

IEEE Draft PTBD.xx/D0.10

Resilient backplane ring (RBR)— Short haul extensions to resilient packet ring (RPR)

Sponsor

TBD
of the
IEEE Computer Society

This standard defines the physical layer (PHY) and frame-transport service extensions for RPR necessary to support the backplane environment.

Copyright © 2003 by the Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue
New York, NY 10016-5997, USA

All rights reserved. This document is an unapproved draft of a proposed IEEE Standard. As such, this document is subject to change. USE AT YOUR OWN RISK! Because this is an unapproved draft, this document must not be utilized for any conformance/compliance purposes. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only. Prior to submitting this document to another standards development organization for standardization activities, permission must first be obtained from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department. Other entities seeking permission to reproduce this document, in whole or in part, must obtain permission from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department.

IEEE Standards Activities Department
Standards Licensing and Contracts
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

List of Special Symbols

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

For the benefit of those who have received this document by electronic means, what follows is a list of special symbols that are produced using non-standard characters. If any of these symbols fail to print out correctly on your machine, the editors apologize, and hope that this table will at least help you to sort out the meaning of the resulting funny-shaped blobs and strokes. Note that **this table will be removed** in the final publication, and is only intended to assist during draft creation and editing. The reader is directed to Clause 3 for normative definitions of symbols and operators that have significance for this draft. Not all of the symbols herein may be used in the draft.

Special symbols and operators

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

Contents		1
		2
List of figures.....	v	3
		4
List of tables.....	vi	5
		6
1. Overview.....	23	7
		8
1.1 Scope and purpose.....	23	9
1.2 RBR topologies.....	24	10
1.2.1 MAC datapath components.....	25	11
1.3 RPR fairness.....	26	12
1.3.1 Service classes.....	26	13
1.3.2 Fairness frame distribution.....	26	14
1.3.3 Frame transmissions.....	27	15
1.3.4 Frame formats.....	28	16
1.3.5 Spatial reuse.....	28	17
1.3.6 Protection methods.....	29	18
1.4 Backplane wiring.....	30	19
1.5 Transactions.....	31	20
1.5.1 Transaction sequences.....	31	21
1.6 Destination back-pressure.....	32	22
1.6.1 Directed transmission retries.....	32	23
1.6.2 Broadcast transmission retries.....	32	24
1.6.3 Local sequence numbers.....	32	25
1.6.4 Load-dependent back-pressure.....	32	26
1.7 Request/response transactions.....	33	27
1.7.1 Request/response frame formats.....	33	28
1.7.2 Responder frame processing.....	34	29
1.7.3 Unaligned data padding.....	34	30
1.7.4 Transaction commands.....	35	31
1.7.5 Default transaction-request data values.....	37	32
1.8 Physical layer options.....	38	33
1.9 Time synchronization.....	39	34
1.9.1 System timer master signals.....	39	35
1.9.2 System time synchronization.....	40	36
1.9.3 Symmetric transfers.....	41	37
		38
2. Normative references.....	43	39
		40
3. Terms, definitions, and notation.....	45	41
		42
3.1 Conformance levels.....	45	43
3.2 Terms and definitions.....	45	44
3.3 State machines.....	50	45
3.3.1 State table notation.....	50	46
3.4 Arithmetic and logical operators.....	52	47
3.5 Numerical representation.....	52	48
3.6 Field notations.....	53	49
3.6.1 Use of italics.....	53	50
3.6.2 Field conventions.....	53	51
3.6.3 Field value conventions.....	54	52
3.7 Bit numbering and ordering.....	54	53
3.8 Byte sequential formats.....	55	54

List of figures

	1
	2
Figure 1.1—Physical topology comparisons.....	3
Figure 1.2—Dual-ring structure	4
Figure 1.3—MAC data paths	5
Figure 1.4—Fairness frames	6
Figure 1.5—Unicast frame transmissions	7
Figure 1.6—Flooded transmissions.....	8
Figure 1.7—Basic data frame format	9
Figure 1.8—Concurrent data transfers	10
Figure 1.9—Protection mechanisms	11
Figure 1.10—Logical backplane wiring.....	12
Figure 1.11—Interleaved backplane wiring.....	13
Figure 1.12—Partially populated backplane wiring.....	14
Figure 1.13—Transaction sequences.....	15
Figure 1.14—Destination back pressure	16
Figure 1.15—Request/response frame formats	17
Figure 1.16—Responder frame processing	18
Figure 1.17—Unaligned data padding	19
Figure 1.18—Default update values.....	20
Figure 1.19—Physical layer options	21
Figure 1.20—Timer-reference flows.....	22
Figure 1.21—Loop-back timer-frame flows	23
Figure 1.22—Clock synchronization.....	24
Figure 1.23—Symmetric timer synchronization	25
Figure 3.1—Bit numbering and ordering	26
Figure 3.2—Byte sequential field format illustrations	27
Figure 3.3—Multi-field illustrations	28
Figure 3.4—Ringlelet orientation conventions	29
	30
	31
	32
	33
	34
	35
	36
	37
	38
	39
	40
	41
	42
	43
	44
	45
	46
	47
	48
	49
	50
	51
	52
	53
	54

Resilient backplane ring (RBR)— Short haul extensions to resilient packet ring (RPR)

1. Overview

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

Revision History:

Draft 0.01, December 2003

Initial draft document for study group review.

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

RPR supports features desired for backplane applications, including quality-of-service (QOS), plug-and-play, fairness, and concurrency. To utilize these desirable properties, this standard extends the baseline RPR protocols to support board-to-board communications on a telecommunications backplane.

1.1 Scope and purpose

This document supplements Part 17 of IEEE Std. 802, extending the RPR medium access control (MAC) and physical layer (PHY) specifications with features desired for backplane applications.

Scope: Supplement RPR protocols with features defined to efficiently support backplane environments. These include a cost-effective physical layer interface (PHY), extended frame-transfer protocols with memory-mapped transaction support, flow-control based on local destination backpressure, and accurate clock synchronization capabilities.

Purpose: To support efficient cost-effective transfer of Ethernet and memory-mapped access packets within the backplane environment.

A small amount of informative RPR content is replicated within this standard, for the convenience of the reader. In cases of conflict, the RPR standard has precedence.

1.2 RBR topologies

The RBR and RPR protocols assume the same counter-rotating ringlet structure, as illustrated in Figure 1.1. The resilient packet ring (RPR) was optimized for up to 256 stations, communicating through serial optical links, with a large 2,000 kM metropolitan circumference, as illustrated in Figure 1.1-a. The resilient backplane ring (RBR) is optimized for up to 64 stations, communicating through parallel copper links, with a short 2 M backplane circumference, as illustrated in Figure 1.1-b.

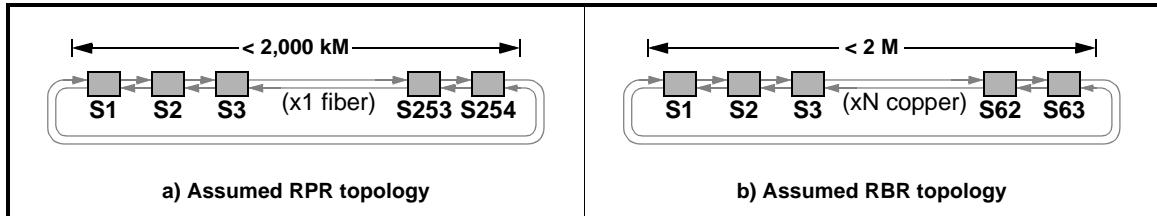


Figure 1.1—Physical topology comparisons

RPR and RPR employs a ring structure using unidirectional, counter-rotating ringlets. Each ringlet is made up of links with data flow in the same direction. The ringlets are identified as ringlet0 and ringlet1, as shown in Figure 1.2.

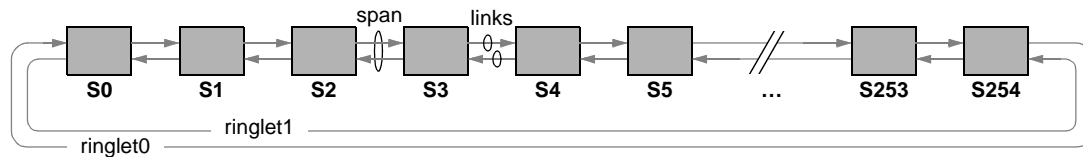


Figure 1.2—Dual-ring structure

Stations on the ring are identified by an IEEE 802 48-bit MAC address as specified in IEEE Std 802-2002, “IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture.” All links on the ring operate at the same data rate, but may exhibit different delay properties.

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

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

1.2.1 MAC datapath components

The RBR/RPR MAC datapaths provide check logic that determines which frames are stripped, and transit queues that hold received frames waiting to be transmitted. Stations have a primary transit queue (PTQ), typically only a few MTUs in size, and have the option of providing another typically much larger secondary transit queue (STQ), as illustrated in Figure 1.3-a and Figure 1.3-b, respectively.

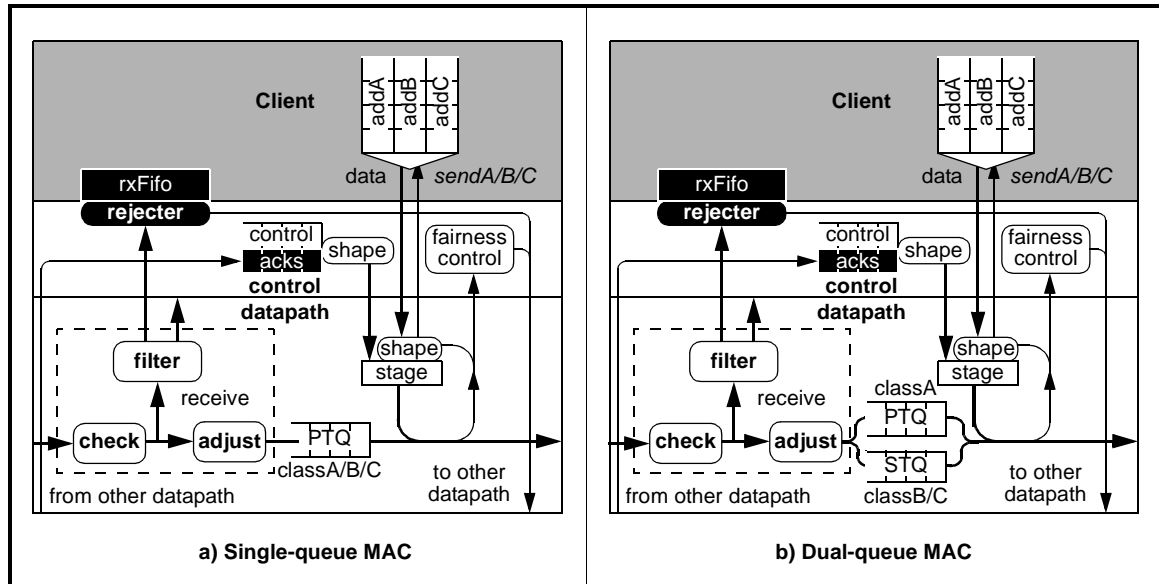


Figure 1.3—MAC data paths

Frames are transmitted from a MAC-resident *stage* queue, rather than transmitted directly from the client, to decouple MAC-to-client interface timings from the timings of the physical-layer interface. All traffic is shaped before transmission, so that bandwidths can be guaranteed regardless of other stations' loads and/or physical placements.

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

RBR (as opposed to RPR) assumes the presence of additional destination-receive flow control protocols. The distinct shaded-black components support robust congestion-insensitive request/response transaction-based communications. Under congestion conditions, the rejecter is responsible for sensing the *rxFifo* filling condition and (after threshold levels are reached) determining which transaction-related frames are discarded. Whether saved or discarded, all such frames are acknowledged on the opposing ringlet.

