Proposals for
Resilient Ring Protocols

1. Overview

This is a very rough draft proposal for a Resilient Packet Ring Access Protocol (RRP). This document represents a consensus position of multiple participants, but has not yet been reviewed by a wide audience. Therefore, this document does not necessarily represent the consensus of the P802.17 Working Group.

1.1 Document scope and purpose

The following scope and purpose statements, as stated in the project PAR, apply to this standards activity.

> **Scope:** Define a Resilient Packet Ring Access Protocol for use in Local, Metropolitan, and Wide Area Networks, along with appropriate Physical Layer specifications for transfer of data packets at rates scalable to multiple gigabits per second.
>
> **Purpose:** The standard will define a very high-speed network protocol that is optimized for packet transmission in resilient ring topologies. Current standards are either optimized for TDM transport, or optimized for mesh topologies. There is no high-speed (greater than 1 billion bits per second) networking standard in existence, which is optimized for packet transmission in ring topologies.

1.2 Objectives

1.2.1 Objectives summary

The primary objective of RRP is to provide enhanced services for the transmission of Ethernet packets over a ring-based interconnect topology. Secondary objectives include the following:

1) Flexible. Flexibility features, which increases the usefulness of the standard, includes the following:.
   a) Partitions. RRP is easily partitioned to support distinct service level agreements (SLAs).
   b) Synchronized. Nodes can be accurately synchronized to standard global timers.
   c) Classified. Multiple classes of transport services are provided, including:
      i) Synchronous (low latency, voice and video)
      ii) Asynchronous (high-bandwidth, cost-effective)
   d) Ordered. Ordered delivery is maintained within each packet flow.
   e) Extensible: The protocols can be readily extended, in standard or vendor-dependent ways:
      i) Standard headers. Efficient standardized extended header are provided.
      ii) Dependent headers. Self-administered vendor-dependent extended headers are provided.
      iii) Payload. Larger (but less than 18k-byte) packets are allowed.
2) Subclasses. Multiple asynchronous subclasses are available:
   a) Unfair. Latency insensitive high priority traffic, such as prioritized HTTP traffic.
   b) Fair. Latency insensitive lower priority traffic, such as management messages
      i) Fairness with one source/class
      ii) Fairness measured in bytes, over congested segment
   c) Bulk. Latency insensitive opportunistic traffic, such as file transfers.
3) Robust: The interconnect operates well in the presence of transient transmission errors:
   a) Errors: The interconnect operates well in the presence of transient transmission errors:
      i) Checked. Error-checked header and extended headers
      ii) Covered. Encapsulated Ethernet packet (including FCS)
   b) Faults: The interconnect operates well in the presence of persistent of cable failures:
      i) Recovery. Fast topology-change recovery (under 50ms)
      ii) Resilient. Ring survives in presence of single-fiber failure
   c) Plug-and-play: The interconnect operates well in the presence of cable topology changes:
      i) Changes. Nondisruptive insertion and deletion reduces the impact of on-line upgrades.
      ii) Topology. An arbitrary physical-cable topology is allowed
      (only a ring subset of the physical topology is actively enabled).
4) Scalable: The protocols should be scalable in multiple ways:
   a) Distance: RRP is applicable between within-the-home and within-the-planet distances.
   b) Bandwidth: RRP is applicable to low-rate as well as multiple gigabit/second rate media.

1.2.2 Objectives, requirements, and strategies

1.2.2.1 Streaming traffic

Requirement: Support streaming telephony and video traffic.
Strategy: Provide 1 ms latency guarantees for synchronous traffic.
Separate bypass FIFOs for synchronous and asynchronous traffic.
Allow synchronous mid-packet preemption of asynchronous traffic.

1.2.2.2 Service level agreement (SLA)

Requirement: Support SLA (service level agreements) of guaranteed bandwidth.
Strategies: Provide 1 ms latency guarantees for isochronous traffic.
Provide bandwidth guarantees for asynchronous and isochronous traffic.

1.2.2.3 Fairness

Objective: Non-negotiated asynchronous bandwidth should be fairly allocated.
Requirement: Non-negotiable asynchronous traffic shall have bounded delivery delays.
Strategies: Round-robin processing of asynchronous traffic, with queue-size bounds.

1.2.2.4 Wallclock synchronization

Objective: The clock-slave node timers can be accurately synchronized to the clock-master node.
Requirement: The clock-slave node timers do not drift from the timer in the clock-master node.
Strategies: Periodic time-reference packets are sent over duplex links, allowing attached nodes monitor the arrival and departure times of these packets. The clock-slave nodes compensate their frequencies based on the differences between expected and observed times.

1.2.2.5 Scalable

Objective: The link should be applicable to within-the-home through within-the-planet applications.
Requirement: The link should be applicable to within-the-city through within-the-nation applications.
Strategies: All timeouts are based on self-calibrating ring-circulation timers, so the protocols readily adapt to changed in ring diameters. Synchronous traffic preempts asynchronous traffic, so the length of asynchronous packets doesn't effect synchronous packet latencies.

1.2.2.6 Resilient

Objective: The link should recover from topology changes within 50 ms.
Requirement: The link shall recover from topology changes within 10 ringlet-circulation latencies.
Strategies: Link changes involve a topology rediscovery, to handle all join/sever combinations.
Merging and dividing of two segments are optimized special cases.

Objective: The link should operate over arbitrary cable topologies.
Requirement: The link shall operate over daisy-chain or loop cable topologies.
Strategies: A cable topology invokes topology exploration.
That exploration attempts to form a spanning tree by selectively disabling paths.
The spanning-tree protocols detect and enables (rather than disables) a loop topology.

Objective: The location of failed or marginal links should be easily identifiable.

Requirement: The location of failed links should be easily identifiable.

Strategies: Data CRC checks are performed on a hop-by-hop basis.
Bad-CRC packets are distinctively marked, to avoid incrementing spurious error counts.
Miss-addressed packets are aged and quickly discarded on their second ringlet circulation.

1.2.2.7 Ordered

Objective: Within each flow, all packets are transmitted and received in the same order.

Requirement: On each ringlet, packets within each flow are transmitted and received in the same order.

Strategies: On the same ringlet:
All stream packets are transmitted and received in the same order.
All unfair packets are transmitted and received in the same order.
All fair packets are transmitted and received in the same order.

1.3 RRP topologies

1.3.1 Ring topologies

RRP is targeted for cable-ring topologies and maximizes bandwidth capabilities through the use of full-duplex cabling, as illustrated in figure 1. The full-duplex cable infrastructure normally allows concurrent transmissions on the clockwise and counter-clockwise rings, as illustrated in the left of figure1. After a single link failure, communication continues (but at a reduced rate) over the remaining ring, as illustrated in the center figure 1. After a duplex cable failure, communication continues (but at a further reduced rate) over the daisy-chained connection, as illustrated in the right of figure1.

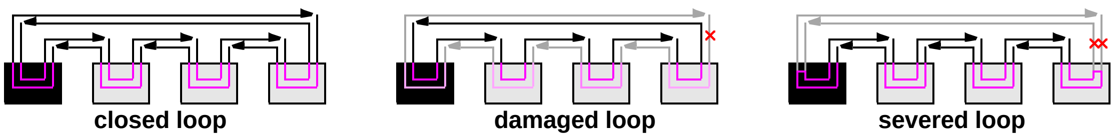

Figure 1—Ring topologies

Higher performance nodes are expected to have two attachments (an attachment is a location where packets can be inserted or extracted). This allows packets to be sent in their preferred direction, or interleaved and sent over both rings. On the average, assuming randomly-distributed traffic and preferred direction prediction, the average path lengths can be reduced by nearly a factor of four (when compared to an open-loop topology). Bandwidth improvements of 2.5 are more typical, due to lowed efficiencies of synchronous traffic, which circumscribes the ring and can therefore not benefit from traveling in the shortest direction.

Similar performance enhancing techniques have also been used on serial-copper SSA, serial-fiber FDDI[1] (Fiber Distributed Data Interface”), and parallel-copper SCX interconnects[2].

[1] A set of communication protocols and a physical-layer interface developed with the intent of exceeding Ethernet data-transmission bandwidths

[2] The SCX Channel: A New, Supercomputer-Class System Interconnect, Steven Scott (Cray Research Inc.), Presented at the Hot Interconnects III (IEEE Computer Society), August 10-12, 1995

1.3.2 Hub topologies

An RRP topology can include larger multiported nodes, called hubs, as illustrated in figure1. An N-ported hub can be used to expand the number of lower-cost single-ported nodes, as illustrated in the left of figure1. Alternatively, hubs can support redundant fault tolerant connections, as illustrated in the right of figure1.

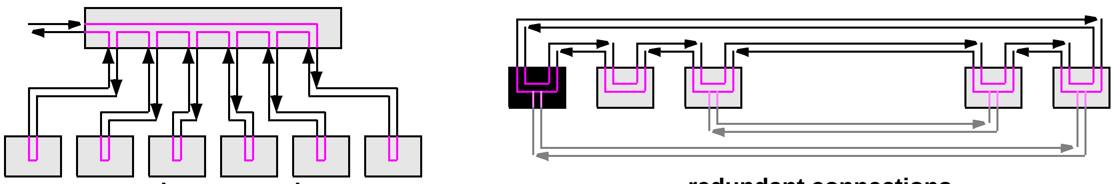

Figure 2—Ring topologies

1.3.3 Root nodes

Each ringlet has a distinctive node, called the root, as illustrated by the black-shaded nodes of figure 1. The root node is selected when the ring is initialized and has several responsibilities, listed below:

1) Timing. The root provides information to synchronize wallclock timers on other attached nodes.
2) Packet aging. The root decrements nonzero time-to-live identifiers in passing through packets.
3) Arbitration grants. The root processes arbitration requests to provide arbitration grants, where these request and grant indications are transported within idle symbols.

The root node selection is constrained as listed below. The intent is to simplify 2-ported nodes by assigning the root node (which is most effected by multiported initialization protocols) to a larger multiported node.

1) Primary. The root is selected from the set of n-ported nodes, where n is larger than 2.
2) Secondary. In the absence of n-ported nodes, the root is selected from the set of 2-ported nodes.
3) Incapable. The root is never selected from the set of 1-ported nodes.

