware modules (east and west) that are interconnected and the interconnecting cable or circuitry fails.

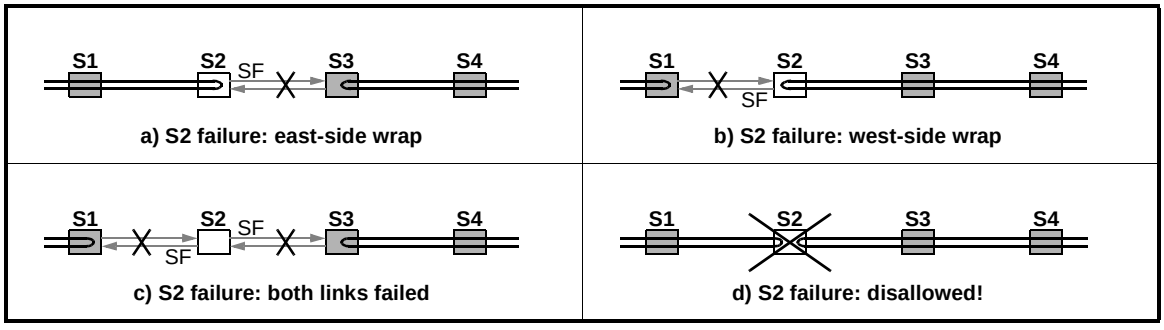


Figure 10.30—Edge condition updates

To accommodate such failures, a station may wrap its west-side span and assert SF on its east-side span, as illustrated in Figure-Cell 10.30.a. The SF can be communicated from station S2-to-S3 by either by suspending transmission of keepalives on the S2-to-S3 link, or reporting SF in the transmitted TP frames. Alternatively, a station may wrap its east-side span and assert SF on its west-side span, as illustrated in Figure-Cell 10.30.b.

Alternatively, station S2 could report a failure on both spans, which results in the station disappearing from the ring. The failure is communicated either by suspending transmission of keepalives on both spans, or by reporting SF in TP frame transmitted in both directions.

However, station S2 shall never wrap in both directions, as illustrated in Table-Cell 10.29-d. This would have unduely complicated topology discovery, since the same station could then appear at both edge locations. This station could then be confused with a pair of stations with duplicate identical MAC addresses. To avoid such confusion (or complexities necessary to resolve confusion), the double-wrap stations behavior is prohibited.

10.7.3 Topology inconsistency

Checking for topology database consistency can often identify duplicate MAC address errors, due to the irregular or asymmetric nature of the apparent closed-ring topology, as illustrated in Figure-Cell 10.31-a and Figure-Cell 10.31-b, respectively.