Additional logic (not illustrated) processes received transaction-related ack frames. Based on these returned frames, pending subaction transmissions are either completed, retried, or (after excessive delay timeouts) are discarded.

1 **1.3 RPR fairness**

2
3 **1.3.1 Service classes**

4
5 The MAC data primitive identifies a service class (A, B, or C) with which the data transfer is associated, as
6 summarized in Table 1.1. The classA service provides low-jitter transfer of traffic (and therefore lower
7 worst-case delays) up to its allocated rate. The classB service provides bounded delay transfer of traffic. The
8 classC service provides best-effort data-transfer services.
9

10 **Table 1.1—Service classes and their quality-of-service relationships**

11
12

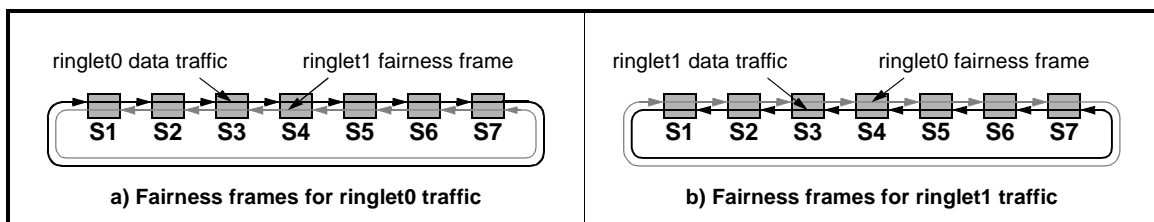
Class of service	Qualities of service		
	bandwidth	jitter	type
A	guaranteed	low	allocated
B	guaranteed	bounded	allocated
C	no guarantee	unbounded	opportunistic

13
14
15
16
17
18
19
20
21
22

23 Ring capacity required to support the classA service and classB service is allocated via provisioning and
24 these services can be characterized as allocated services. The provisioning activity is expected to ensure that
25 the aggregate service commitment on each link does not exceed that link’s capacity.
26

27 **1.3.2 Fairness frame distribution**

28
29 Shaping parameters for fairness eligible traffic are computed using a distributed fairness algorithm. The
30 fairness algorithm relies on fairness frames that are circulated periodically on the ringlet opposing that of the
31 associated data traffic, as illustrated in Figure 1.4. The basic fairness frame is processed at each station and
32 carries the identity of the station that is the most congested due to this station's transmissions, and the
33 congestion state associated with that station.
34
35



43 **Figure 1.4—Fairness frames**

1.3.3 Frame transmissions

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

1.3.3.1 Unicast transmissions

A local unicast frame is transmitted on one of the ringlets, as illustrated in Figure 1.5-a. A unicast frame is added by the source station and (for efficiency) is normally stripped at the destination station. Within Figure 1.5, and other data-flow illustrations, the end-of-flow curves identify where the frames are stripped.

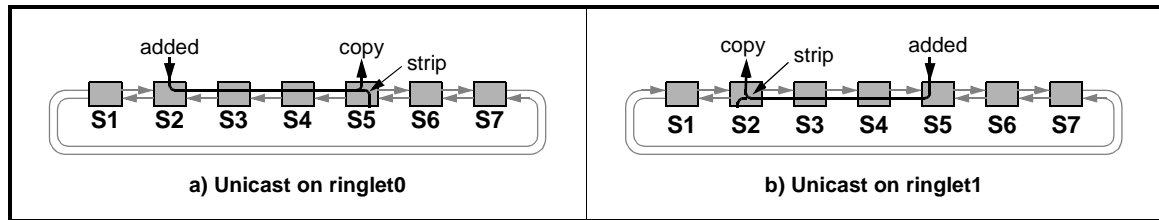


Figure 1.5—Unicast frame transmissions

If ringlet selection is based on shortest hop-count, a returning frame is likely to take an opposing run path, as illustrated in Figure 1.5-b.

1.3.3.2 Flooded transmissions

Transparent bridging relies on the ability to flood unknown-unicast frames to all bridges, to ensure that the frame can be delivered to all the segments within the bridged LAN.

Unidirectional flooding involves sending the frame to all other stations in the ring via either ringlet0 (as illustrated in Figure 1.6-a) or ringlet1 (not illustrated). Bidirectional flooding involves sending frames to all other stations in the ring via both ringlet0 and ringlet1, as illustrated in Figure 1.6-b.

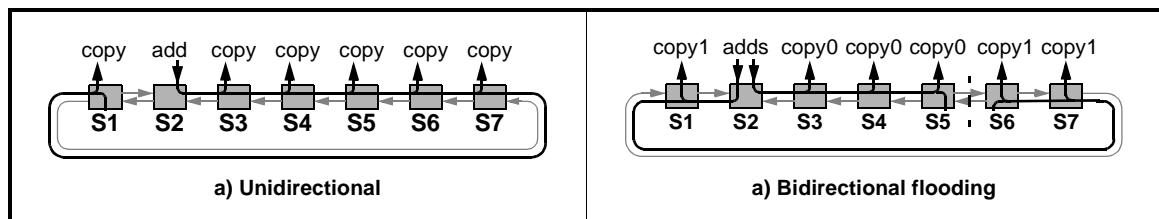


Figure 1.6—Flooded transmissions

The selected fi (flooding indication) field value within the *baseControl* field of a frame (see 1.3.4) determines whether the frame is to be not-flooded, unidirectionally flooded, or bidirectionally flooded.

1.3.4 Frame formats

The fields within the basic data frame (see Clause 5) are illustrated in Figure 1.7. End-to-end parameters are supplemented with additional information necessary to transmit the frame over the RBR/RPR interconnect (see 5.2 for details).

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

Figure 1.7—Basic data frame format

1.3.5 Spatial reuse

The adding of individual frames is not synchronized between ringlets and the frame transmission event on any one link is independent of frame transmission events on other links. With the RBR/RPR ring topology, this allows per-link bandwidths to be utilized beyond that possible with other ring based LAN technologies such as IEEE Std 802.5-1998 Token Ring or ANSI FDDI based protocols, as illustrated in Figure 1.8.

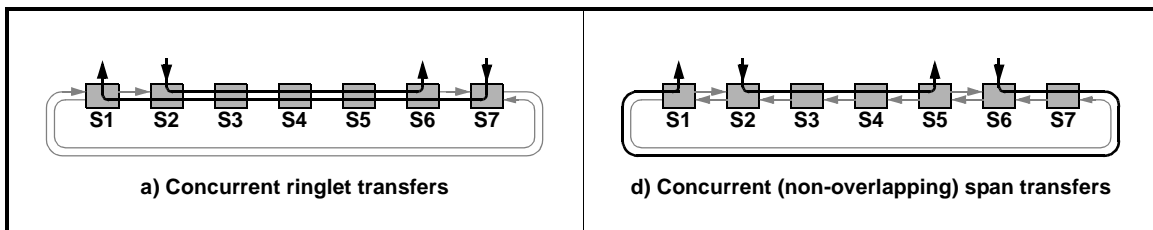


Figure 1.8—Concurrent data transfers

By supporting concurrent per-ringlet transmissions (Figure 1.8-a), the cumulative bandwidths for stations on that ringlet exceed the capacity of any individual link. The effective levels of allocated bandwidths (Figure 1.8-b) are similarly improved, since allocated bandwidths can be independently committed on non-overlapping segments.

1.3.6 Protection methods

The dual-ring topology ensures that an alternate path between source station and destination station(s) is available following the failure of a single span or station. Protection involves either steering or wrapping.

Steering is supported by all stations. In a ring using steering protection, traffic begins being discarded at stations adjacent to the point of failure. To avoid continued discards, source stations initiate steering of traffic by redirecting unicast traffic to the other ringlets that retain connectivity to destinations beyond the point of failure. Multicast and broadcast traffic is normally directed to both ringlets, so as to reach all stations on the ring.

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

The effect of a station-detected failure depends on the failure condition and the state of other stations. The failure can become visible as a station-removal (Figures 1.9-a and 1.9-b) or span-removal change to the operational topology (Figure 1.9-c and 1.9-d), as illustrated in Figure 1.9. Isolation can involve discarding endpoint traffic and steering traffic so that the endpoint is never reached (Figure 1.9-c) or wrapping traffic at the edges (Figure 1.9-d).

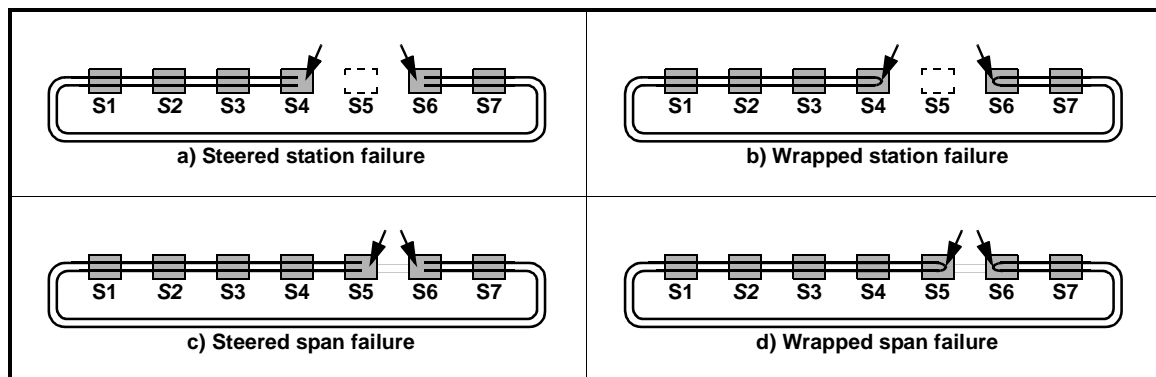


Figure 1.9—Protection mechanisms

1.4 Backplane wiring

From a logical perspective, an RPR backplane consists of counter-rotating ringlets, as illustrated in Figure 1.10. The counter-rotating ringlets provide resilience, improve throughput by allowing frames (from multiple stations) to be sent concurrently, and reduce latency by allowing each frame (from a station) to be sent in the shortest direction.

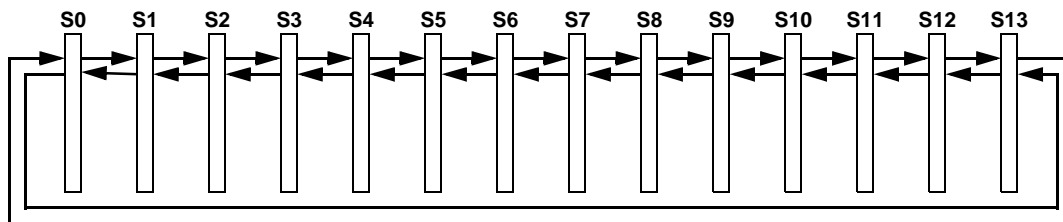


Figure 1.10—Logical backplane wiring

The logical backplane wiring of Figure 1.10 suffers from the requirement to drive long-distance wrap-around paths between station S0 and station S15 cards. To overcome this limitation, card slots are expected to be interleaved, as illustrated in Figure 1.11. Such backplane layouts have near equal-distant card-to-card backplane wiring distances.

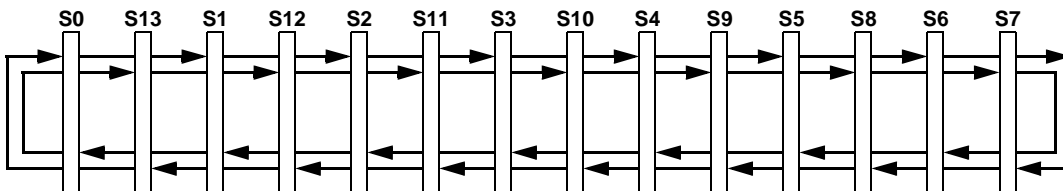


Figure 1.11—Interleaved backplane wiring

To support partially populated backplanes, additional backplane traces and a termination card are required. For example, the station S4 termination card allows stations S0-to-S3 and S10-to-S13 to be interconnected, as illustrated in Figure 1.12.