1.4 Node data paths

1.4.1 Adaptable link routing

Within a simple 2-ported node, the data-paths are expected to flow through four multiplexers, whose controls are set during each bus reset. These multiplexers allow the node to utilize both ports (the normal condition), or to bypass a defective/redundant left or right port, as illustrated in figure3

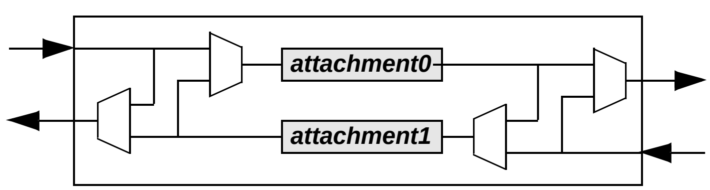

Figure 3—Selectable data paths

This concept of an electronically-switched router is not new; a similar capability is provided by nodes attached to Serial Bus. Although SerialBus supports N-port attachments, a 2-port design is sufficient to support the common topologies and simplifies the hardware design.

1.4.2 Simplified attach-point routing

A node's attach point has multiple queues in each attach component, as described in 5.1. To simplify this overview discussion, only a few of these queues and their simplified behaviors is described. Each attach point has a receive interface, a transmit interface, and a bypass buffer for resolving concurrent reception/transmission conflicts, as illustrated in frame-1 of figure4.

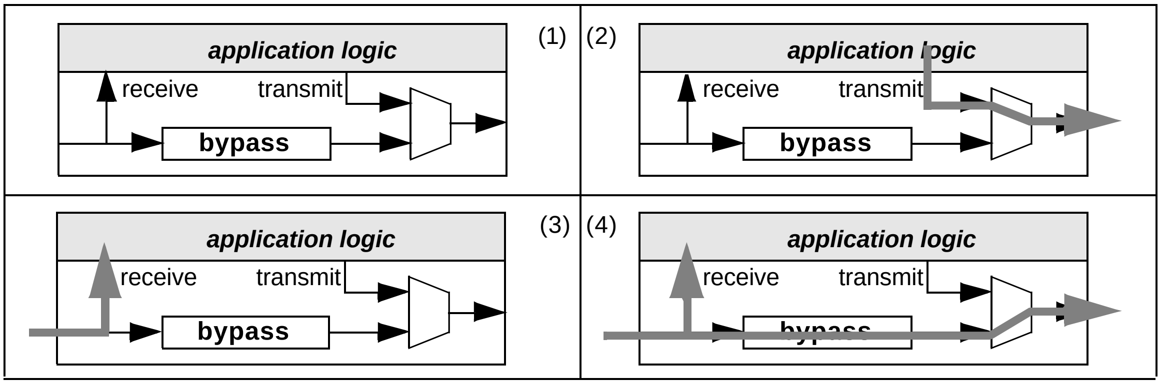

Figure 4—Attach point routing

The simplified packet-transmission protocols are described below:

1) Setup. An outbound packet is readied for transmission. A send buffer (not illustrated) may be necessary for the packet to be transmitted at the (possibly faster) speed of the link.
2) Transmit. The send packet is transmitted, as illustrated in frame-2 of figure 4. During the transmission interval, incoming traffic is saved in the bypass buffer.
3) Receive. Packet are received based on their *destinationMacAddress*, as follows:
   a) Directed. Unicast packets are stripped at their destination, as illustrated in frame-3 of figure 4:
      i) Copy. A packet copy is routed to the receive port.
      ii) Strip. Idle symbols (rather than the packet) are routed to the bypass buffer.
   b) Multicast. Multicast/broadcast packets are copied, as illustrated in frame-4 of figure4:
      i) Copy. A packet copy is routed to the receive port.
      ii) Pass. A packet copy is routed to the bypass buffer.
4) Recover. Incoming idle symbols are discarded, allowing the bypass-buffer symbols to be emptied.

1.5 Arbitration protocols

On traditional backplane buses, arbitration protocols are used to resolve conflicts and schedule packet transmissions. To avoid transmission conflicts, arbitration protocols are invoked before every packet transmission. The arbitration protocols may invoke prioritized and/or fair arbitration protocols to determine which packets can be sent.

In the networking environment, the overhead of invoking arbitration protocols before each packet transmission is no longer acceptable. Transmission conflicts are more efficiently resolved by transmitting during inter-packet gaps, buffering conflicting arrivals in bypass FIFO storage, and repeating the buffered packet when packet transmissions cease. Arbitration protocols are still needed to allocate bandwidth, so that the node's service level agreements (SLA) can be met, but are only invoked when heavy congestion conditions necessitate their use. A more efficient opportunistic transmission protocol is used under light loading conditions, to maximize bandwidth utilization under typical usage conditions.Traffic classes

1.5.1 Traffic classes

RRP supports three types of delivery services, as illustrated in figure5 and listed below. Each node is required to police its stream and unfair traffic to avoid exceeding its prenegotiated limits.

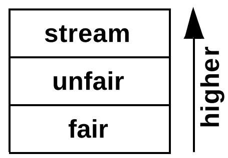

Figure 5—Attach point routing

1) Stream (negotiated synchronous): Prenegotiated low-latency bandwidth.
   This service is expected to be used for transmission of streaming audio and/or video traffic.
2) Unfair (negotiated asynchronous): Prenegotiated higher-latency bandwidth.
   This service is expected to be used for transmission of contracted nonstreaming traffic.
3) Fair (fair asynchronous): Fairly assigned residual asynchronous bandwidth.
   This service is expected to be used for transmission of noncontracted traffic.

1.5.2 Arbitration signals

The key element in the design of the arbitration protocols is the low-latency transmission of arbitration-related information within idle symbols, including the following:

1) Priority. For congested prioritized traffic, unfair arbitration involves the following:
   a) Stream. Assertion of *request4/grant4* values inhibits unfair & fair transmissions.
   b) Unfair. Assertion of *request3/grant3* values inhibits fair transmissions.
2) Fairness. For congested non-prioritized traffic, fair arbitration involves the following:
   a) Requests. Assertion of *request1*/*request2* indications activates round-robin transmissions.
   b) Grants. Observance of *grantA/grantB* indications enables selective fair transmissions.

The arbitration indications flow in the reverse direction, with respect to the data-packet flows, starting from nodes currently requesting their share of the bus bandwidth. The reverse-flow direction allows inactive nodes to delay forwarding of arbitration indications while filling of their bypass FIFO generates idles; nodes which cannot generate idles quickly forward arbitration indications to throttle upstream nodes.

The arbitration indications are level-sensitive signals, rather than tokens or edge-sensitive values, making the protocols robust. Most importantly, from a simplicity perspective, these idle-transported values can be freely inserted and deleted (without effect) when passing through elasticity buffers.

1.5.3 Arbitration signal flow

For efficiency, arbitration signals and data packets normally flow in the opposite direction, as illustrated in figure6. Arbitration signals for top-run transmissions (solid lines) are sent on the bottom run (dotted lines), as illustrated in the left half of figure 6. Arbitration signals for bottom-run transmissions (solid lines) are sent on the top run (dotted lines), as illustrated in the right half of figure6.

Arbitration on nonloop topologies attempts to send arbitration and data in opposite directions, but these attempts are less effective, as illustrated in figure 7. Packet transmissions from the black attachments are coupled to arbitration indications within the white attachments; packet transmissions from the white attachments are coupled to arbitration indications within the black attachments.

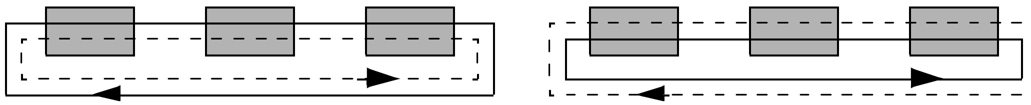

Figure 6—Opposing packet and arbitration paths

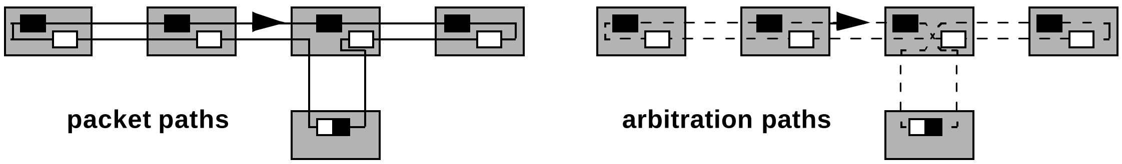

Figure 7—Reversed packet and arbitration flows

When passing through a multiport node, data and arbitration flow in opposite directions. Thus, if data ports are connected in a clockwise fashion, arbitration signal paths are connected in a counterclockwise fashion.

1.5.4 Arbitration concepts

For simplicity, ringlet arbitration is introduced in the context of a damaged ringlet (see 1.3), where only one of the ringlets is logically enabled, as illustrated in frame-1 of figure8. Within this frame, the enabled and disabled ringlets are drawn with solid and dotted lines respectively. Since arbitration protocols are uninfluenced by the disabled ringlet, the dotted disabled ringlet is removed from the illustrations, as illustrated in frame-2 of figure8.

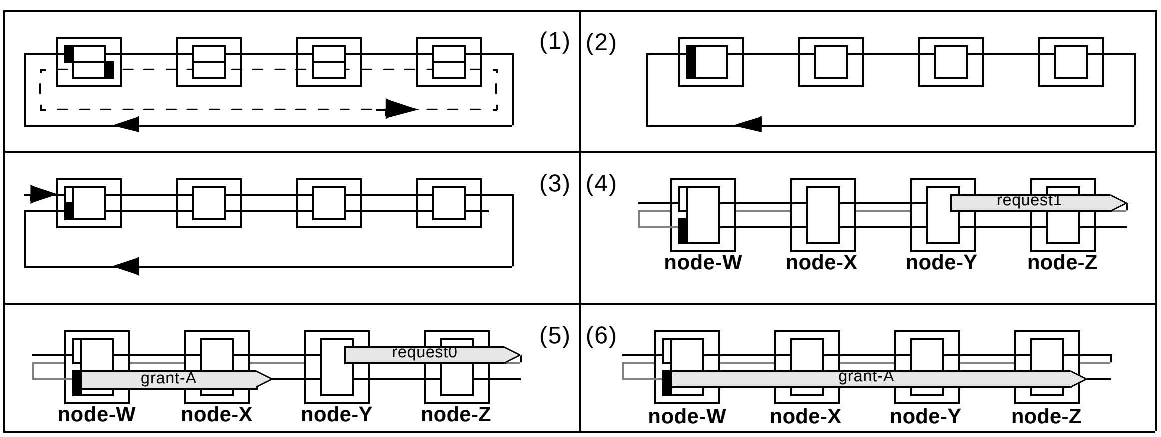

Figure 8—Arbitration illustrations

Arbitration signaling involves encoding state indications within the *arbs* field inside idle symbols. The coding of the arbs field supports the sampling of request indications on pass-1 and the distribution of grant indications on pass-2. Separate lines are used to illustrate the flow of logically distinct pass-1 and pass-2 information, as illustrated in frame-3 of figure8, even though this information flows over only one link.