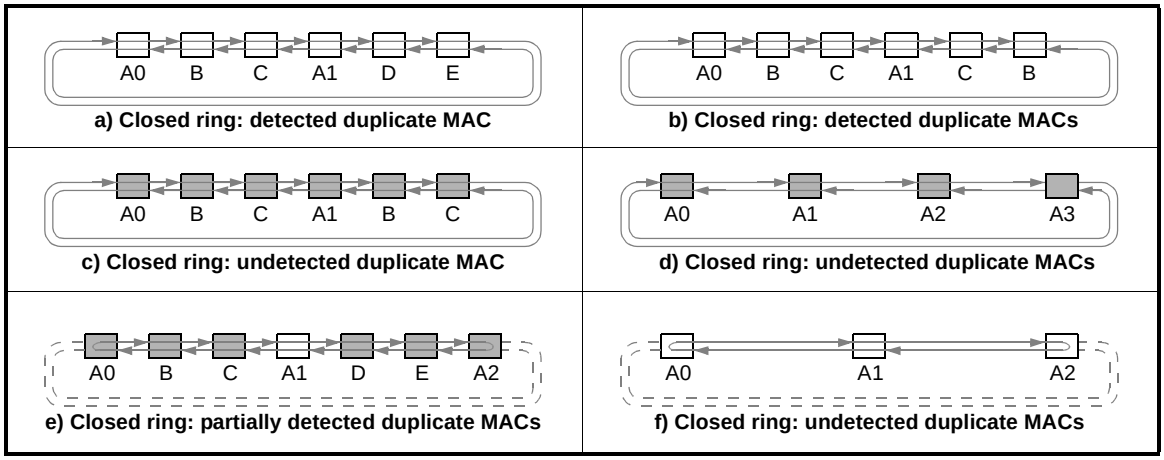


Figure 10.31—Span failure and recovery

Symmetric topologies with duplicate MAC address errors cannot be detected on closed-ring topologies, as illustrated in Figure-Cell 10.31-c. Similarly, the all-duplicate MAC address errors of closed-ring Figure-Cell 10.31-d cannot be readily distinguished from a station looped back to itself.

Some open-ring topologies with duplicate MAC address errors can be detected be certain stations, as illustrated in Figure-Cell 10.31-e. However, the all-duplicate MAC address errors of closed-ring Figure-Cell 10.31-f cannot (as with open-ringl topologies) be readily distinguished from a station looped back to itself.

10.8 Protocol Implementation Conformance Statement (PICS) proforma for Clause 10[1]

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

No valid items should be assumed to be present in this PICS proforma. Conformance of any type to any portion of this clause shall not be claimed on the basis of any item in this section. This standards draft shall not be considered to be complete until this PICS proforma is complete. The editors estimate that the level of completeness of this PICS proforma is 100%.

Comments to add and delete PICS items are not necessary, as the editors will do this automatically with the addition and deletion of mandatory items. Review of the current PICS proforma for completeness and accuracy is highly appreciated by the editors, but the lack thereof should be considered strictly editorial.

10.8.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 10, Topology discovery and protection, shall complete the following Protocol Implementation Conformation Statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the same, can be found in Annex A of IEEE Std 802.1Q-1998.

10.8.2 Identification

10.8.2.1 Implementation identification

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

a—Required for all implementations.

b—May be completed as appropriate in meeting the requirements for the identification.

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

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

10.8.2.2 Protocol summary

Identification of protocol standard	IEEE Std 802.17-200x, Resilient packet ring access method and physical layer specifications, Topology discovery and protection
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [ ] Yes [ ] (The answer Yes means that the implementation does not conform to IEEE Std 802.17-200x.)	

Date of Statement	

10.8.3 PICS tables for Clause 10

10.8.3.1 State machines

Item	Feature	Subclause	Value/Comment	Status	Support
SM1	TopologyDbUpdate1	10.6.3	Topology database update state machine 1 is supported	M	Yes [ ]
SM2	TopologyDbUpdate12	10.6.4	Topology database update state machine 2 is supported	M	Yes [ ]
SM3	ProtectionUpdate	10.6.5	Protection update state machine is supported	M	Yes [ ]
SM4	TopologyValidation	10.6.6	Topology validation state machine is supported	M	Yes [ ]
SM5	TransmitTpFrame	10.6.7	Transmit TP frame state machine is supported	M	Yes [ ]
SM6	ReceiveTpFrame	10.6.8	Receive TP frame state machine is supported	M	Yes [ ]
SM7	TransmitTcFrame	10.6.9	Transmit TC frame state machine is supported	M	Yes [ ]
SM8	ReceiveTcFrame	10.6.10	Receive TC frame state machine is supported	M	Yes [ ]
SM9	TransmitAtdFrame	10.6.11	Transmit ATD frame state machine is supported	M	Yes [ ]
SM10	ReceiveAtdFrame	10.6.12	Receive ATD frame state machine is supported	M	Yes [ ]
SM11	SecondaryUpdate	10.6.13	Secondary MAC update state machine is supported	M	Yes [ ]
SM12	SecondaryValidation	10.6.14	Secondary MAC validation state machine is supported	M	Yes [ ]
SM13	TimingLrttFrame	10.6.15	Timing of LRTT frames state machine is supported	M	Yes [ ]