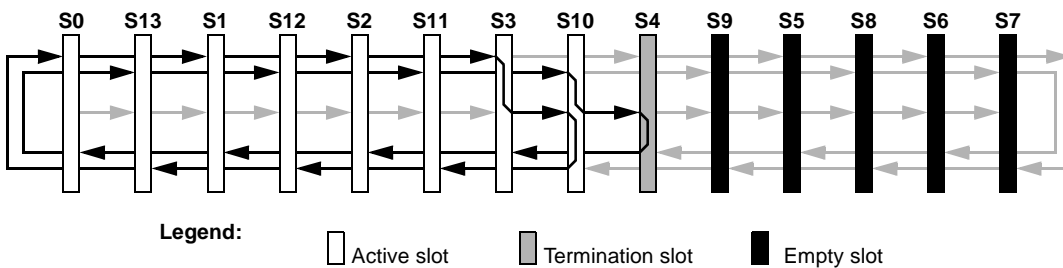


Figure 1.12—Partially populated backplane wiring

With such connectivity, the RBR backplane can survive any single-card failure or removal.

1.5 Transactions

1.5.1 Transaction sequences

Transactions provide a robust mechanism for communicating between stations. Transactions are initiated by a *requester* and completed by a *responder*. Transactions consist of two subactions. During the *request subaction* address and command are transferred from the requester to the responder. The *response subaction* returns completion status from the responder to the requester. Depending on the transaction command, data are transferred in the request subaction (writes), the response subaction (reads) or both subactions (locks).

A subaction consists of two frame transmissions, sent between sender and target stations. A subaction is initiated by the sender, which generates a send frame. The subaction is completed by the target, which returns an ack frame. Hence, a typical transaction involves the transfer of four frames, as illustrated in Figure 1.13.

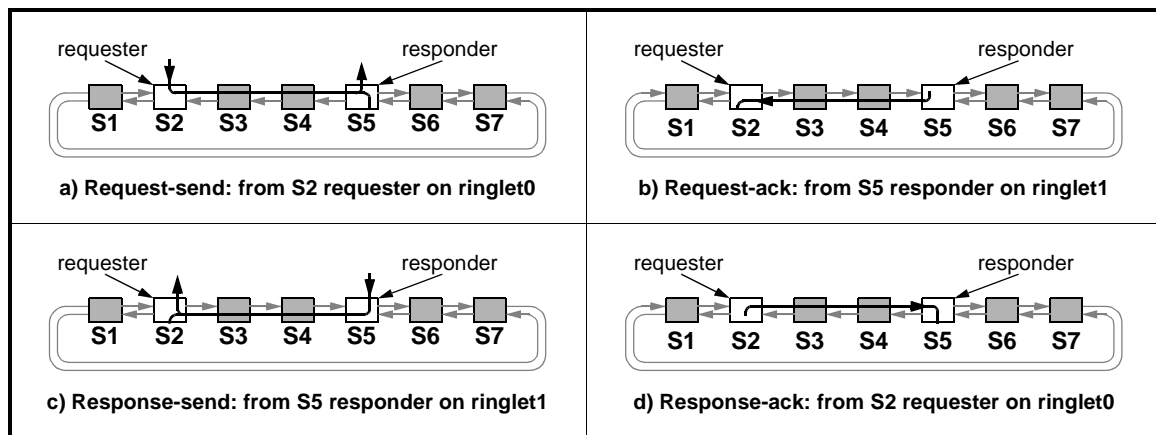


Figure 1.13—Transaction sequences

While individual acks are nominally supplied, a congested target may drop all but the last ack frame. The ack's returned status and index values confirm (or deny) the successful acceptance of preceding subactions.

1.6 Destination back-pressure

1.6.1 Directed transmission retries

TBD

1.6.2 Broadcast transmission retries

TBD

1.6.3 Local sequence numbers

TBD

1.6.4 Load-dependent back-pressure

The traditional problem with busy-retry forms of flow control is the overhead of constant busy-retry polling traffic. To reduce such overheads, the destination is expected to rate-limit the ack frame release rates, as illustrated in Figure 1.14.

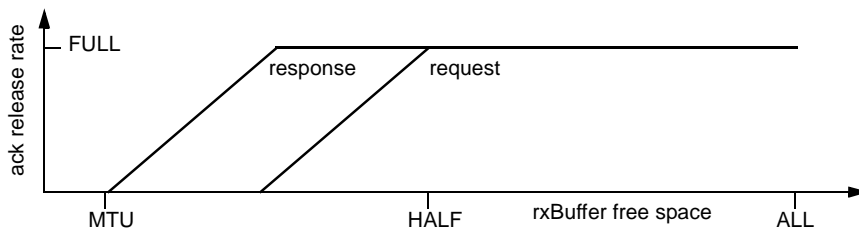


Figure 1.14—Destination back pressure

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

1.7 Request/response transactions

Request/response transactions are efficient backplane operations which are generated by the requester to invoke standardized memory-mapped actions in the requester. As examples, variable-length read and write transactions are commonly supported on most computer backplanes, allowing memory (or memory-mapped hardware) to be shared between general or special purpose multiprocessors.

To simplify their implementation, standard request/response transactions are based on variants of the write (which transfers data to the memory) and read (which transfers data from memory). Four variations of the read/write like transactions allow data to be transferred in either or both directions: default-to-memory with nothing returned (clear), default-to-memory with previous data returned (read or increment), data-to-memory with nothing returned (write), and data-to-memory with previous data returned (swap). These are discussed in the following subclauses.

1.7.1 Request/response frame formats

The fields within the basic RPR and extended RBR request/response data frames are illustrated in Figure 1.15. The *pacing* field takes the place of RPR's normal *da* field; other common RBR/RPR defined fields are shaded gray.

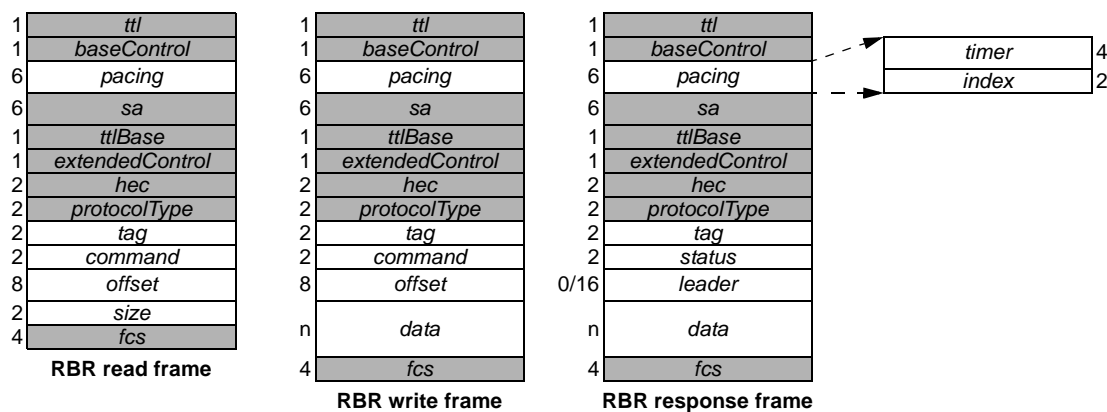


Figure 1.15—Request/response frame formats

The 32-bit *timer* specifies the frame's creation time, so that hopelessly delayed frames can be dropped. The *timer* value also allows the oldest of the busied frames to be successfully accepted, thus ensuring forward progress at congested destination stations.

The 16-bit *index* field distinctively identifies sequential subactions associated with the same destination station, for the purposes of supporting in-order fault-tolerant busy-retried subaction transfers.

A distinct *protocolType* field distinguishes the request/response data frames from other transmitted frames.

The 16-bit *tag* field distinctively identifies outstanding transactions associated with the requester station.

In request frames, the 16-bit *command* specifies the request command; the *offset* field specifies the address of a location within the responding station.

In response frames, the 16-bit *status* supplies the response completion status; the *leader* field provides optional extended-status (typically consisting of a pointer to a distinct memory-mapped location).

1.7.2 Responder frame processing

The transaction model assumes the request is addressed to memory on the responder, as illustrated in Figure 1.16. The *offset* is mapped to an address in the station's memory. The command specifies the operations that are performed on successive 64-bit aligned words of *requestData* and memory. A variety of two-input (request payload and memory content) update operations is supported, as specified in 1.7.4.

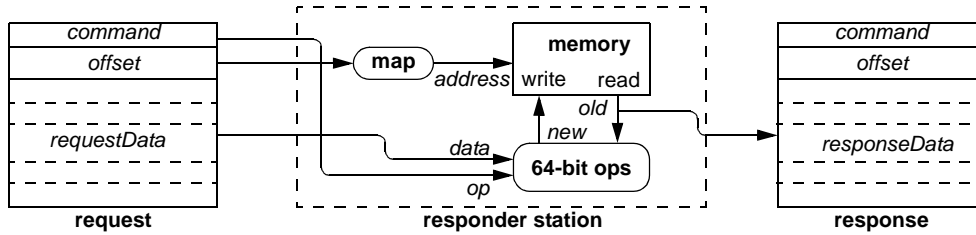


Figure 1.16—Responder frame processing

1.7.3 Unaligned data padding

All memory operations are defined as sequences of aligned 64-bit operations. When accessing unaligned addresses, data is zero-filled to generate an integer number of 64-bit request-parameter and memory-parameter operands, as illustrated in Figure 1.17. When returning unaligned data, the unaffected memory addresses are not changed nor is their content returned in the response frame.

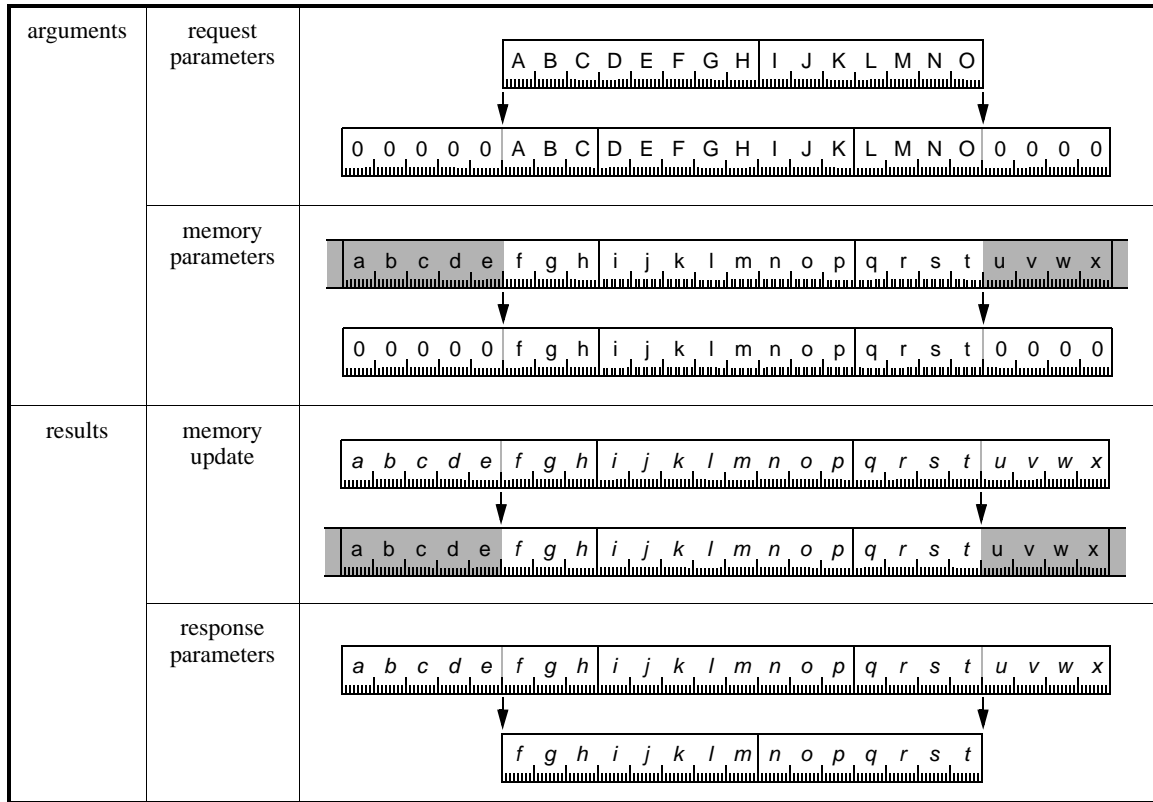


Figure 1.17—Unaligned data padding

This processing of unaligned addresses maintains much of the hardware simplicity associated with uniform 64-bit aligned updates, while facilitating non-vector updates of smaller (32-bit or less) scalar values.