Wide shaded-gray lines illustrate the flow of asserted pass-1 request and pass-2 grant information. One node (called the root) has enabled arbitration-processing components, shaded white and black. The pass-1 root processing initializes the pass-1 indication to a deasserted value. An intervening node can assert a pass-1 request-0 indication, as illustrated in frame-4 of figure8, which eventually flows to the root.

That root node is responsible for converting the arbitration request into an arbitration-grant indication, as illustrated in frame-5 of figure8. An asserted grant indication has the effect of inhibiting packet-send operations on less congested nodes.

In this and the following discussions, arbitration is described in an unusual worst-case condition, for the purpose of simplifying the illustrations and affiliated discussions. In reality, arbitration is typically a more transient event; the *reguest* indications are often deasserted before *grant* indications are returned, as illustrated in frame-6 of figure8.

1.5.5 Prioritized arbitration

Prioritized arbitration involves the assertion of priority-4 or priority-3 requests, for stream and unfair traffic respectively. The arbitrating node-Y asserts a request4/grant4 indication, inhibiting transmissions from adjacent nodes, as illustrated in frame-1 of figure 9. At the root, the incoming request4 is converted to a grant4 and the incoming grant4 is ignored, as illustrated in frame-2 of figure 9. The grant4 indication eventually propagates through the ringlet, as illustrated in frame-3 of figure 9.

Another node-X attachment may assert a request3 indiction, which propagates to the request4-generating node, as illustrated in frame-4 of figure 9. When node-Y is satisfied, it propagates the incoming request3, rather than asserting the request4 indication, as illustrated in frame-5 of figure 9. The closer-to-the-root node-W and node-X attachments are the first to observe the lowered priority level, as illustrated in frame-6 of figure9. The lowered priority indication circulates quickly to the remaining nodes, as illustrated in frame-7 of figure 9.

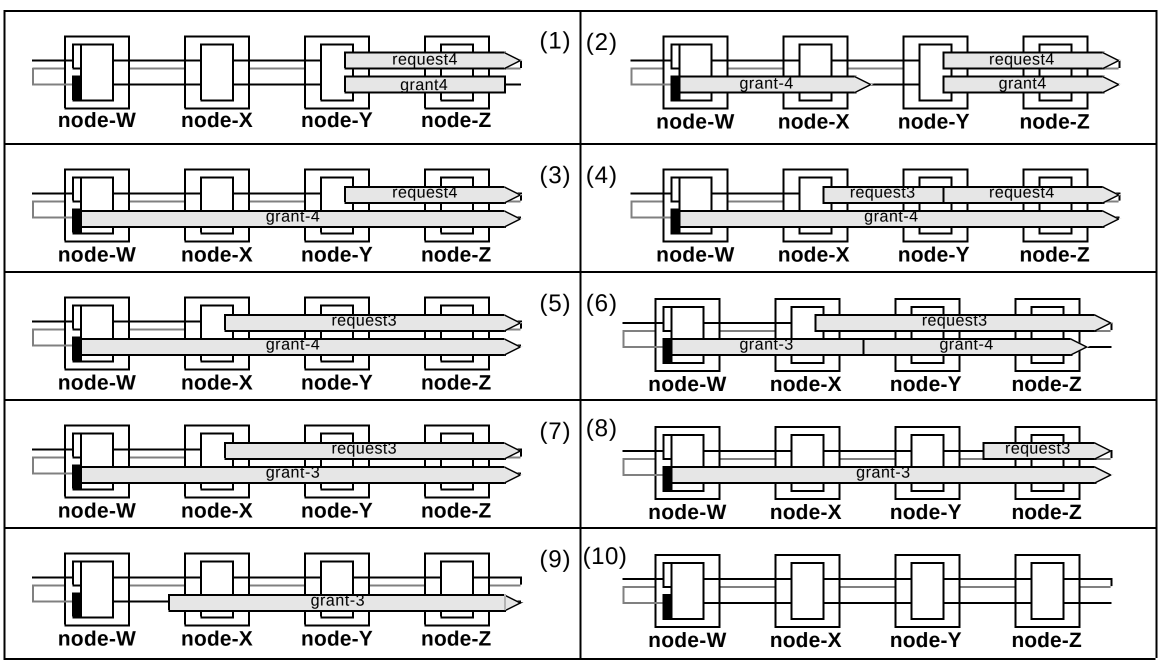

Figure 9—Prioritized arbitration

Deassertion of request3 indication has no effect on other nodes while the deassertion propagates to the root, as illustrated in frame-8 of figure9. The root then deasserts its grant-3 indication, as illustrated in frame-9 of figure9. That deassertion quickly propagates around the ringlet, allowing nodes to arbitrate in an opportunistic fashion, as illustrated in frame-10 of figure 9.

1.5.6 Fairness arbitration

Fair arbitration involves the assertion of level-1 or and level-2 requests, for initial and aged requests respectively. The arbitrating node-Y asserts a request-1 indication, inhibiting fair transmissions from downstream nodes, as illustrated in frame-1 of figure 10. That request is converted into a grant-A or grantB indication (depending on the arbitration history of the root) as illustrated in frame-2 of figure 10. That grant-A indication eventually returns to the underserviced node-Y, as illustrated in frame-3 of figure 10.

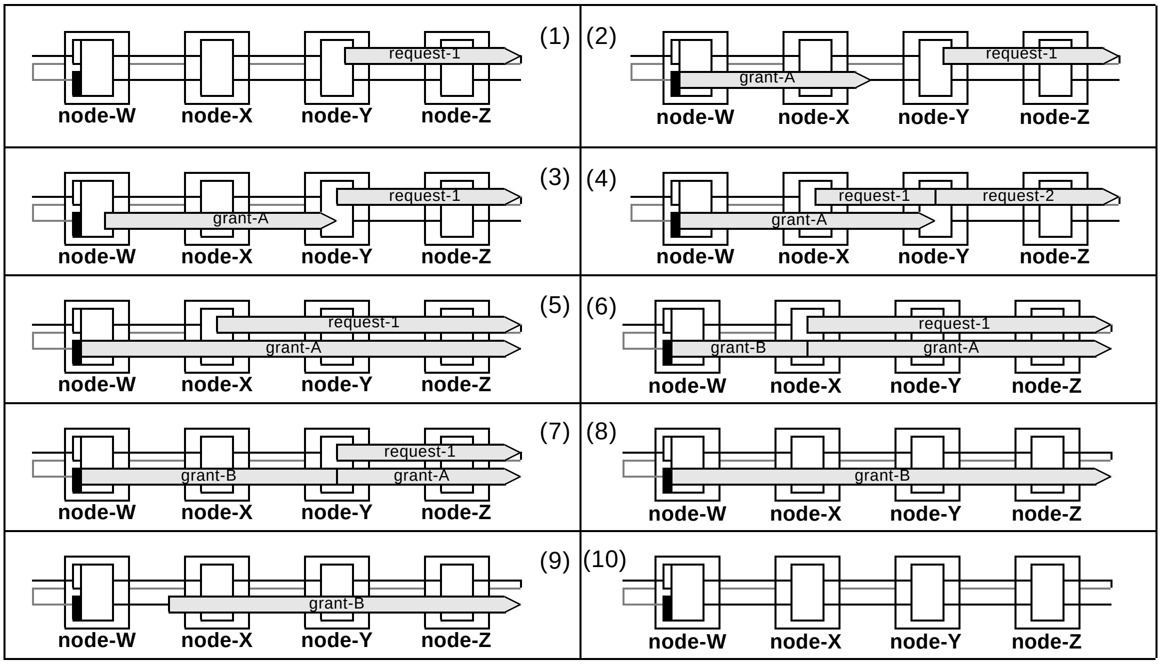

Figure 10—Fairness arbitration

The underserviced node-Y inhibits propagation of grant-A indications until its packets have been sent. The upstream node-X observes grant-A input&output indication, which inhibits its transmissions, as illustrated in frame-4 of figure 9. Then every node except node-Y is inhibited from sending (node-Y has the “token”), so node-Y will rapidly get enough idles to clear its transmission queues.

The token-holding node-Y generates a request-2 output when aging its request-1 input, as illustrated in frame-4 of figure 9. The request-2 assertion has the effect of unlodging redundant downstream token holders, when and if such a transmission-error-related condition arises.

When the grant-A indication returns to the root, the root’s simultaneous observation of request-1 and grant-A indications triggers the generation of grant-B indications, as illustrated in frame-6 of figure9. Propagation of the grant-B wavefront continues after node-X has finished its transmissions, as illustrated in frame-7 of figure 9. In the absence of other assertions, the request deassertion and the grant-B assertion circulated together through the ringlet, as illustrated in frame-8 of figure9.

When the grant-B indication returns to the root, the root’s simultaneous observation of deasserted request and asserted grant-A indications triggers the deassertion of grant indications, as illustrated in frame-9 of figure9. Opportunistic transmissions continue when the grant deassertion has reached the nodes, as illustrated in frame-10 of figure 9.

At the root, the continued presence of asserted request indications has the effect of generating alternate grant-A and grant-B indications, with only one wavefront edge (which marks the enabled node) present at any time. This ensures continued round-robin servicing, while a congestion condition remains. In this way, the transmission token (the grant-assertion wavefront) rotates continuously around the bus, with the stable performance under high load characteristic of token-ring systems.