10.8.3.2 Frame formats

Item	Feature	Subclause	Value/Comment	Status	Support
FF1	TP frame	10.3.1	TP frames are transmitted by the MAC in the format defined	M	Yes [ ]
FF2	TC frame	10.3.2	TC frames are transmitted by the MAC in the format defined	M	Yes [ ]
FF3	LRTT request frame	10.3.3	LRTT request frames are transmitted by the MAC in the format defined	FA6b:M	Yes [ ]
FF4	Tail latency in field setting	10.3.3	Requestor sets this field to zero	FF3:M	Yes [ ]
FF5	Tail latency out field setting	10.3.3	Requestor sets this field to zero	FF3:M	Yes [ ]
FF6	LRTT response frame	10.3.4	LRTT response frames are transmitted by the MAC in the format defined	M	Yes [ ]
FF7	Tail latency in field setting	10.3.4	Responder clears this field to zero	M	Yes [ ]
FF8	Tail latency out field setting	10.3.4	Responder clears this field to zero	M	Yes [ ]
FF9	ATD frame	10.3.5	ATD frames are transmitted by the MAC in the format defined	M	Yes [ ]
FF10	ATT types usable	10.3.5	All ATT types usable within ATD frame	M	Yes [ ]
FF11	Type dependent length	10.3.5	Length of type dependent value information less than 1024 bytes	M	Yes [ ]

10.8.3.3 ATT entries

Item	Feature	Subclause	Value/Comment	Status	Support
ATT1	Weights other than 1 advertised in weight ATT	10.4.1	Weights other than 1 advertised in weight ATT	M	Yes [ ]
ATT2	Weight of 1 (if on both ringlets) does not need to be advertised	10.4.1	Weight of 1 on both ringlets may be advertised	O	Yes [ ] No [ ]
ATT3	Non-zero reserved bandwidth advertised in station bandwidth ATT	10.4.2	Non-zero reserved bandwidth advertised in station bandwidth ATT	M	Yes [ ]
ATT4	Zero reserved bandwidth on both ringlets does not need to be advertised	10.4.2	Zero reserved bandwidth on both ringlets may be advertised	O	Yes [ ] No [ ]
ATT5	Reserved bandwidth within station bandwidth ATT	10.4.2	Uses same normalization as in fairness frames	M	Yes [ ]
ATT6	Excess reserved rate defect	10.4.2	Unreserved bandwidth less than zero triggers defect	M	Yes [ ]
ATT7	Station settings ATT with non-default values	10.4.3	Included in ATD frame if it contains any non-default value	M	Yes [ ]
ATT8	Station settings ATT		May be included in ATD frame with default values	O	Yes [ ] No [ ]
ATT9	Station name ATT	10.4.4	Station name ATT may be included in ATD frame	O	Yes [ ] No [ ]
ATT10	Station management ATT	10.4.5	Station management ATT may be included in ATD frame	O	Yes [ ] No [ ]
ATT11	Station ifIndex ATT	10.4.6	Station ifIndex ATT may be included in ATD frame	O	Yes [ ] No [ ]
ATT12	Secondary MAC address ATT	10.4.7	Included in ATD frame if any secondary MAC is registered	O	Yes [ ] No [ ]
ATT13	Secondary MAC address setting	10.4.7	First secondary MAC address set to zeros if not configured	M	Yes [ ]
ATT14	Secondary MAC address setting	10.4.7	Second secondary MAC address set to zeros if not configured	M	Yes [ ]
ATT15	Organization-specific ATT	10.4.8	Organization-specific ATT may be included in ATD frame	O	Yes [ ] No [ ]