1.7.4 Transaction commands

Transactions include logical and arithmetic operations that can be performed as vector operations on 64-bit quantities, as specified in Table 1.2.

Table 1.2—Transaction commands

Name		Row	data payload		Algorithmic operation
Casual	Formal		request	response	
Fence	NULL_READ_VOID	1	(size)	—	if (data != NULL) value[address] = data;
Read	NULL_READ_PAST	2	(size)	data	
Write	DATA_MODS_VOID	3	data	—	
Swap	DATA_MODS_PAST	4	data	data	
—	—	5	—	—	// Reserved
—	DATA_TEST_VOID	6	data	—	if (value == 0) value = data;
NullSwap	DATA_TEST_PAST	7	data	data	
—	DATA_ZAPS_VOID	8	data	—	if (value == data) value = zero;
TestClear	DATA_ZAPS_PAST	9	data	data	
Increment	LSBS_ADDS_VOID	10	(size) ^a	—	value[address] += data;
Fetch&Increment	LSBS_ADDS_PAST	11	(size) ^a	data	
Add	DATA_ADDS_VOID	12	data	—	
Fetch&Add	DATA_ADDS_PAST	13	data	data	
Decrement	LSBS_SUBS_VOID	14	(size) ^a	—	if (value >= data) value -= data;
Fetch&Decrement	LSBS_SUBS_PAST	15	(size) ^a	data	
Subtract	DATA_SUBS_VOID	16	data	—	
Fetch&Subtract	DATA_SUBS_PAST	17	data	data	
Clear	ONES_NAND_VOID	18	(size) ^b	—	value &= ~data;
Load&Clear	ONES_NAND_PAST	19	(size) ^b	data	
ClearBits	DATA_NAND_VOID	20	data	—	
Fetch&ClearBits	DATA_NAND_PAST	21	data	data	
Set	ONES_SETS_VOID	22	(size) ^b	—	value = data;
Load&Set	ONES_SETS_PAST	23	(size) ^b	data	
SetBits	DATA_SETS_VOID	24	data	—	
Fetch&SetBits	DATA_SETS_PAST	25	data	data	

^aA default value of *lsbOnly* is used; see 1.7.5 for details.

^bA default value of *oneBits* is used; see 1.7.5 for details.

Row 1.2-1: The NULL_READ_VOID (Fence) transaction leaves memory unchanged and returns no data.	1
A typical use would be to confirm the completion of the preceding transaction.	2
Row 1.2-2: The NULL_READ_PAST (Read) transaction returns the current memory contents.	3
A typical use would be to provide instruction or data values to executing hardware or software programs.	4
Row 1.2-3: The NULL_MODS_VOID (Write) transaction copies request-supplied data into memory.	5
A typical use would be to save new data values generated by executing hardware or software programs.	6
Row 1.2-4: The NULL_MODS_PAST (Swap) transaction copies request-supplied data into memory, and returns the previous memory values as response-supplied data.	7
A typical use would be to implement simple singly-linked-list update operations.	8
	9
Row 1.2-5: One memory-update command is reserved.	10
	11
	12
Row 1.2-6: The DATA_TEST_VOID transaction changes zero memory values to the request-supplied data.	13
Row 1.2-7: The DATA_TEST_PAST (NullSwap) transaction changes zero memory values to the request-supplied data, and returns the previous memory values as response-supplied data.	14
A typical use would be to access a semaphore, when obtaining mutually exclusive access to shared data.	15
	16
	17
Row 1.2-8: The DATA_ZAPS_VOID transaction clears matching memory values.	18
Row 1.2-9: The DATA_ZAPS_PAST (TestClear) transaction clears matching memory values, and returns the previous memory values as response-supplied data.	19
A typical use would be to access a semaphore, when releasing mutually exclusive access to shared data.	20
	21
	22
Row 1.2-10: The LSBS_ADDS_VOID (Increment) transaction increments memory values.	23
A typical use would be to update shared action-accomplished counts.	24
Row 1.2-11: The LSBS_ADDS_PAST (Fetch&Increment) transaction increments memory values, and returns the previous memory values as response-supplied data.	25
A typical use would be to update a completion-count, before initiating completion-dependent activities.	26
Row 1.2-12: The DATA_ADDS_VOID (Addition) transaction adds request-supplied data to memory.	27
A typical use would be to update statistics counters, based on the volume of processed data.	28
Row 1.2-13: The DATA_ADDS_PAST (Fetch&Add) transaction adds request-supplied data to memory, and returns the previous memory values as response-supplied data.	29
A typical use would be to update statistics counters, while checking for overflow conditions.	30
	31
	32
	33
Row 1.2-14: The LSBS_SUBS_VOID (Decrement) transaction decrements memory values until zero.	34
Row 1.2-15: The LSBS_SUBS_PAST (Fetch&Decrement) transaction decrements memory values until zero, and returns the previous memory values as response-supplied data.	35
A typical use would be to access resource-available counts, before starting a one-count dependent task.	36
Row 1.2-16: The DATA_SUBS_VOID (Subtract) transaction subtracts request-supplied data from memory.	37
Row 1.2-17: The DATA_SUBS_PAST (Fetch&Subtract) transaction subtracts request-supplied data from memory, and returns the previous memory values as response-supplied data.	38
A typical use would be to update resource-available counts, before starting a multi-count dependent task.	39
	40
	41
	42
Row 1.2-18: The ONES_NAND_VOID (Clear) transaction clears memory values to zero.	43
A typical use would be to efficiently initialize unknown or intended-to-be-secure memory regions.	44
Row 1.2-19: The ONES_NAND_PAST (Fetch&Clear) transaction clears memory values to zero, and returns the previous memory values as response-supplied data.	45
A typical use would be to clear memory for new activities, before saving the results of preceding actions.	46
Row 1.2-20: The DATA_NAND_VOID (ClearBits) transaction clears memory-bit values to zero.	47
A typical use would be to maintain recently-used indications, for replacement-selection purposes.	48
Row 1.2-21: The DATA_NAND_PAST (Fetch&ClearBits) transaction clears memory-bit values to zero, and returns the previous memory values as response-supplied data.	49
A typical use would be when servicing recently-used indications, for replacement-selection maintenance.	50
	51
	52
	53
	54

- Row 1.2-22: The ONES_SETS_VOID (Set) transaction sets memory values to all ones. 1
- A typical use would be to efficiently initialize unknown or intended-to-be-secure memory regions. 2
- Row 1.2-23: The ONES_SETS_PAST (Fetch&Set) transaction sets memory values to all ones, and returns 3
- the previous memory values as response-supplied data. 4
- A typical use would be to initialize memory for new activities, before saving the results of preceding actions. 5
- Row 1.2-24: The DATA_SETS_VOID (SetBits) transaction sets memory-bit values to one. 6
- A typical use would be to maintain of recently-used indications, for replacement-selection purposes. 7
- Row 1.2-25: The DATA_SETS_PAST (Fetch&SetBits) transaction clears selected memory bits to one, and 8
- returns the previous memory values as response-supplied data. 9
- A typical use would be when servicing recently-used indications, for replacement-selection maintenance. 10

1.7.5 Default transaction-request data values

Implementation of these transactions assumes the availability of two default values, as illustrated in Figure 1.18. The *lsbOnly* default value is used by the arithmetic updates, thereby eliminating the need to transfer data values when only increment/decrement operations are performed. The *allOnes* default value is used by the logical updates, thereby eliminating the need to transfer data values when all bits are affected.

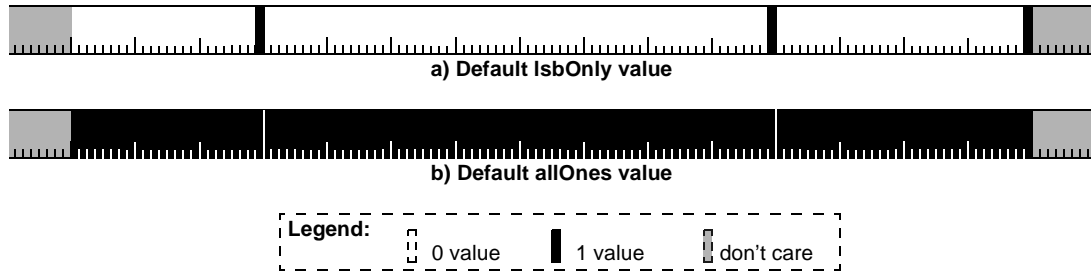


Figure 1.18—Default update values

1.8 Physical layer options

A variety of high-speed board-to-board communication options are possible, as illustrated in Figure 1.19. The properties of these PHY layer options are summarized, although the PHY selection has been deferred.

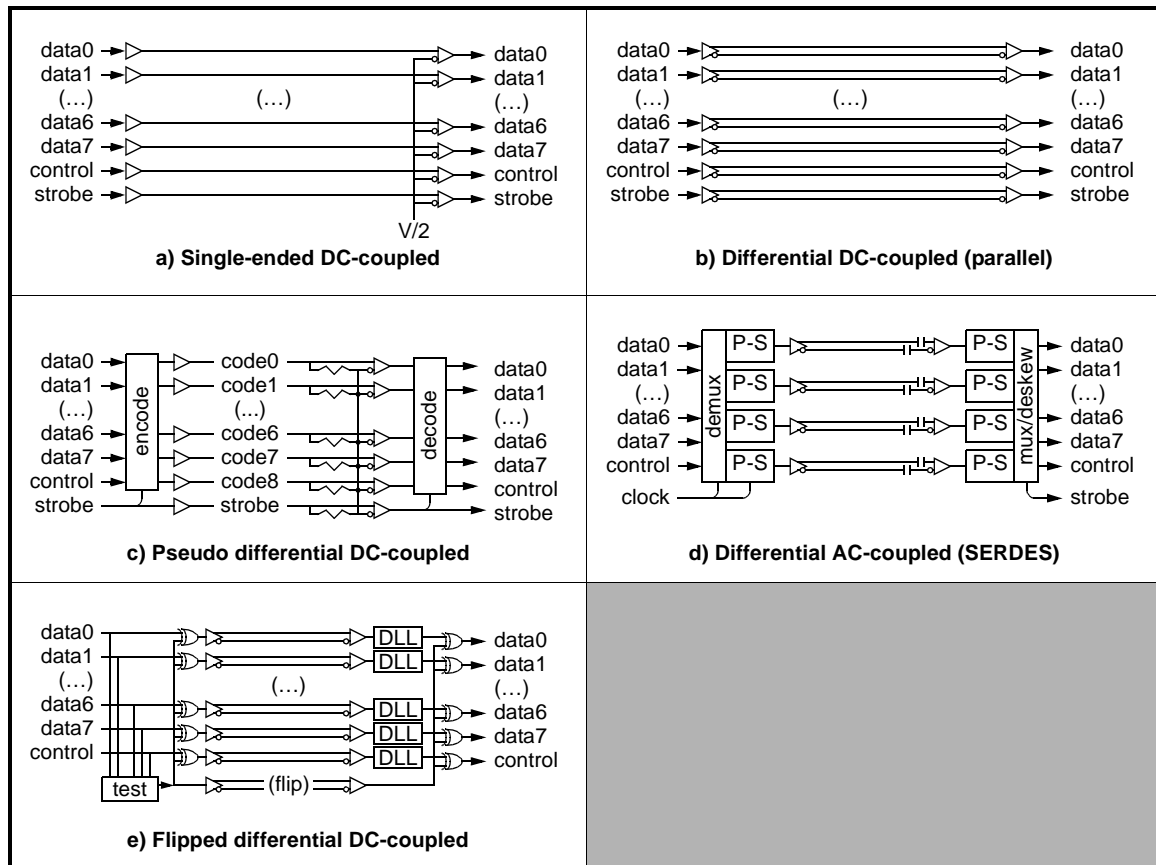


Figure 1.19—Physical layer options

The standard single-ended design of Figure 1.19-a relies on the presence of a $V/2$ voltage reference at the receiver. Noise or board-to-board differences in the ground potential result in $V/2$ jitter, which directly affect receive signal sensitivity and skew.