2. References

The following documents are referenced by this standard:

ANSI X3.159–1989, Programming Language—C.[3]

ANSI/IEEE Std 1596-1992, Scalable Coherent Interface (SCI) (or IEC/ISO DIS 13961).[4]

[3]ANSI publications are available from the American National Standards Institute, Sales Department, 11 West 42nd St., 13th Floor, New York, NY 10036, 212-642-4900.

[4]ANSI/IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

3. Glossary and notation

3.1 Definitions

3.1.1 Conformance levels

Several keywords are used to differentiate between different levels of requirements and optionality, as follows:

3.1.1.1 expected: A keyword used to describe the behavior of the hardware or software in the design models assumed by this S^2I standard. Other hardware and software design models may also be implemented.

3.1.1.2 may: A keyword that indicates flexibility of choice with no implied preference.

3.1.1.3 shall: A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other S^2I standard conformant products.

3.1.1.4 should: A keyword indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “it is recommended.”

3.1.2 Definitions of RRP related terms

A large number of network and interconnect-related technical terms are used in this document. These terms are defined below:

3.1.2.1 aligned: A term which refers to the constraints placed on the address of the data; the address is constrained to be a multiple of the data format size.

3.1.2.2 asynchronous packet: A component of asynchronous traffic.

3.1.2.3 asynchronous traffic: A collection of data packets with no prenegotiated transmission bandwidth, whose transmission can be preempted by higher priority asynchronous traffic.

3.1.2.4 big endian: A term used to describe the physical location of data-byte addresses within a multibyte register. Within a big-endian register or register set, the data byte with the largest address is the least significant.

3.1.2.5 byte: An 8-bit entity. Within other standards, this is also called an octet.

3.1.2.6 doublet: A data format or data type that is 2 bytes in size.

3.1.2.7 hexlet: A data format or data type that is 16 bytes in size.

3.1.2.8 ignored; ign: A term used to describe the fields within unit-specific CSRs or command/status entries whose zero or last-written values shall be ignored.

3.1.2.9 node: The entity associated with a particular set of (typically bus-dependent) control register addresses (including identification ROM and reset command registers). These registers are typically used to identify the node and to initialize the state of unit-dependent registers. In normal operation each node can be accessed independently (a control register update on one node has no effect on the control registers of another node).

3.1.2.10 octlet: A data format or data type that is 8 bytes in size. Not to be confused with an octet, which has been commonly used to describe 8 bits of data. In this document, the term byte, rather than octet, is used to describe 8 bits of data.

3.1.2.11 DRAM: An acronym for dynamic random-access memory.

3.1.2.12 quadlet: A data format or data type that is 4 bytes in size.

3.1.2.13 reserved; res or **r:** A term used to describe the fields within unit-specific CSRs. On a write, the reserved-field value shall be ignored. On a read, the reserved field value shall be zero.

3.1.2.14 ringlet: A term used to describe a enabled looping data path over which packets can be transferred. Within a given cable topology, there may be one circumscribing ringlet or two counter-rotating ringlets enabled.

3.1.2.15 SCI: *See:* **Scalable Coherent Interface standard**.

3.1.2.16 Scalable Coherent Interface (SCI) standard: Refers to IEEE Std 1596-1992 Scalable Coherent Interface [see clause2], which provides computer-bus-like services using a collection of point-to-point unidirectional links.

3.1.2.17 synchronous packet: A component of synchronous traffic.

3.1.2.18 synchronous traffic: Data traffic for which the transmission bandwidth is prenegotiated and the transmission priority exceeds that of asynchronous traffic.

3.2 Acronyms

A large number of network and interconnect-related acronyms are used in this document; these are defined below:

RRP Resilient Packet Ring Access Protocol
SLA Service level agreement.

3.3 Field names

This document describes the values in unit control registers, mover control registers, status-report parameters, command entries, status entries, and unit-specific control-and-status registers (CSRs). For clarity, names of these values have an italics font and contain the context as well as field names, as illustrated in table1:

Table 1—Names of command, status, and CSR values

Name	Description
UnitCsr.timeOfDay	The unit's synchronized time-of-day clock.
MoverCsr.control	The mover's control register.
Command.code	The *code* field within a command entry.
Status.label	The *label* field within a status entry

3.4 C code notation

The behavior of data-transfer command execution is frequently specified by C code, such as equation 1. To differentiate such code from textual descriptions, such C code listings are formatted using a fixed-width Courier font. Similar C-code segments are included within some figures.

```
// Return maximum of a and b values
Max(a,b) {
  if (a<b)
    return(LT);
  if (a>b)
    return(GT);
  return(EQ);
}                                                              (1)
```

Since the meaning of many C code operators are not obvious to the casual reader, their meanings are summarized in table 2:

Table 2—C code expression summary

<table>
<tr><th>Expression</th><th>Description</th></tr>
<tr><td><code>~i</code></td><td>Bitwise complement of integer i</td></tr>
<tr><td><code>i^j;</code></td><td>Bitwise EXOR of integers i and j</td></tr>
<tr><td><code>i&j;</code></td><td>Bitwise AND of integers i and j</td></tr>
<tr><td><code>i<<j;</code></td><td>Left shift of bits in i by value of j</td></tr>
<tr><td><code>i*j;</code></td><td>Arithmetic multiplication of integers i and j</td></tr>
<tr><td><code>!i</code></td><td>Logical negation of Boolean value i</td></tr>
<tr><td><code>i&&j;</code></td><td>Logical AND of Boolean i and j values</td></tr>
<tr><td><code>i||j;</code></td><td>Logical OR of Boolean i and j values</td></tr>
<tr><td><code>i^= j;</code></td><td>Equivalent to <code>i= i^j</code>.</td></tr>
<tr><td><code>i==j;</code></td><td>Equality test, true if i equals j</td></tr>
<tr><td><code>i!=j;</code></td><td>Equality test, true if i does not equal j</td></tr>
<tr><td><code>i<j;</code></td><td>Inequality test, true if i is less than j</td></tr>
<tr><td><code>i>j;</code></td><td>Inequality test, true if i is greater than j</td></tr>
</table>

3.5 Data formats

3.5.1 Numerical values

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

Decimal numbers are represented in their usual 0, 1, 2, ... format. Hexadecimal numbers are represented by a string of one or more hexadecimal (0-9,A-F) digits followed by the subscript 16, except in C-code contexts, where they are written as `0x123EF2` etc. Binary numbers are represented by a string of one or more binary (0,1) digits, followed by the subscript 2. Thus the decimal number "26" may also be represented as "$1A_{16}$" or "11010_2".

3.5.2 Reserved registers and fields

The portions of the command-entry fields, status-entry fields, and unit-specific CSRs that are not implemented have defined reserved values. This includes optional registers (when the option is not implemented) and reserved registers (which are required to be unimplemented), or unused portions of command/status entries. The capabilities of reserved fields are exactly defined, to minimize conflicts between current implementations and future definitions.

For command-entry and status-entry fields, which are located in system-memory or transfer-buffer spaces, the reserved fields and bits shall be zero when the command is written.

Unused fields within register locations may be either *reserved* or *ignored*. The reserved option allows these field locations to be hardwired-to-zero, so that physical memory elements are not required. Alternatively, an ignored field (a read of an ignored field returns the last-written value) allows registers to be special addresses within a general-purpose memory array.

3.6 Duplex ring reset

Sequences of line states are sometimes used to illustrate the phases of ringlet initialization, as illustrated in figure 11. Within these illustrations, the planned introduction of a new cable (or recent removal of an old cable) is illustrated with dotted lines.,

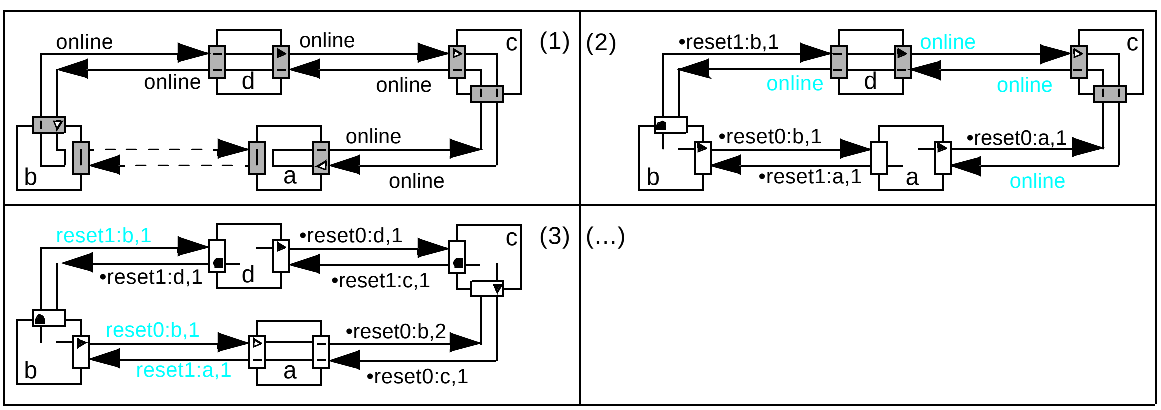

Figure 11—Duplex ring reset

Resets are triggered (2) by cable changes and these resets emanate from the cable-changed detecting nodes. Resets are labeled by name and the umbers after the reset-packet name describe the parameters contained within. Within reset0:d,1 for example, d is the *eui64* and 1 is the *hopCount* value. For clarity, the labels *a*-to-*d* represent increasing larger *eui64* values.

Within these illustrations, the round dot in the packet label indicates a change from the previously illustrated frame. Thus, the line state *•reset0:d,1* in the second frame has changed from the *online* line state of the first frame. The stable line signaling states have no preceding '•' and are colored cyan, which appears as gray in black and white copies.

4. Packet and idle formats

4.1 Packet formats

4.1.1 Aligned payload formats

A packet consists of a basic header, zero or more optional extended header, and payload components. The basic header starts with 64-bit *destinationMacAddress* and *sourceMacAddress* components, as illustrated in figure 12.