10.8.3.4 Protection

Item	Feature	Subclause	Value/Comment	Status	Support
PP1	Link or station failure detection	??	Detection in under 10 ms	M	Yes [ ]
PP2	Sub-50 ms protection	??	Sub-50 ms protection	M	Yes [ ]
PP3	Steering protection	??	Steering protection	M	Yes [ ]
PP4	Wrapping protection	??	Wrapping protection	O	Yes [ ] No [ ]
PP5	Revertive operation	??	Revertive operation by default	M	Yes [ ]
PP6	Non-revertive operation	??	Non-revertive operation may be supported	O	Yes [ ] No [ ]
PP7	Bidirectional protection	??	Stations at both ends of a span consider the span as an edge for both transmit and receive directions	M	Yes [ ]
PP8	WTR configuration	10.1.5.6	Range, resolution, and default supported	M	Yes [ ]
PP9	Hold-off time configuration	10.1.5.6	Range, resolution, and default supported	M	Yes [ ]
PP10	Hold-off timer operation	10.1.5.6	Hold-off timer starts after signal fail detected and elapses before signal fail triggers	M	Yes [ ]
PP11	Keepalives across edge	10.1.5.6	Keepalives are transmitted and received across edge	M	Yes [ ]
PP12	Keepalive timeout configuration	10.1.5.6	Range, resolution, and default supported	M	Yes [ ]
PP14	Protection status determination	10.1.5.6	Combination of LINK_STATUS from PHY and internal RPR MAC status	M	Yes [ ]
PP15	Failure of transit path	10.7	Station does not wrap on both spans	PP4:M	Yes [ ]

10.8.3.5 Topology validation

Item	Feature	Subclause	Value/Comment	Status	Support
TV1	??	??	??	??	??

10.8.3.6 TP frame handling

Item	Feature	Subclause	Value/Comment	Status	Support
TP1	*txFastDelay* value	10.2.3	Range, resolution, and default for the fast-rate transmissions	M	Yes [ ]
TP2	*txSlowDelay* value	10.2.3	Range, resolution, and default for the slow-rate transmissions	M	Yes [ ]
TP3	TransmitTpFrame state table	10.6.7.4	Transmission of TP frames	M	Yes [ ]
TP4	ReceiveTpFrame state table	10.6.8.4	Reception of TP frames	M	Yes [ ]

10.8.3.7 TC frame handling

Item	Feature	Subclause	Value/Comment	Status	Support
TC1		10.6.9.4	Transmission of TC frames	M	Yes [ ]
TC2		10.6.10.3	Reception of TC frames	M	Yes [ ]

10.8.3.8 ATD frame handling

Item	Feature	Subclause	Value/Comment	Status	Support
TC1		10.6.11.3	Transmission of ATD frames	M	Yes [ ]
TC2		10.6.12.3	Reception of ATD frames	M	Yes [ ]

10.8.3.9 Defect indications

Item	Feature	Subclause	Value/Comment	Status	Support
DF1	Defect support	10.2.9	Defects supported as specified in subclause	M	Yes [ ]

10.8.3.10 ATD frame handling

Item	Feature	Subclause	Value/Comment	Status	Support
ATD1	ATD timer	10.2.3	Range, resolution, and default supported	M	Yes [ ]
ATD2	Changed values of ATTs that use resources	10.5.2	Handled as specified	M	Yes [ ]
ATD3	Claiming ownership of resources through ATTs	10.5.2	No station claims ownership of resources if another station has claimed ownership	M	Yes [ ]
ATD4	Handling of race conditions	10.5.2	Stations all remove their claims and generate defect	M	Yes [ ]

10.8.3.11 LRTT measurements

Item	Feature	Subclause	Value/Comment	Status	Support
LRTT1	LRTT request processing	10.3.3	All stations process LRTT requests	M	Yes [ ]
LRTT2	LRTT response time	10.6.15	Response to LRTT request within 100 ms	M	Yes [ ]
LRTT3	No late LRTT response	10.6.15	If response not received within 100 ms, then response ignored	M	Yes [ ]
LRTT4	*lrttreDelay* value	10.6.15.2	Range, resolution, and defaults	M	Yes [ ]
LRTT5	LRTT re-measurement	10.6.15	Re-measurement done after at least 10 seconds	O	Yes [ ] No [ ]
LRTT6	TimingLrttFrame state table	10.6.15.4	State machine behavior	O	Yes [ ] No [ ]