The standard differential design of Figure 1.19-b eliminates the requirement for a $V/2$ voltage reference, but relies on minimal skew. The elimination of the $V/2$ reference is not without cost: twice as many signal traces are required.

The pseudo-differential scheme of Figure 1.19-c reduces the pin-count and power requirements of Figure 1.19-b, with moderate impact on signal noise sensitivities. Unfortunately, these parallel schemes (Figure 1.19-a through Figure 1.19-c) rely on small (fraction of a bit period) line-to-line skews. Such skews are hard to sustain at higher frequencies, where differential-trace-length delays approach or exceed the bit period intervals.

For long-distance site-to-site transmissions, each byte is serialized and 8/10 encoded, as illustrated in Figure 1.19-d. For simple backplane solutions, the AC-coupling advantages are minimal. However, the

increased silicon area, power dissipation, latency, and high-speed 4-to-5 and 5-to-4 clock-rate conversion circuits are typically required.

For short-distance DC-coupled transmissions, bit transitions can be forced once in every 10 bits, as illustrated in Figure 1.19-e. By selectively complementing the transmissions, a bit[0] transition is forced in cycle[N+0], a bit[1] transition is forced in cycle[N+1], and (in general) a bit[n] transition is forced in cycle[N+(n%10)]. This supports efficient deskew circuits, while eliminating most of the encode/decode latency and eliminating the need for rate-conversion circuits.

1.9 Time synchronization

1.9.1 System timer master signals

Timer synchronization involves the distribution of a time reference from a system timing master to one or more system timing slaves. Unidirectional timing synchronization involves synchronization of stations in the direction of either ringlet0 (as illustrated in Figure 1.20-a) or ringlet1 (not illustrated). Bidirectional flooding involves synchronization of stations in both ringlet0 and ringlet1 directions, as illustrated in Figure 1.20-b. In either case, timer synchronization involves reference communications between the system timing master and associated timing slaves.

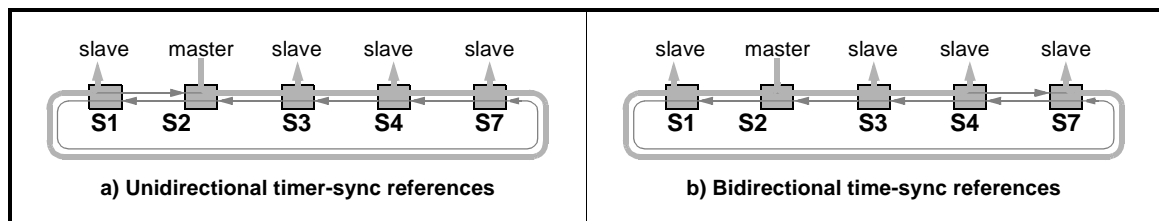


Figure 1.20—Timer-reference flows

To best compensate for symmetrical transceiver and transceiver delays, clock synchronization involves link-local loop-back communications between adjacent pairs of link-master and link-slave ports, as illustrated in Figure 1.21. Rather than transmitting frame at specific times (a difficult and bandwidth-constraining implementation alternative), timer synchronization is based on precisely measuring timeSync frame ingress and egress times, and then communicating this information in a pipelined one-cycle-delayed fashion..

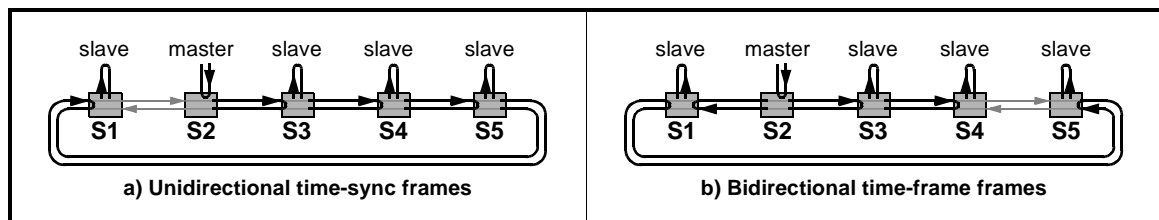


Figure 1.21—Loop-back timer-frame flows

1.9.2 System time synchronization

System time synchronization is performed in a sequential fashion: station B synchronizes to client A, station C synchronizes to station B, station D synchronizes to station C, and client E synchronizes to station D, as illustrated in Figure 1.22. Neighbor synchronization involves sending a clockSync request and returning a clockSync response. Each station is responsible for latching the times when clockSync frames pass across its boundaries.

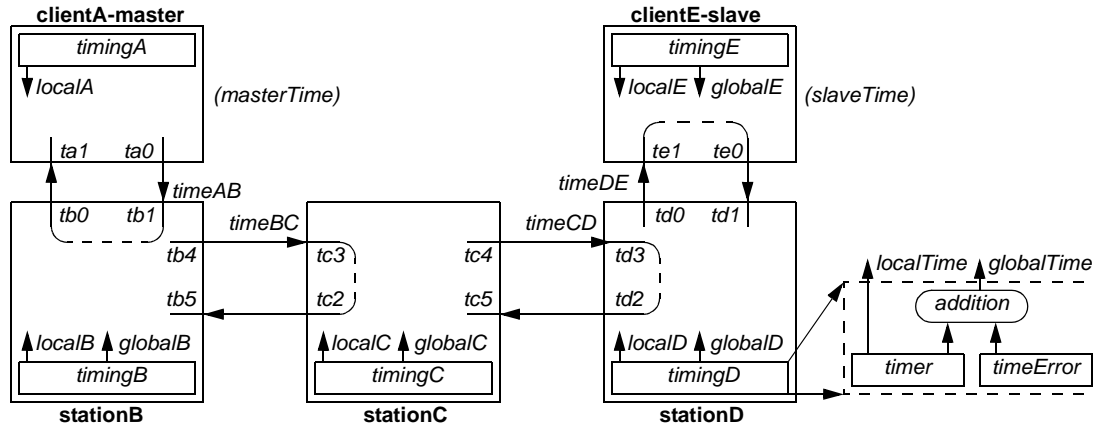


Figure 1.22—Clock synchronization

Synchronization of stationB to clientA proceeds as follows. In cycle[n], clientA sends a clockSync frame to stationB; its departure and arrival times are latched as local times *ta0* and *ta1* respectively. StationB quickly returns a clockSync response; its departure and arrival times are latched as local times *tb0* and *tb1* respectively. For numerical stability, the local free-running timer (not the global synchronized timer) is latched at the aforementioned times.

After the arrival of the n'th timeSync frame, clientA is responsible for computing the mid-point time between the previous *ta0* transmission and *ta1* reception (call this *timeLoopA*). In a similar fashion, clientA is responsible for computing the preceding time between the *ta1* reception and *ta1* transmission (call this *timeThruA*), as illustrated by the C code of Equation 1.1.

$$\begin{aligned} \text{timeLoopA}[n] &= (\text{ta0}[n] + \text{ta1}[n])/2; \\ \text{timeThruA}[n] &= (\text{ta0}[n] - \text{ta1}[n-1])/2; \end{aligned} \quad (1.1)$$

$$\begin{aligned} \text{timeBError}[n] &= \text{timeLoopA}[n] - (\text{tb0}[n] + \text{tb1}[n-1])/2; \\ \text{timeBDelay}[n] &= (\text{tb1}[n-1] - \text{tb0}[n-1])/2 - \text{timeThruA}[n]; \end{aligned} \quad (1.2)$$

After the arrival of this time-sync resident information from stationA, stationB has sufficient information to compute synchronized time and the average cable propagation from clientA, as illustrated by the C code of Equation 1.2. The mid-point time measured in stationB is the average of the previous *tb1* arrival and *tb0* transmission times; the timer error is the difference between the stationB and clientA computations, as specified by the first line of Equation 1.2. In a similar fashion, computation of cable delays involves measuring the just-measured difference between the *tb0* transmission and *tb1* reception, minus the propagation time through stationB, as specified by the second line of Equation 1.2.

At the clock slave port, the timer deviation measurement (in this case *timeBError*) is saved in a hardware register, allowing the synchronized *globalTime* to be derived from the free-running *localTime* value. The value of the measured cable delay (in this case *timeBDelay*) is saved in the topology database, allowing the cumulative station-to-station cable delays, between any two stations *n* and *m*, to be easily derived.

In a similar fashion, stationC is synchronized to stationB, stationD is synchronized to stationC, and clientE is synchronized to stationD, as specified by the C code of Equations 1.3, 1.4, and 1.5. Although each link is expected to send clockSync frames at similar rates, there is no transmit-synchronization requirement.

$$\begin{aligned} \text{timeLoopB}[n] &= (\text{tb4}[n] + \text{tb5}[n])/2; & (1.3) \\ \text{timeThruB}[n] &= (\text{tb4}[n] - \text{tb5}[n-1])/2; \\ \text{timeCError}[n] &= \text{timeLoopB}[n] - (\text{tc2}[n] + \text{tc3}[n-1])/2; \\ \text{timeCDelay}[n] &= (\text{tc3}[n-1] - \text{tc2}[n-1])/2 - \text{timeThruB}[n]; \end{aligned}$$

$$\begin{aligned} \text{timeLoopC}[n] &= (\text{tc4}[n] + \text{tc5}[n])/2; & (1.4) \\ \text{timeThruC}[n] &= (\text{tc4}[n] - \text{tc5}[n-1])/2; \\ \text{timeDError}[n] &= \text{timeLoopC}[n] - (\text{tc2}[n] + \text{tc3}[n-1])/2; \\ \text{timeDDelay}[n] &= (\text{td3}[n-1] - \text{td2}[n-1])/2 - \text{timeThruC}[n]; \end{aligned}$$

$$\begin{aligned} \text{timeLoopD}[n] &= (\text{td0}[n] + \text{td1}[n])/2; & (1.5) \\ \text{timeThruD}[n] &= (\text{td0}[n] - \text{td1}[n-1])/2; \\ \text{timeEError}[n] &= \text{timeLoopD}[n] - (\text{te0}[n] + \text{te1}[n-1])/2; \\ \text{timeEDelay}[n] &= (\text{te1}[n-1] - \text{te0}[n-1])/2 - \text{timeThruD}[n]; \end{aligned}$$

NOTE—Implementations can use phase-locked-loop (PLL) circuits to further reduce the clock jitter in their global time reference.

1.9.3 Symmetric transfers

Although previously shown as directional time-synch flows, the time-master/time-slave synchronization communications are actually symmetrical and each station is provided with knowledge of its neighbor's timer reference, as illustrated in Figure 1.23. The clock-master/clock-slave setting of a station affects the updating of its timing error register, but has no direct effect on each station's time-synchronization communications with its neighbors.