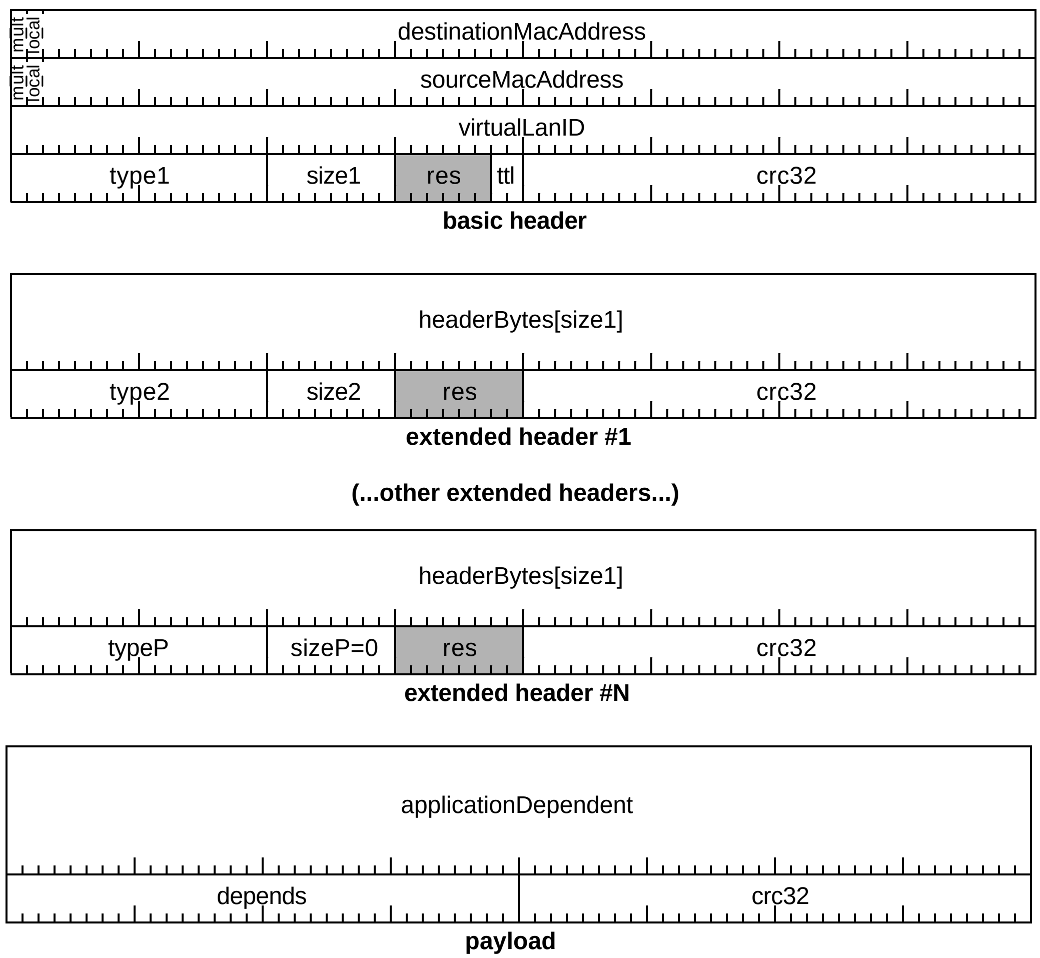

Figure 12—Packet format

The 64-bit ***destinationMacAddress*** field specifies the address of the destination node. The 64-bit ***sourceMacAddress*** field specifies the address of the source node. The 64-bit ***virtualLanId*** field specifies the logical LAN over which the packet is to be transferred.

The 16-bit ***type1*** field specifies the type of this extended header; the 8-bit ***size1*** field specifies the length (in bytes) of the first extended header.

Within the next extended header, the 16-bit ***type2*** field specifies the type of this extended header; the 8-bit ***size2*** field specifies the length (in bytes) of the second extended header. In a similar fashion, the 16-bit ***typeN*** and 8-bit ***sizeN*** fields within the N'th extended header specified the type and size of the following extended header.

Within the final extended header, the 16-bit ***typeP*** field specifies the type packet payload that follows; the 8-bit ***sizeP*** field shall be zero.

When the application data consists of an encapsulated Ethernet packet, the Ethernet-defined CRC-32 shall also be included within the transported packet. The intent is to ensure end-to-end error-checking coverage.

4.1.2 Vendor-dependent extended headers

The format of packets and between packet idle symbols are based on a 64-bit word size, as illustrated in figure 13.

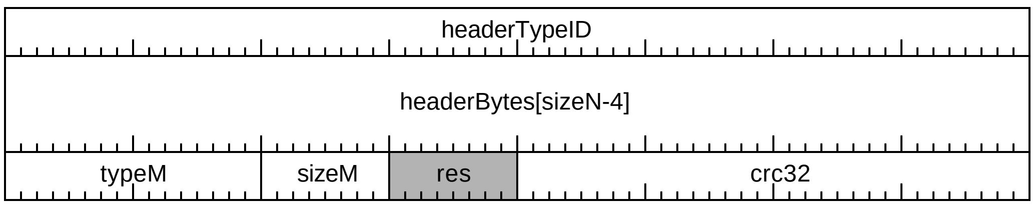

Figure 13—Vendor-dependent extended header

The 64-bit ***headerTypeID*** is an EUI-64 value that distinctively identifies the type of vendor-dependent header.

The meaning of the following ***headerBytes[]*** are *headerTypeID* dependent.

The 16-bit ***typeM*** and 8-bit ***sizeM*** fields specify the type and size of the following extended header. The 32-bit ***crc32*** field covers the preceding data bytes within the extended header.

4.1.3 Unaligned payload formats

To simplify high-speed processing, all data payloads shall be padded (as necessary) to the next 8-bytes boundary, as illustrated in figure 14.

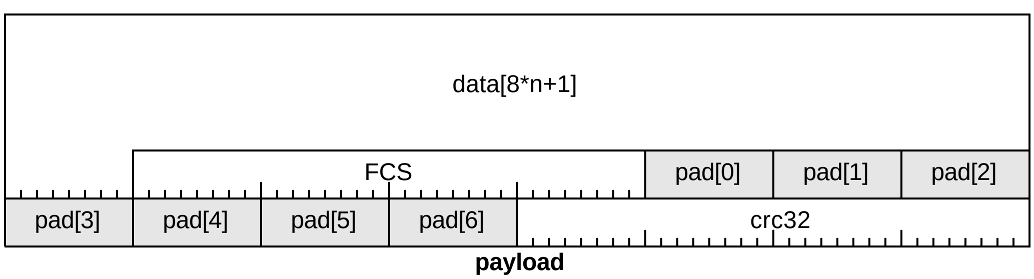

Figure 14—Padded payloads

Possible physical-layer-dependent identification of pad bytes include the following:

1) Code. A distinct control code (distinct from the 256 data codes) identifies a pad byte.
2) Count. Information within the following idle symbol specifies the number of pad bytes.

4.2 Idle symbols

4.2.1 Idle symbol format

The between packet 32-bit idle symbols provide packet framing and control information, as illustrated in figure 15.

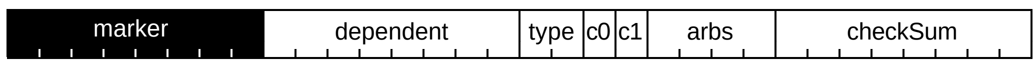

Figure 15—Idle format

The 8-bit ***dependent*** field is physical layer dependent; this value may (for example) be used to communicate synchronization information to the receiver's scrambler circuitry.

The 2-bit ***type*** field value distinctively identifies different types of signaling idles, as specified in table 3.

Table 3—Idle symbol types

Value	Name	Description
0	IDLE_LO	Asynchronous packet or idle follows
1	IDLE_LC	Asynchronous fragment follows
2	IDLE_HI	Synchronous packet or idle follows
3	—	Reserved

The ***c0*** and ***cc*** bits are used to assert circulation control indications, as described in 6.1.

The 4-bit ***arbs*** field communicates asserted and returned ringlet priorities, as specified in 4.2.5.

The 8-bit ***checkSum*** field is the binary exclusive-OR of the previous two data bytes.

4.2.2 Idle symbol framing

Rather than providing an explicit start-of-frame indication, the start of a packet in cycle N is implied by the presence of an idle in cycle N-1 and the presence of a data symbol in cycle N, as illustrated in figure 16. Similarly, the end of a packet in cycle M is indicated by the presence of a data symbol in cycle M and an idle symbol in cycle M.

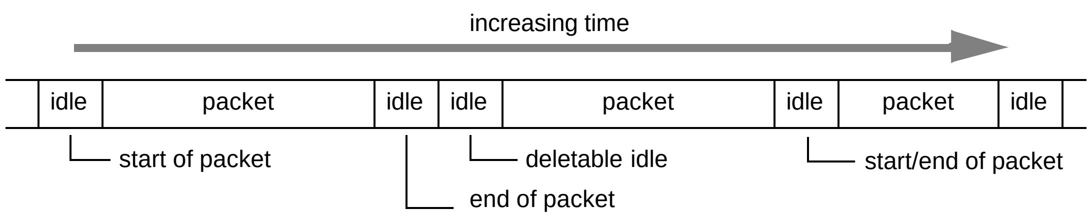

Figure 16—Idle symbol framing

Idles can be inserted and deleted, for idle elasticity (see 6.6) and packet insertion/removal purposes. Idle insertion and deletion protocols are restricted by the following simple rules:

1) Deletion. All but the first idle of a same-type idle-symbol sequence can be deleted.
2) Insertion. Any idle symbol can be replicated any number of times.

4.2.3 Idle framed classification

The two basic idle symbol types, IDLE_LO and IDLE_HI, are used to identify following packetized data as asynchronous or synchronous in nature, as illustrated in figure 17. The packet classification determines whether the packet is destined for the next node's low-priority slowPath or high-priority fastPath queues respectively.

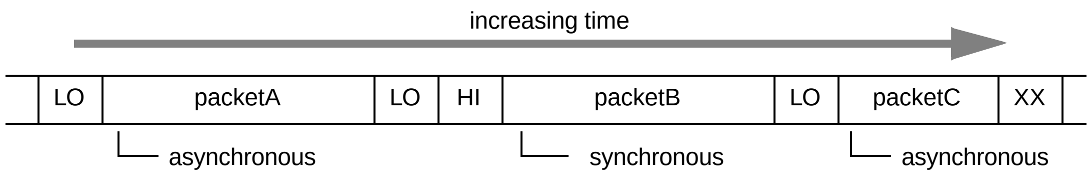

Figure 17—Idle-framed classification

4.2.4 Idle frame preemption

An asynchronous data stream followed a IDLE_LO symbol is used to preempt the asynchronous packet for a synchronous packet transmission, as illustrated in figure 18. The asynchronous transmission continues after the following IDLE_LC symbols.

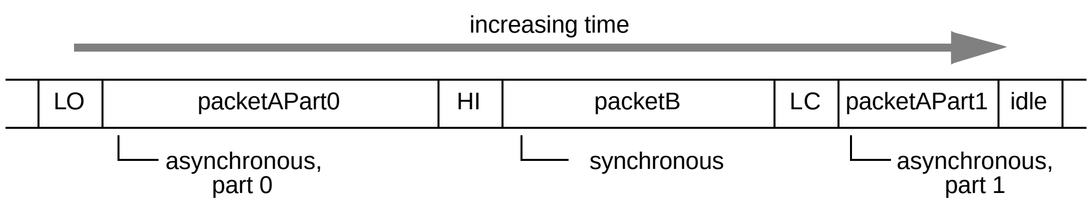

Figure 18—Idle frame preemption

4.2.5 Arbitration (arbs) field values

The 4-bit ***arbs*** field communicates sampled and asserted ringlet priorities, as specified in table 7. Within this table, the *next* value is generated by scrubbing of the *arbs* input (see 6.1) and the *level* value is the priority associated with an outgoing *arbs* value (see 5.3). Within this context, *level* values of 0 through 4 represent lowest through highest priorities respectively.

Table 4—*arbs* values definitions

level	Value	Name	request	response	Description
0	0	ARB_00	-none-	-none-	Deasserted arbitration
2	1	ARB_01	-none-	grant-A	Assertive fair-A grant
	2	ARB_11	request-1	grant-A	Utilized fair-A grant
	3	ARB_21	request-2	grant-A	Captured grant-A grant
	4	ARB_02	-none-	grant-B	Assertive fair-B grant
	5	ARB_12	request-1	grant-B	Utilized fair-B grant
	6	ARB_22	request-2	grant-B	Captured fair-B grant
3	7	ARB_03	-none-	grant-3	Unfair grant
	8	ARB_13	request-1	“	Fair assertion, unfair grant
	9	ARB_23	request-2	“	“
	10	ARB_33	request-3	“	Unfair assertion, unfair grant
4	11	ARB_04	-none-	grant-4	Synchronous grant
	12	ARB_14	request-1	“	Fair assertion, stream grant
	13	ARB_24	request-2	“	“
	14	ARB_34	request-3	“	Unfair assertion, stream grant
	15	ARB_44	request-4	“	Upstream stream assertion

5. Arbitration protocols

Arbitration protocols are based on the policing of offered traffic, based on the class of the traffic and the prenegotiated bandwidths. Higher level protocols are expected to further partition bandwidth restrictions of flows from within a node, based on per-flow service level agreements (SLAs) maintained within that node. However, the use of within-the-node per-flow restrictions is beyond the scope of this standard.

5.1 Attach point queues

A typical attach point has synchronous and asynchronous components, as illustrated in the top and bottom halves of figure 19 respectively. If the synchronous application bandwidths match that of the interconnect and data transfers can be properly staged, synchronous transmit and receive buffers are unnecessary (and therefore have not been illustrated).

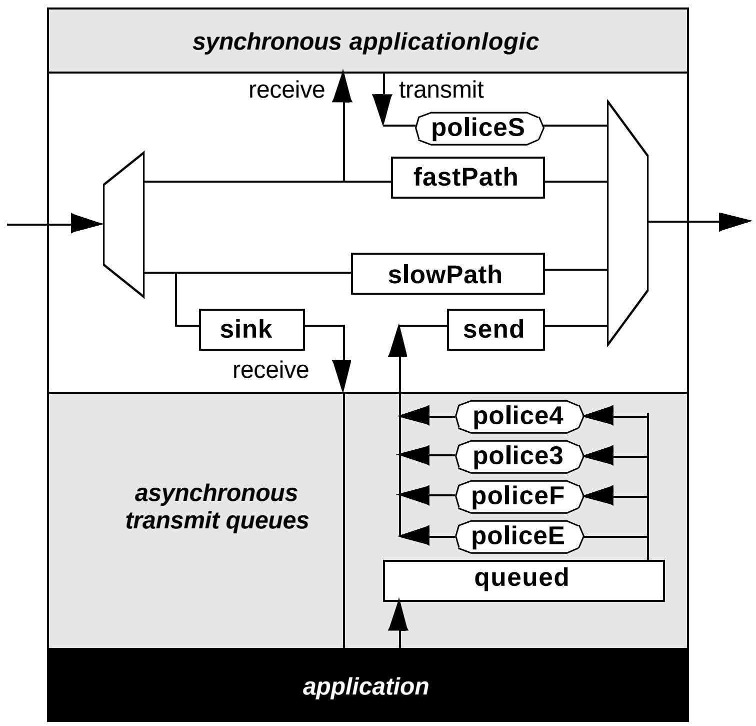

Figure 19—Attach point routing

5.1.1 Bypass FIFOs

The purpose of the *fastPath* buffer is to hold portions of synchronous packets that are received during synchronous packet transmissions. An additional synchronous packet cannot be sent until the *fastPath* has been half emptied. Therefore, a sufficiently sized *fastPath* buffer holds twice the maximum synchronous packet transmitted by this node.

The purpose of the *slowPath* buffer is to hold portions of asynchronous packets that are received during synchronous or asynchronous packet transmissions. Multiple asynchronous packet can be sent before the *slowPath* has been emptied. Therefore, an optimally sized *slowPath* buffer depends on the arbitration parameters and ringlet circulation times, possibly larger than the maximum asynchronous packet transmitted by this node.

NOTE—When used within large metropolitan or small-state environments, the slowPath will be multiple megabytes (not kilobytes) in size. Such buffers are expected to be implemented in high-density high-bandwidth DRAM technologies, not on-chip SRAM technologies.

5.1.2 Rate matching FIFOs

The ***send*** buffer holds an asynchronous packet, so that the transmission of that asynchronous packet can be safely paced in the presence or absence of higher priority synchronous packets. A sufficient ***send*** buffer size equals the size of the maximum asynchronous packet transmitted by this node, but larger sizes may be appropriate to sustain link-bandwidth transmission rates.

Similarly, the purpose of the *sink* buffer is to hold an asynchronous packet, so that the reception of that asynchronous packet can be safely paced in the presence or absence of higher priority synchronous packets. A sufficient ***sink*** buffer size equals the size of the maximum asynchronous packet received by this node, but larger sizes may be appropriate to sustain link-bandwidth transmission rates.

5.1.3 Policing

The ***policeS*** circuit limits the synchronous transmissions to their semi-stable prenegotiated bandwidth limit, averaged over a 1ms interval. Furthermore, absolute policing (no packets are allowed to pass) occurs when the *fastPath* FIFO is over half full.

The ***police4*** circuit limits the asynchronous transmissions to their semi-stable prenegotiated bandwidth limit, averaged over a 1ms interval.

The ***police3*** circuit limits the asynchronous transmissions to their semi-stable prenegotiated bandwidth limit, since the asynchQueue was previously empty. Furthermore, absolute policing (no packets are allowed to pass) occurs when the ringPriority level equals STREAM.

The ***policeF*** circuit limits the asynchronous transmissions to a fixed cumulative amount, equal to one fourth of the ringlet circulation time, sent once during each arbitration interval. Furthermore, absolute policing (no packets are allowed to pass) occurs when the ringPriority level equals STREAM or UNFAIR.

The ***policeE*** circuit limits the asynchronous transmissions to occur in opportunistic (priority=0) intervals.

5.1.4 Selection precedence

The precedence order for transmit packet selection is listed below. This precedence order can be dynamic, because arbitration observations may enable or disable selection mechanisms on a per-cycle basis. Also, policing dynamics allows transmissions decisions to depend on which queues or policed pseudo queues are congested, where congestion is indicated by an over-half-full queue condition.

policeS, the source of prenegotiated synchronous traffic.

fastPath, the source of passing through synchronous traffic.

police4, the source of fine-grained-rate limited asynchronous traffic.

police3, the source of coarse-grained-rate limited asynchronous traffic.

policeF, the source of fair residual asynchronous traffic.

policeE, the source of efficient opportunistic asynchronous traffic.

5.2 Priority assertions

The asserted priorities depend on *halfFull* indications from the queues and policing circuits, as specified in table5.

Table 5—Priority assertion conditions

halfFull indications				asserted priority
slowPath	**queued**	**police3**	**policeF**	
1	—	—	—	STREAM
0	1	1	—	UNFAIR
		0	1	FAIR
			0	NONE
	0	—	0	NONE

5.3 Arbitration controls

Arbitration assert and sense information are placed in ***arbs*** field within idle symbols. The most recently received *arbs* value is processed through *aging*, *assert*, and *observe* components, as illustrated in figure 19. The revised *arbs* value is immediately placed in the following idles, bypassing packets queued within the node's bypass buffers.

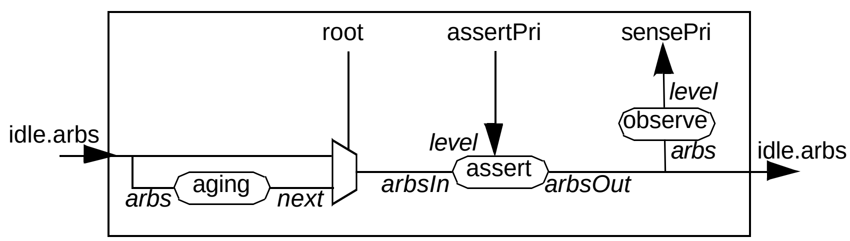

Figure 20—Arbitration controls

Aging of the *arbs* value involves *aging* revisions that are done on only one node (the root node) on each ring. Within this root node, the transformation between *arbs* and *next* values is specified in table 7.

Priority assertions involves transformations of *arbsIn* to produce *arbsOut*, based on the nodes asserted priority *level*, as specified in table 6.