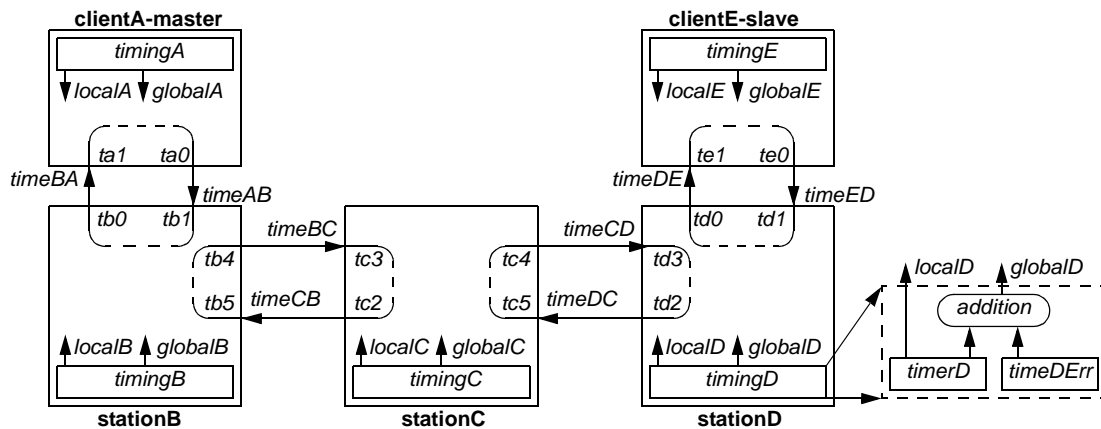


Figure 1.23—Symmetric timer synchronization

1 The boundary-crossing updates are thus specified by the C code of Equation 1.6 and Equation 1.7. The
2 much slower time-adjustment computation, which compensates the received clockSync time by the closely
3 matched duplex-cable delays, is specified by the C code of Equation 1.8.

```
4  
5 void SinkFrame(Frame rxFrame) // Done on receive (1.6)  
6 {  
7     if (rxFrame.type != INVALID) { // VALID frame times are latched  
8         sinkLatch0 = myLocalTime; // (typically by hardware)  
9         sinkFrame= rxFrame; // Accept new timeSync frames  
10    }  
11 }
```

```
12 void SendFrame(Frame sendFrame) // Done on transmit (1.7)  
13 {  
14     if (sendFrame.type != INVALID) // VALID frame times are latched  
15         sendLatch0 = myLocalTime; // (typically by hardware)  
16 }
```

```
17 // Periodic firmware steps, invoked after timeSync has been received (1.8)  
18 Frame FormFrame(Frame sinkFrame, Times sinkLatch, Times sendLatch)  
19 {   Frame sendFrame;  
20  
21     thisType = (sinkFrame.sa > sinkFrame.da) ? REQ:RES;  
22     thisJunk = (sinkFrame.da != myMacAddress);  
23     thisJunk ||= (sinkFrame.sa != pastAddr);  
24     thisJunk ||= (pastType != thisType);  
25  
26     if (sinkFrame.valid != INVAL && !thisJunk) {  
27         if (sinkFrame.number == pastNumb+1 && !pastJunk) {  
28             if (sinkFrame.valid == VALID) {  
29                 myTimeError = sinkFrame.timeLoop - (agedLatch + sendLatch)/2;  
30                 myLinkDelay = sinkFrame.timeThru - (sinkLatch-sendLatch)/2;  
31             }  
32             sendFrame.timeThru = (sendLatch0 - sinkLatch1)/2;  
33             sendFrame.timeLoop = (sinkLatch0 + sendLatch0)/2;  
34             sendFrame.valid = VALID;  
35         } else {  
36             sendFrame.valid = START;  
37         }  
38         pastJunk = FALSE;  
39         myNumber += (thisType == RES);  
40     } else {  
41         thisJunk ||= (sinkFrame.number != pastNumb);  
42         sendFrame.valid = INVAL;  
43     }  
44     pastNumb = (thisType == RES ? myNumber : sinkFrame.number);  
45     pastAddr = sinkFrame.sa;  
46     pastJunk ||= thisJunk;  
47     sendFrame.da= sinkFrame.sa;  
48     sendFrame.number= (thisType == RES ? myNumber+1 : sinkFrame.number);  
49     return(sendFrame);  
50 }  
51  
52  
53  
54
```

2. Normative references

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

Revision History:
Draft 0.01, December 2003 *Initial draft document for study group review.*

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

IEEE P802.17, IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 17: IEEE Draft Standard for Resilient packet ring (RPR) access method and physical layer specifications.

ISO/IEC 9899: 1999, Programming languages - C

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

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

3. Terms, definitions, and notation

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

Revision History:

Draft 0.01, December 2003

Initial draft document for study group review.

3.1 Conformance levels

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

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

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

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

3.2 Terms and definitions

3.2.1 allocated bandwidth: Bandwidth that may be subject to reclamation, but which is to be made available to the allocating station when requested. A class of traffic is said to be allocated when the limits associated with that class are determined by the network operator's engineering / operations department.

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

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

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

3.2.5 bit error ratio (BER): The ratio of the number of bits received in error to the total number of bits transmitted. (IEEE Std 802.3-2000).

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

3.2.7 broadcast: The act of sending a frame addressed to all stations on the network. (See also multicast and unicast.) (IEEE Std 802.5-1998).

3.2.8 broadcast address: A group address that consists of all-ones and identifies all of the stations in a network. (See also individual address and multicast address.) (ISO/IEC2382-25).

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

1 **3.2.10 congested:** The indication from a local station that a threshold has been crossed for the declaration of
2 a state of congestion in attempting to add fairness eligible traffic. (See also congestion.)
3

4 **3.2.11 congestion:** The state of a ringlet or station determined by calculation of the MAC fairness algorithm
5 that causes restriction or regulation of the addition of fairness eligible traffic to the ring. (See also
6 congested.)
7

8 **3.2.12 copy:** The transfer of an inbound frame from the ring to the MAC sublayer. The copying of a frame
9 does not imply its removal from the ring.
10

11 **3.2.13 cyclic redundancy check (CRC):** A specific type of frame check sequence computed using a
12 generator polynomial.
13

14 **3.2.14 delay¹:** The time required to transfer information from one point to another.
15

16 **3.2.15 destination station:** A station to which a frame is addressed.
17

18 **3.2.16 destination stripping:** The removal of frames from the ring by the destination station.
19

20 **3.2.17 downstream:** The direction of data flow on a ringlet. (See also upstream.)
21

22 **3.2.18 dual-ring:** A ring composed of two ringlets sharing the same set of stations such that the order of
23 station traversal for data frames on one ringlet is exactly opposite that of the other ringlet when no faults
24 exist on the ring.
25

26 **3.2.19 edge:** A span on which data frames are not allowed to pass. This is a span on which a wrap is present
27 in a wrapping ring, or a span which is unused for data transmission due to a protection condition in a steering
28 ring. (See also closed ring and open ring.)
29

30 **3.2.20 flooding:** The act of transmitting a frame such that all stations on the ring receive the frame once.
31

32 **3.2.21 frame:** The MAC sublayer protocol data unit (PDU).
33

34 **3.2.22 frame check sequence (FCS):** The field immediately preceding the closing delimiter of a frame,
35 allowing the detection of payload errors by the receiving station
36

37 **3.2.23 group address:** An address that identifies a group of stations on a network. (ISO/IEC 2382-25).
38

39 **3.2.24 header error check (HEC):** A field appended to the frame header for the purpose of detecting, and
40 optionally correcting, errors in the transmission of the header portion of the frame.
41

42 **3.2.25 hop:** The distance associated with the transfer of data from one station to the next station in its path.
43 This distance between adjacent stations is described as 'one hop'.
44

45 **3.2.26 hop-count:** The number of hops traversed by data circulating between stations on the ring.
46

47 **3.2.27 individual address:** An address that identifies a particular station on a network. (See also multicast
48 address and broadcast address.) (ISO/IEC 2382-25).
49

50 **3.2.28 insert:** The placement of a frame on a ringlet by the source station.
51

52 **3.2.29 jitter:** The variation in delay associated with the transfer of frames between two points.
53

54 ¹Delay and latency are synonyms for the purpose of this standard. Delay is the preferred term.

- 3.2.30 link:** A unidirectional channel connecting adjacent stations on a ringlet. (See also span.) 1
- 3.2.31 link rate:** The data rate with which a MAC communicates with its adjacent MAC entity. (See also line rate.) 2
- 3.2.32 line rate:** The rate at which a PHY transfers data on the attached physical medium. (See also link rate.) 3
- 3.2.33 local area network (LAN):** A communications network designed for a small geographic area, typically not exceeding a few kilometers in extent, and characterized by moderate to high data transmission rates, low delay, and low bit error rates. (Adapted from IEEE Std 100, IEEE Std 1278.2-1995, IEEE Std 1278.3-1996.) (See also metropolitan area network.) 4
- 3.2.34 MAC client:** The layer entity that invokes the MAC service interface. 5
- 3.2.35 maximum transfer unit (MTU):** The largest frame (comprising payload and all header and trailer information) that can be transferred across the network. (IEEE Std 100, IEEE Std 610.7-1995). 6
- 3.2.36 medium** (plural: **media**): The material on which information signals are carried; e.g., optical fiber, coaxial cable, and twisted-wire pairs. (IEEE Std 100, ISO/IEC 8802-6-1994, IEEE Std 802.5-1998). 7
- 3.2.37 medium access control (MAC) sublayer:** The portion of the data link layer that controls and mediates the access to the network medium. In this standard, the MAC sublayer comprises the MAC datapath sublayer and the MAC control sublayer. (IEEE Std 100, ISO/IEC 8802-5-1995). 8
- 3.2.38 multicast:** Transmission of a frame to stations specified by a group address. (See also broadcast and unicast.) 9
- 3.2.39 multicast address:** A group address that is not a broadcast address, i.e., is not all-ones, and identifies some subset of stations on the network. (See also individual address and broadcast address.) (ISO/IEC 2382-25). 10
- 3.2.40 neighbor:** A station that is exactly one link distant from a given station on the network. 11
- 3.2.41 network:** A set of communicating stations and the media and equipment providing connectivity among the stations. 12
- 3.2.42 open ring:** A ring that has been cut, thereby preventing it from completing a full loop. An open ring has at least one detected edge. (See also edge and closed ring.) 13
- 3.2.43 opportunistic:** The property of an algorithm that seeks the ability to utilize capacity that is available, or that becomes available. 14
- 3.2.44 opposing ringlet:** A ringlet whose traffic circulates in the direction opposite that of a given ringlet. 15
- 3.2.45 packet:** A generic term for a PDU associated with a layer-entity above the MAC sublayer. 16
- 3.2.46 path:** The specific sequence of stations and links traversed by a frame transferred between two stations on the network. 17
- 3.2.47 physical layer (PHY):** The layer responsible for interfacing with the transmission medium. This includes conditioning signals received from the MAC for transmitting to the medium and processing signals received from the medium for sending to the MAC. (ISO/IEC 8802-5-1995). 18

1 **3.2.48 plug-and-play:** The requirement that a station perform transit, strip, and ring control activities
2 without operator intervention (except for any intervention needed for connection to the ring). The station can
3 additionally copy and insert frames.
4

5 **3.2.49 police:** The enforcement of traffic parameters.
6

7 **3.2.50 protocol data unit (PDU):** Information delivered as a unit between peer layer entities that contains
8 control information and, optionally, data. (IEEE Std 802.5-1998).
9

10 **3.2.51 protocol implementation conformance statement (PICS):** A statement of which capabilities and
11 options have been implemented for a given Open Systems Interconnection (OSI) protocol. (IEEE Std
12 802.5-1998).
13