Table 6—Arbitration assertion processing

inputs			output	
level	**arbsIn**	**state**	**arbs**	**state**
NONE	ARB_00, ARB_01, ARB_11, ARB_02, ARB_12, ARB_22, ARB_03, ARB_13,ARB_23, ARB_33, ARB_04, ARB_14, ARB_24, ARB_34, ARB_44	—	arbsIn	—
FAIR	TBDs	TBDs	TBDs	TBDs
UNFAIR	ARB_00, ARB_01, ARB_11, ARB_02, ARB_12, ARB_22, ARB_03, ARB_13, ARB_23, ARB_33	—	ARB_33	—
	ARB_04, ARB_14, ARB_24, ARB_34	—	ARB_34	—
	ARB_44	—	ARB_44	—
STREAM	—	—	ARB_44	—

NOTE—The previous table should be updated to reflect the influence of past state on the currently asserted output value.

Priority sensing involves classification of the outgoing *arbs* value to determine its priority level, as specified in table7.

6. Maintenance activities

6.1 Loop counters

Loop counts are maintained by a distinctive node on the ring, called the root. This allows counts to be updated flexibly, quickly, accurately, and with minimal additional implied ring-propagation latencies. Also, the simplified *ttl* (time to live) adjustments allows CRCs to be simply compensated, rather than recomputed from temporarily unprotected values.

The root is responsible for maintaining loop circulation counts for the benefit of other nodes. These count values are self-calibrating, so that systems continue to work correctly when the cable delays increase or decrease from their previously established values. Three types of counters are maintained in a similar fashion, as illustrated in figure 21.

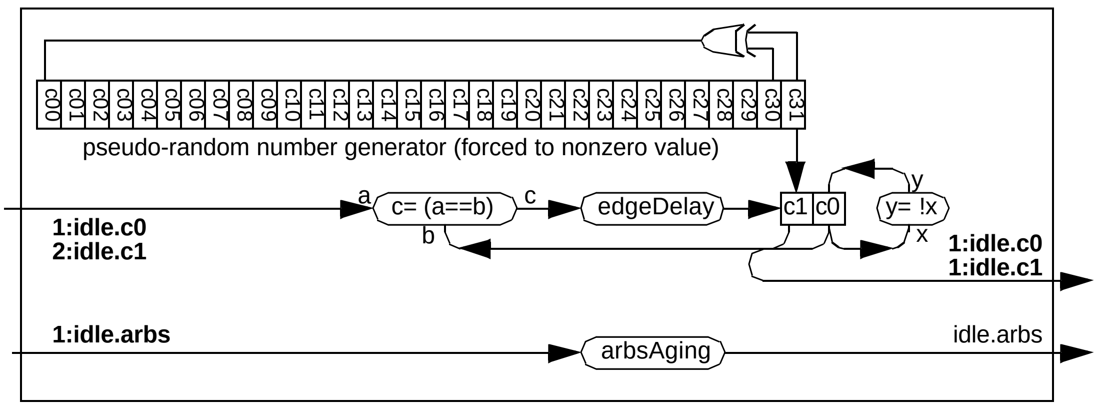

Figure 21—Loop count components

Equality comparisons between the root's internal *c0/c1* values and the observed like-named *idle* values determine when these idle-resident counts have successfully circulated and the *c0/c1* values can be safely updated. The *edgeDelay* circuit inserts an additional latency *L* before the *c0/c1* values are updated; the value of L equals S*N, where S is the maximum packet size and N is the number of nodes on the ring. The intent is to inhibit the onset of restrictive arbitration protocols by artificially increasing the period between *c0/c1* value updates.

*TBD - Verify that this actually has a period of 2**32-1, or pick another generator algorithm that does.*
With the limited 1-bit *c0*-field size, it is theoretically possible to have cyclical sequences of outstanding idle symbols, whereupon *idle[n].c0=0*, *idle[n+1].c0=1*, *idle[n+2].c0=0* and *idle[n+4].c1=1* are concurrently circulating through the ring. To eventually eliminate such cycles, another pseudo-random *c1* value is included in the idle symbols. The *c1* value is generated by a pseudo random number generator generated by a 32-bit *cCount* register that is forced to a nonzero value.

The update rate of the *c0* bit is based on the asynchronous-packet pass-through delays of these values in the attached nodes.

6.2 Packet aging

Multicast and unicast packets are normally scrubbed at their producing and consuming nodes respectively, based on their packet header components. However, packet headers can be occasionally corrupted by transmission errors, which could generate invalid packet-header values.

The packet's *ttl* (time-to-live) field is used to ensure timely scrubbing of such corrupted packet headers, as follows:

1) Transmission, When transmitted, the packet's *header.ttl* value is set to 3.
2) Adjustments. When passing through the root node, nonzero *header.ttl* values shall be decremented.
3) Discarding. When reaching the root node, packets with a zero-valued *header.ttl* shall be discarded.

6.3 CRC aging

Each packet's CRC value is aged when passing through nodes. Aging involves logging the transmission error, then changing the bad CRC value to a distinctive *stomp* CRC value, to inhibit further error logging in downstream nodes. The *stomp* CRC value is defined to be the exclusive-OR of the valid CRC value an a defined STOMP value.

The STOMP value definition is (TBD).

6.4 Arbitration aging

Arbitration indications are “aged” as they pass through the root, as illustrated in table 7.

Table 7—*arbs* values and properties

input		output		
arbs	**state**	**arbs**	**state**	**Description**
ARB_00	—	ARB_00	—	Unasserted indications
ARB_01	—	ARB_00	—	Discarded grant indication
ARB_11	A	ARB_02	B	Initiate grant-B processing
	B	ARB_02	—	Await grant-A completions
ARB_21	A	ARB_01	—	Await grant-A completions
	B	ARB_01	A	Await grant-B completions
ARB_02	—	ARB_00	—	Discarded grant indication
ARB_12	A	ARB_02	—	Await grant-B completions
	B	ARB_01	—	Initiate grant-A processing
ARB_22	A	ARB_01	—	Await grant-A completions
	B	ARB_02	—	Await grant-B completions
ARB_03	—	ARB_00	—	Discarded grant indication
ARB_13, ARB_23	A	ARB_01	—	Reassert grant-A indication
	B	ARB_02	—	Reassert grant-B indication
ARB_33	—	ARB_03	—	Synchronous request is granted
ARB_04	—	ARB_00	—	Discarded grant indication
ARB_14, ARB_24	A	ARB_01	—	Reassert grant-A indication
	B	ARB_02	—	Reassert grant-B indication
ARB_34	—	ARB_03	—	Unfair request is granted
ARB_44	—	ARB_04	—	Synchronous request is granted

6.5 Wallclock synchronization

NOTE—The following wallclock synchronization protocols are preliminary in nature. Further details describing the use rate-adjustments and time-set overrides are required.

With bidirectional cables, the clockSync transmissions can account for the constant cable-induced delays, by measuring round-trip cable delays. Using such techniques, the accuracy of these wallclock synchronization protocols is dependent on the delay differences between incoming and outgoing links, not the overall delay of either. Implementation of these wallclock synchronization protocols involves monitoring the arrival and departure time of synchronous packets, called clockSync packets, as described in this subclause.

Each parent is responsible for measuring the propagation time through its child, by measuring the observed clockTick symbol times and comparing these times to the following clockSync-packet-specified time, as illustrated in figure 22.

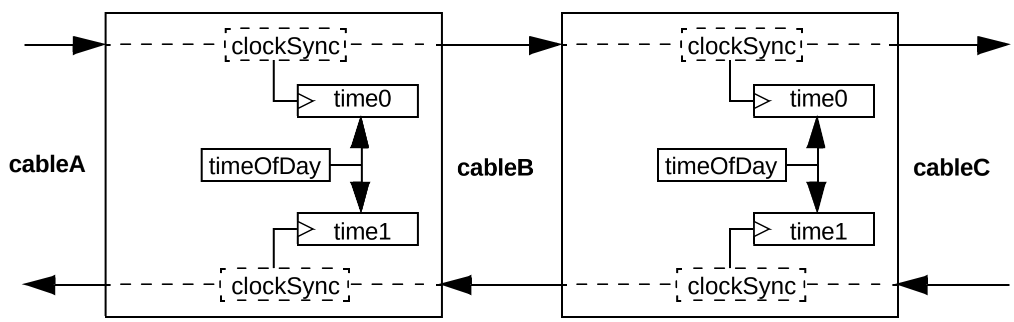

Figure 22—Cable-delay computations

Before cycle N, each node captures the times at which the clockTick symbol is observed on primary and secondary runs. Thus, the values of time0 and time1 are captured in that order. From these internal values, each node is responsible for computing and saving its observed *localTime* value, as specified in equation 2.

`localTime= (time0+time1)/2;` (2)

During cycle N, each node adjusts its *localTimeOfDay* clock based on the *packetTimeOfDay* value observed in either of the clockTime packets, as specified in equation 3.

`localTimeOfDay+= (packetTimeOfDay-localTime)/2;` (3)

These adjustments assume that the upstream reference node sets *packetTimeOfDay* in cycle N to (*time0*+*time1*)/2, where the *time0* and *time1* values were observed in cycle N.

6.6 Idle symbol pools

NOTE—The following elasticity-control protocols are preliminary in nature. Alternative approaches, that utilize the existing bypass FIFO structures, are being considered.

Each node has elasticity buffers that may insert or delete passing through idle symbols, based on the differences between their clock frequencies and the clock frequencies of their transmitting neighbors. To ensure successful operation of these elasticity buffers, idle symbol are managed in several ways, as follows:

1) Sending. When transmitting packets, an replicated idle (in addition to the packet’s EOF idle) is periodically sent (this effectively extends the length of some packet by an additional idle symbol).
2) Recovering. While emptying the bypass FIFO, a periodic stream of deletable idles passes through.
3) Deletion and insertion. Elasticity buffers attempt to fix the distance between deletable idles.

The first two tasks (sending and recovering) affect the send sequencer’s operation, as described in xx. The elasticity buffer’s deletion and insertion constraints are described below.

The default distances between idles can increase if elasticity buffers in multiple nodes concurrently delete passing-through idles. This has the temporary undesirable effect of temporarily starving other elasticity buffer from desired deletable idles, and could (if allowed) cause elasticity buffer failures.