14 **3.2.52 rate:** the number of bytes transferred per time.
15

16 NOTE—In this standard, rate is measured in bytes per AGEcoef agingIntervals when nothing else is stated explicitly.
17

18 **3.2.53 receive:** The transfer of an inbound frame from the ring to the medium access layer of the station.
19 (See also transmit and transit.)
20

21 **3.2.54 replicated frame:** A frame that is copied for sending bidirectionally on both ringlets.
22

23 **3.2.55 ring:** The collection of stations and links forming a resilient packet ring.
24

25 **3.2.56 ringlet:** A closed unidirectional path formed by an ordered set of stations and interconnecting links,
26 such that each station has exactly one link entering the station and one link exiting the station.
27

28 **3.2.57 service class²:** The categorization of MAC service by delay bound, relative priority, data rate
29 guarantee, or similar distinguishing characteristics.
30

31 **3.2.58 service data unit (SDU):** Information delivered as a unit between adjacent layer entities, possibly
32 also containing a PDU of the upper layer. (IEEE Std 802.5-1998).
33

34 **3.2.59 service guarantee:** Delay or jitter bounds or bandwidth guaranteed for an instance of a service class.
35

36 **3.2.60 shaper:** A device that converts an arbitrary traffic flow to a smoothed traffic flow at a specified data
37 rate.
38

39 **3.2.61 source station:** The station that originates a frame with respect to the ring.
40

41 **3.2.62 source stripping:** The removal of frames by the source station after circulation on the ring.
42

43 **3.2.63 span:** The portion of a ring bounded by two adjacent stations. A span consists of a pair of unidirec-
44 tional links transmitting in opposite directions. (See also link.)
45

46 **3.2.64 spatial reuse:** The concurrent transfer of independent traffic on nonoverlapping portions of a ringlet.
47

48 **3.2.65 station:** A device attached to a network for the purpose of transmitting and receiving information on
49 that network. (IEEE Std 100, IEEE Std 1073.3.1-1994, IEEE Std 1073.4.1-1994, ISO/IEC 8802-5-1995).
50
51
52

53 _____
54 ²Service class and traffic class are synonyms for the purposes of this standard. Service class is the preferred term.

- 3.2.66 steering:** A protection method in which a source station redirects a unicast frame to the ringlet retaining connectivity to a destination station, or a multicast/broadcast frame to one or both ringlets retaining connectivity to destination stations. (See also wrapping.) 1
2
3
4
- 3.2.67 strict data frame:** A frame having its *so* (strict order) bit set. 5
6
- 3.2.68 strip:** The removal of a frame from a ringlet. 7
8
- 3.2.69 topology:** The arrangement of links and stations forming a network, together with information on station attributes. 9
10
- 3.2.70 traffic class:** See service class. 11
12
- 3.2.71 transfer:** The movement of an SDU from one layer to an adjacent layer. Also used generally to refer to any movement of information from one point to another in the network. (ISO/IEC 2382-09). 13
14
15
- 3.2.72 transit:** The passing of a frame through a station via the ring. (See also receive and transmit.) 16
17
- 3.2.73 transit delay:** The delay experienced by a frame transiting a station on the ringlet. 18
19
- 3.2.74 transit queue:** A queue maintained for the purpose of storing a transit frame at a station until that frame has permission to continue transit of the ringlet. 20
21
22
- 3.2.75 transmit (transmission):** The action of a station placing a frame on the medium. (IEEE Std 802.5-1998, ISO/IEC 8802-5-1995.) (See also receive and transit.) 23
24
25
- 3.2.76 transparent bridging:** A bridging mechanism that is transparent to the end stations. (ISO/IEC 8802-5-1995). 26
27
28
- 3.2.77 unicast:** The act of sending a frame addressed to a single station. (See also broadcast and multicast.) 29
30
- 3.2.78 unidirectional flooding:** A frame forwarding transfer involving sending a flooding frame in the downstream direction of one ringlet, with that frame being directed to its sending station. (See also bidirectional flooding.) 31
32
33
34
- 3.2.79 unknown unicast:** A unicast frame for which the location of its destination is unknown. 35
36
- 3.2.80 unallocated bandwidth:** Bandwidth which is not allocated to any provisioned service. (See also allocated bandwidth.) 37
38
39
- 3.2.81 unreserved:** A designation for traffic capacity which is not reserved. This is also a designation for the traffic occupying that bandwidth. In addition, unreserved bandwidth may or may not be allocated bandwidth. (See also reserved bandwidth, allocated bandwidth, unallocated bandwidth, and reclaimable bandwidth.) 40
41
42
43
44
- 3.2.82 upper layers:** The collection of protocol layers above the data-link layer. 45
46
- 3.2.83 upstream:** The direction opposite that of the data flow on a ringlet. (See also downstream.) 47
48
- 3.2.84 virtual destination queuing (VDQ):** The maintenance of a dedicated transmit-queue for each destination. 49
50
51
- 3.2.85 virtual LAN (VLAN):** A subset of the active topology of a bridged local area network. (IEEE Std 100). 52
53
54

1 **3.2.86 wrapping:** The transmission of a frame on an opposing ringlet in order to route around a fault in a
2 given ringlet. (See also steering.)
3

4 **3.3 State machines**

5
6 The operation of a protocol can be described by subdividing the protocol into a number of interrelated
7 functions. The operation of the functions can be described by state machines. Each state machine represents
8 the domain of a function and consists of a group of connected, mutually exclusive states. Only one state of a
9 function is active at any given time. A transition from one state to another is assumed to take place in zero
10 time (i.e., no time period is associated with the execution of a state), based on some condition of the inputs to
11 the state machine.
12

13 The state machines, whether embodied in formal state tables or diagrams, contain the authoritative statement
14 of the functions they depict. When apparent conflicts between descriptive text and state machines arise, the
15 order of precedence shall be formal state tables first, followed by the diagrams representing the state
16 machines, over the descriptive text. This does not override, however, any explicit description in the text that
17 has no parallel in the state diagrams. Where a state machine is described by both a state table and a state
18 diagram, the table is normative and the diagram informative.
19

20 The models presented by state machines are intended as the primary specifications of the functions to be
21 provided. It is important to distinguish, however, between a model and a real implementation. The models
22 are optimized for simplicity and clarity of presentation, while any realistic implementation might place
23 heavier emphasis on efficiency and suitability to a particular implementation technology. It is the functional
24 behavior of any unit that has to match the standard, not its internal structure. The internal details of the
25 model are useful only to the extent that they specify the external behavior clearly and precisely.
26

27 **3.3.1 State table notation**

28 State machines may be represented in tabular form. The table is organized into two columns: a left hand side
29 representing all of the possible states of the state machine and all of the possible conditions that cause transi-
30 tions out of each state, and the right hand side giving all of the permissible next states of the state machine as
31 well as all of the actions to be performed in the various states. No time period is associated with the
32 transition from one state to the next.
33

34 Each combination of current state, next state, and transition condition linking the two is assigned to a
35 different row of the table. Each row of the table, read left to right, provides: the name of the current state; a
36 condition causing a transition out of the current state; an action to perform (if the condition is satisfied); and,
37 finally, the next state to which the state machine transitions, but only if the condition is satisfied. The symbol
38 “—” signifies the default condition (i.e., operative when no other condition is active) when placed in the
39 condition column, and signifies that no action is to be performed when placed in the action column.
40 Conditions are evaluated in order, top to bottom, and the first condition that evaluates to a result of TRUE is
41 used to determine the transition to the next state. If no condition evaluates to a result of TRUE, then the state
42 machine remains in the current state. The starting or initialization state of a state machine is always labeled
43 "START" in the table (though it need not be the first state in the table). Every state table has such a labeled
44 state.
45

46 Each row of the table is preferably provided with a brief description of the condition and/or action for that
47 row. The descriptions are placed after the table itself, and linked back to the rows of the table using numeric
48 tags.
49

50 A RETURN state is the terminal state of a state machine that is intended to be invoked by another state
51 machine. Once the RETURN state is reached, the state machine terminates execution, effectively ceasing to
52 exist until the next invocation by the caller, at which point it begins execution again from the START state.
53
54

State machines that contain a RETURN state are considered to be only instantiated when they are invoked. They do not have any persistent (static) variables.

Table 3.1 provides an informative example of the state table notation. Note that the syntax of the expressions follows standard C notation (see 3.11).

Table 3.1—State table notation example

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

Row 3.1-1: The size of the queued MAC control frame shall be less than the PTQ space.

Row 3.1-2: In the absence of MAC control transmission credits, no action is taken.

Row 3.1-3: MAC control transmissions have precedence over client transmissions.

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

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

3.4 Arithmetic and logical operators

In addition to commonly accepted notation for mathematical operators, Table 3.2 summarizes the symbols used to represent arithmetic and logical (Boolean) operations. Note that the syntax of operators follows standard C notation (see 3.11).

Table 3.2—Special symbols and operators

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

3.5 Numerical representation

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

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

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

3.6 Field notations

3.6.1 Use of italics

All field names or variable names (such as *level* or *myMacAddress*), and sub-fields within variables (such as *thisState.level*) are italicized within text, figures and tables, to avoid confusion between such names and similarly spelled words without special meanings. A variable or field name that is used in a subclause heading or a figure or table caption is also italicized. Variable or field names are not italicized within C code, however, since their special meaning is implied by their context. Names used as nouns (e.g., subclassA0) are also not italicized.

3.6.2 Field conventions

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

Table 3.3—Names of fields and sub-fields

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

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

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

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

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

3.6.3 Field value conventions

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

Table 3.4—wrap field values

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

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

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

3.7 Bit numbering and ordering

Data transfer sequences normally involve one or more cycles, where the number of bytes transmitted in each cycle depends on the number of byte lanes within the interconnecting link. Data byte sequences are shown in figures using the conventions illustrated by Figure 3.1, which represents a link with four byte lanes. For multi-byte objects, the first (leftmost) data byte is the most significant, and the last (rightmost) data byte is the least significant

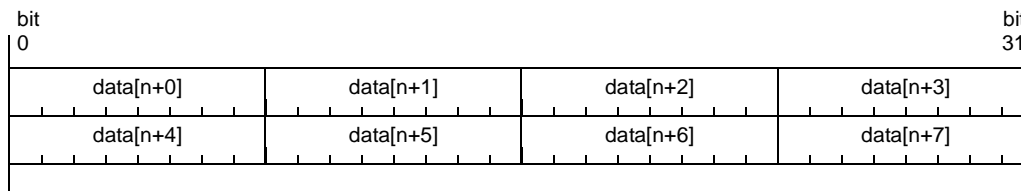


Figure 3.1—Bit numbering and ordering

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

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

3.8 Byte sequential formats

Figure 3.2 provides an illustrative example of the conventions to be used for drawing frame formats and other byte sequential representations. These representations are drawn as fields (of arbitrary size) ordered along a vertical axis, with numbers along the left sides of the fields indicating the field sizes in bytes. Fields are drawn contiguously such that the transmission order across fields is from top to bottom. The example shows that *field1*, *field2*, and *field3* are 1-, 1- and 6-byte fields, respectively, transmitted in order starting with the *field1* field first. As illustrated on the right hand side of Figure 3.2, a multi-byte field represents a sequence of ordered bytes, where the first through last bytes correspond to the most significant through least significant portions of the multi-byte field, and the MSB of each byte is drawn to be on the left hand side.

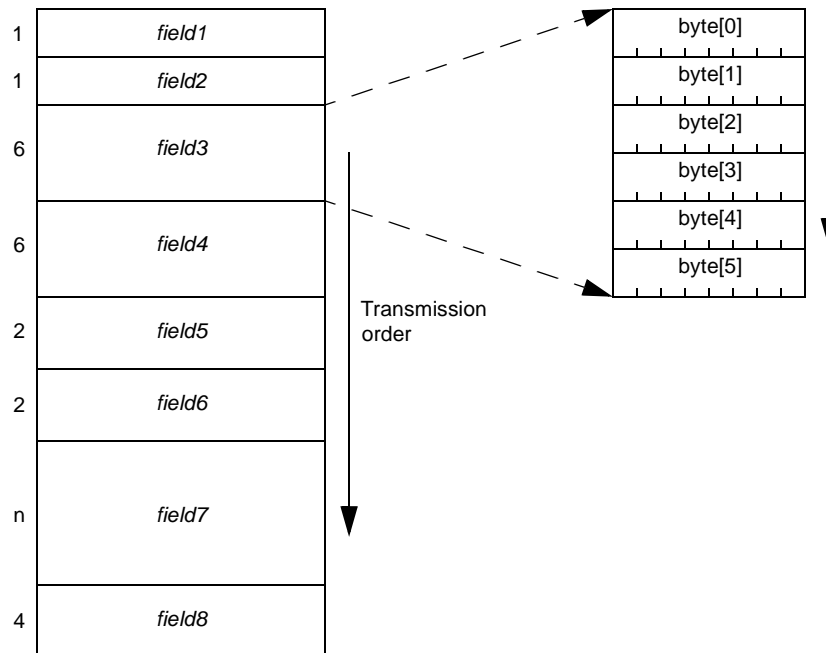


Figure 3.2—Byte sequential field format illustrations

NOTE—Only the left-hand diagram in Figure 3.2 is required for representation of byte-sequential formats. The right-hand diagram is provided in this description for explanatory purposes only, for illustrating how a multi-byte field within a byte sequential representation is expected to be ordered. The tag “Transmission order” and the associated arrows are not required to be replicated in the figures.

3.9 Left-to-right ordering

In many cases, bit fields within byte or multibyte objects are expanded in a horizontal fashion, as illustrated in the right side of Figure 3.3. The fields within these objects are illustrated as follows: left-to-right is the byte transmission order; the left-through-right bits are the most significant through least significant bits respectively.

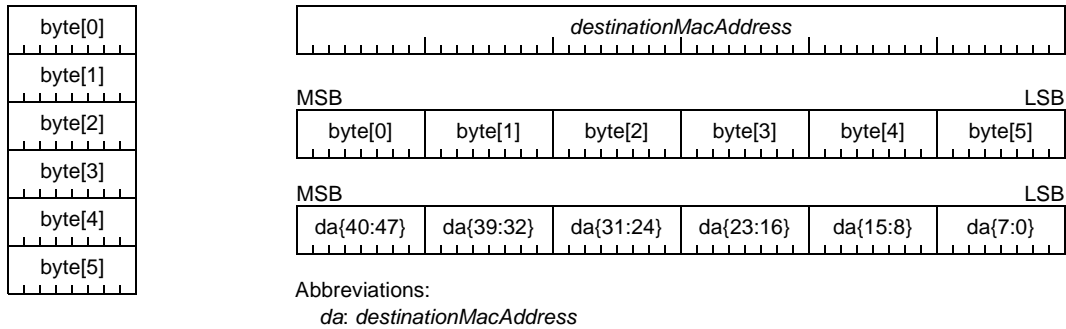


Figure 3.3—Multi-field illustrations

NOTE—Only the right-hand diagram in Figure 3.3 is used within normative clauses. The left-hand diagram is provided in this description for explanatory purposes only.

The 6-byte *destinationMacAddress* field can be illustrated as a single entity or a multibyte entity. For consistency with bit and byte ordering conventions, bits are numbered in a left-to-right order. Depending on the interconnect, this RPR bit-numbering order may or may not differ from the physical layer bit transmission ordering.

3.10 Informative notes

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

NOTE—This is an example of an informative note.

3.11 Conventions for C code used in state machines

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

3.12 Ringlet orientation conventions

By default, figures containing illustrations of RBR interfaces depict the East interface to ringlet0 on the upper right, and the West interface to ringlet1 on the lower left, as illustrated in Figure 3.4.

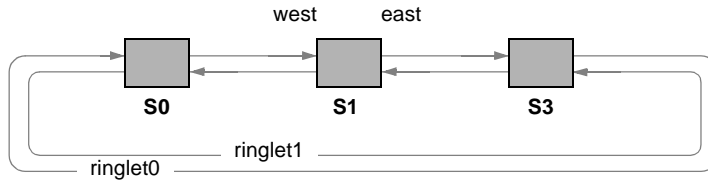


Figure 3.4—Ringlet orientation conventions

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

4. Abbreviations and acronyms

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

References:
None

Definitions:
None.

Abbreviations:
None.

Revision History:

<i>Draft 0.1, February 2002</i>	<i>Initial draft document for WG review.</i>
<i>Draft 0.2, April 2002</i>	<i>Draft 0.2 for TF review, modified according to comments on D0.1.</i>
<i>Draft 0.3, June 2002</i>	<i>Draft 0.3 for TF review, modified according to comments on D0.2.</i>
<i>Draft 1.0, August 2002</i>	<i>Draft 1.0 for WG review, modified according to comments on D0.3.</i>
<i>Draft 1.1, October 2002</i>	<i>Draft 1.1 for WG review, modified according to comments on D1.0.</i>
<i>Draft 2.0, December 2002</i>	<i>Draft 2.0 for WG ballot, modified according to comments on D1.1.</i>
<i>Draft 2.1, February 2003</i>	<i>Draft 2.1 for WG review, modified according to comments on D2.0.</i>
<i>Draft 2.2, April 2003</i>	<i>Draft 2.2 for WG ballot, modified according to comments on D2.1.</i>

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

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

Each acronym in this section should be defined (i.e., spelled out in full) when first used in this draft Standard, as specified in 13.6 of the IEEE Standards Style Manual. For example, the first use of BER should be spelled out as follows: "... bit error ratio (BER) ...". All Section Editors are requested to review their clauses and ensure that this rule is uniformly followed. Note that as far as is known, the editors have adhered to this rule and no changes to the draft are needed. This editor's note is present as a reminder.

This standard contains the following abbreviations and acronyms:

BER	bit error rate
CRC	cyclic redundancy check
DA	destination address
FCS	frame check sequence
FIFO	first in first out
HEC	header error check
IEEE	Institute of Electrical and Electronics Engineers
LAN	local area network
LR	line rate
LSB	least significant bit
LVDS	low-voltage differential signals
LVTTL	low-voltage transistor-transistor logic
MAC	medium access control
MAN	metropolitan area network
MSB	most significant bit
MTU	maximum transfer unit
NE	network element
OAM	operations, administration, and maintenance
OIF	Optical Internetworking Forum
OSI	open systems interconnect
PDU	protocol data unit

1	PHY	physical layer
2	PICS	protocol implementation conformance statement
3	PPM	parts per million
4	PTQ	primary transit queue
5	RBR	resilient backplane ring
6	RPR	resilient packet ring
7	SA	source address
8	SDU	service data unit
9	STQ	secondary transit queue
10	VLAN	virtual LAN
11	WAN	wide area network
12	XAUI	10 gigabit attachment unit interface

13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

Annexes

Annex A

(informative)

Bibliography

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

Revision History:

Draft 0.01, December 2003

Initial draft document for study group review.

[B1] ISO/IEC 8802-5:1998E [IEEE Std 802.5-1998E], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 5: Token ring access method and Physical Layer specifications.

[B2] ISO/IEC 9314-2:1989, Information processing systems—Fibre Distributed Data Interface (FDDI)—Part 2: Token Ring Media Access Control (MAC).

[B3] IEEE Std 802-2002, IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture.

[B4] IEEE Std 1596-1992, IEEE Standards for Scalable Coherent Interface (SCI).

[B5] IEEE Std 1394-1995, IEEE Standards for High Performance Serial Bus.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

Annex B

(informative)

C-code illustrations of TBD computations

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

Revision History:

Draft 0.01, December 2003

Initial draft document for study group review.

This annex provides code examples that illustrate the computation of the TBDs. The code in this annex is purely for informational purposes, and should not be construed as mandating any particular implementation. In the event of a conflict between the contents of this annex and a normative portion of this standard, the normative portion shall take precedence.

The syntax used for the following code examples conforms to ISO/IEC 9899:1999.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

```

//
//      1      2      3      4      5      6      7      8      9      0      1      1      1      1
// 456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
// *****
// ***** Commenting *****
// *****
// The "/" commenting convention will be used.
// An in-line comment will be properly indented.
// A comment may follow the statement line:
//
// Sample code is preceded by "| ", as shown below
// | // Checking for exception conditions
// | if (x > 0) // Warning processing code
// | goto Label;
//
// *****
// ***** Naming Conventions *****
// *****
// The code will use these conventions for symbol name formats:
// MY_DEFINE - All caps and underscores between words
// Used when naming defines and enum values
//
// MyType - First letter is upper case; each run-together word is capitalized
// MyRoutine() - Used when naming typedefs and subroutine names
//
// myData - First letter is lower case; each run-together word is capitalized
// myField - Used when naming variable names, and data-structure fields.
//
// *****
// ***** Integer Type Definitions *****
// *****
// typedef unsigned char uint8_t; // assuming 'char' is an 8-bit value
// typedef unsigned short uint16_t; // assuming 'short' is a 16-bit value
// typedef unsigned long uint32_t; // assuming 'long' is a 32-bit value
// typedef unsigned long long uint64_t; // assuming 'long long' is a 64-bit value
// typedef signed char int8_t; // assuming 'char' is an 8-bit value
// typedef signed long int16_t; // assuming 'short' is a 16-bit value
// typedef signed long int32_t; // assuming 'long' is a 32-bit value
// typedef signed long long int64_t; // assuming 'long long' is a 64-bit value

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

```

// *****
// ***** Indenting *****
// *****
// The code is formatted using indent (GNU version 1.2) with the following flags set:
// indent -bap -bl -bli0 -bls -c50 -cbi0 -cd35 -cil -cli0 -di4 -hnl -i4 -ip0 -l132 -lc132 -nbad -ncdb -ncs -nfc1 -nlp -npcs \
// -nut -sc -ss
//
// -bap force a blank line after procedure bodies
// -bl put braces on line after if, etc.
// -bli0 insert braces 0 spaces
// -bls put braces after structure declaration lines
// -c50 code comments start at line position 50
// -cbi0 insert braces after a case label, 0 spaces
// -cd35 declaration comments start at line position 35
// -cil continuation indent, 1 space
// -cli0 case label indent of 0 spaces
// -di4 put identifiers in second available position after type
// -hnl prefer to break long lines at position of input new lines
// -i4 indentation level is 4 spaces
// -ip0 indent parameters in old-style function definitions, 0 spaces
// -l132 maximum length of a line is 132 characters
// -lc132 the maximum line length for comments
// -nbad no blank line is forced after a declaration
// -ncdb no comment delimiters on blank lines
// -ncs no space after a cast operator
// -nfc1 do not format comments in the first column as normal
// -nlp do not line up continued lines at parentheses
// -npcs no space after a procedure name and '('
// -nut do not use tabs
// -sc put '*' at left of comments
// -ss for 1-line for and while statement, force a blank before the semicolon

// *****
// ***** Compiler dependencies *****
// *****
// The GNU C compiler defines assert() properly, most C compilers do not.
// These assert() definitions are provided for non GNU C compilers.

#include <stdio.h>
#ifdef __GNUC__
#include <assert.h>
#else
#ifdef NDEBUG
#define assert(ex) ((void)0)
#else
#define assert(ex) ((void)((ex) ? 0 : ((void)fprintf(stderr, \
"Assertion failed: file \"%s\", line %d\n", __FILE__, __LINE__), \
(void)abort(), 0)))
#endif
#endif
#endif

```

```

// *****
// ***** TBD-XX Generation Code *****
// *****

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

Index

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54