To minimize the worst-case idle-packet starvation conditions, the dynamics of the elasticity buffer are constrained, as illustrated in figure23. Desired deletions are normally deferred until consecutive deletable idles are encountered, or an elasticity failure is imminent. Similarly, desired insertions are normally deferred until an isolated nondeletable idle is encountered, or an elasticity failure is imminent.

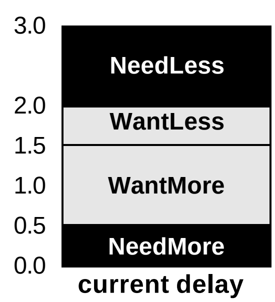

Figure 23—Elasticity delay-adjustment ranges

The elasticity buffer behaves as a variable-delay storage element. Its idle-insertion and idle-deletion decisions are based on the current elastic-buffer delay, as follows:

1) NeedMore. When the delays are very short (less than .5 quadlets), idles are inserted at the next opportunity. The intent is to increase the elastic delay before the 0-delay-limit is reached.
2) WantMore. When the delays are short (.5-to-1.5 quadlets), idle insertions are performed if more than IDLE_SPACING symbols have been received since the last passing-through deletable idle. The intent is to create an evenly-spaced set of deletable idles.
3) WantLess. When delays are long (1.5-to-2.0 quadlets), idle deletions are performed if less than IDLE_SPACING symbols have been received since the last passing-through deletable idle. The intent is to leave an evenly-spaced set of deletable idles, for more-desperate nodes.
4) NeedLess. When the delays are very long (more than 2.0 quadlets), the next deletable idle is deleted. The intent is to reduce the elastic delay before the 3-delay-limit is reached.

The IDLE_SPACING constant value is physical-layer dependent and beyond the scope of this standard.

7. Ring initialization

NOTE—The following initialization protocols are preliminary in nature. Detailed state machines, indication types, and indication codings are TBD.

In the absence of a configurable ringlet, ring initialization degrades to an spanning tree algorithm: spanning tree is formed and the ring is constructed by circumscribing this tree. These protocols support nearly arbitrary cable topologies; extra cables may be inserted for redundancy or performance reasons, or may be installed by an uninformed user.

In the presence of a ring topology, ring initialization deviates from the 802-defined spanning tree algorithm, in its support of ringlet connections, as follows:

1) Dual ringlets. A ring is normally closed, enabling efficient use of counter-rotating ringlets.
2) Single ringlet. An defective ring can be closed, enabling efficient use of a single ringlet.

Dual ringlets deliver more than twice the performance, because packets can travel on either cable an in a preferential direction. Thus, ring cable topologies are not only possible, they are highly desirable from a fault-tolerance perspective.

7.1 Bus reset functionality

Resets begin with the detection of a reset condition (typically the addition or removal of a cable or device), which forces one or more of the nodes into a “contending root” condition. A contending root is responsible for distributing its EUI-64 (an extended unique identifier) to its adjacent neighbors, by including this EUI-64 in reset packets sent to its neighbors. The contending root is also responsible for exploring the interconnect, and changes to a branch or leaf when a larger EUI-64 is observed.

The contending root waits for its reset packets to return. If its reset returns on the outgoing-reset port, the reset is sent out the next (previously inactive) port. Alternatively, if the reset returns on a different port, that port is set to a loop-back mode. Using this algorithm, the resets propagate through the interconnect, establishing its active topology, as illustrated in the left half of figure 24.

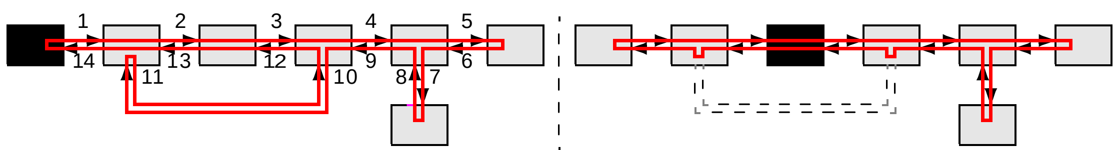

Figure 24—Bus reset sequences

A reset initializes the bus and bypasses ports that are identified as unusable or redundant. Port are bypass involves electronically-switched circuits within the node; no relays or termination-plug components are required.

A bus reset can be initiated by any node, but is completed by the node that was selected to become the root. The root node is selected from among the root-capable nodes, based on the node’s root-selection bits (which can be set by software) and its *eui64* identifier (which is fixed when the node is manufactured).

 Dr. David V. James
P802.17 Working Group Member

Resets have the following properties:

1) Autonomous. The system is self configuring, yielding one of the following two configurations:
    a) Efficient. When a daisy-chained loop is reset, an active counter-rotating ringlet is formed.
    b) Sufficient. When other arbitrary topologies are reset, a (less efficient) active ringlet is formed.
2) Simple. Reset protocols are design to be easily implemented, in the following ways:
    a) Two-port optimized. The reset protocols are optimized for simple 1-port and 2-port nodes.
    b) Simple compares. A node only compares incoming reset-packet identifiers to its own.
    c) Self timed. The protocols are speed insensitive, so low-speed compare logic can be used. However, processing delays are constrained to reduce the worst-case bus reset latencies.
3) Repeatable. A reset consistently yields the same root and active-cable topology, unless the physical cabling topology has actually changed.

7.1.1 Reset identifiers

Each node is assumed to be manufactured with a 64-bit unique identifier (EUI-64). The *resetId* is formed by concatenating a software precedence field to this EUI-64 value, forming a 72-bit identifier. The node with the largest *resetId* is selected to become the root; each child can identify its parent by the port on which the largest *resetId* value was last observed. By convention, the lowest *resetId* (corresponding to all zeros) is asserted by non-root capable nodes, or by root-capable nodes that are currently resetting other links.

Although each node is mandated to have an EUI-64 (see xx), only the root-capable nodes need to manage these values during bus reset sequences. Dual ported nodes that are not root capable have to forward observed resets through them, but have no need to compare passing through *resetIds* to their own.

7.1.2 Reset packets

Several forms of reset packets are distributed during a bus reset, as listed below:

TBDs

When reset and flush packets are being sent, they are sent with only a few interspersed idles. A small number of idles are necessary to provide idle-deletion opportunities. Too many idles would make it difficult to detect absence-of-reset conditions, due to dead or disconnected nodes.

7.2 Ringlet initialization effects

7.2.1 Duplex ring initialization

As example, a duplex ringlet is configures after a cable is plugged into two previously unconnected ports, forming a ring, as illustrated in figure 25. The connection of a cable allows each of the attached ports to observe each others heartbeats, quickly responding to the newly attached cable. The cable connected state is quickly reported by the ports on nodes *b* and *d*, which initiates their reset activities. The aftermath of the reset-invoked initialization changes the configured node topology, by forming more efficient dual ringlet connections.

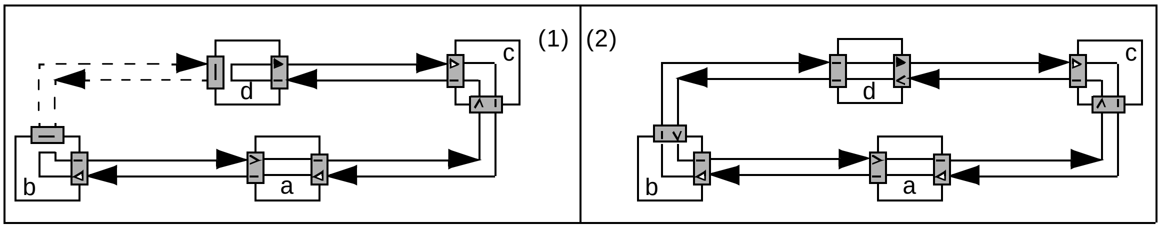

Figure 25—Duplex ring reset

7.2.2 Daisy chain initialization

As another example, a cable is removed from two of the previously connected ports, severing a ring, as illustrated in figure 26. The cable deletion initiates a bus reset; the aftermath of the bus resets changes the configured node topology, forming less efficient single ringlet connections.

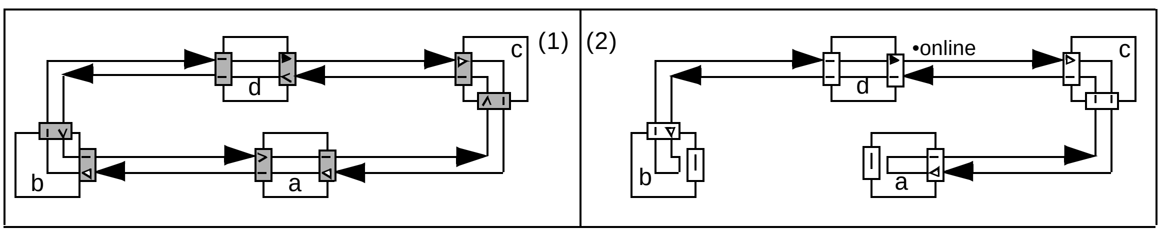

Figure 26—Daisy chain formed

7.2.3 Primary link fault-induced reset

As another example, the secondary link within the duplex cable may fail, as illustrated in figure 27. The link failure initiates a bus reset; the aftermath of the bus resets changes the configured node topology, forming less efficient single ringlet connections.

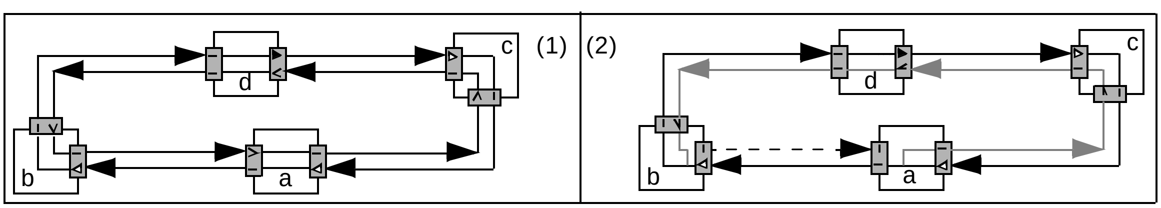

Figure 27—Primary link fault-induced reset

7.2.4 Secondary link fault-induced reset

As another example, the secondary link within the duplex cable may fail, as illustrated in figure 27. The link failure initiates a bus reset; the aftermath of the bus resets changes the configured node topology, forming less efficient single ringlet connections.

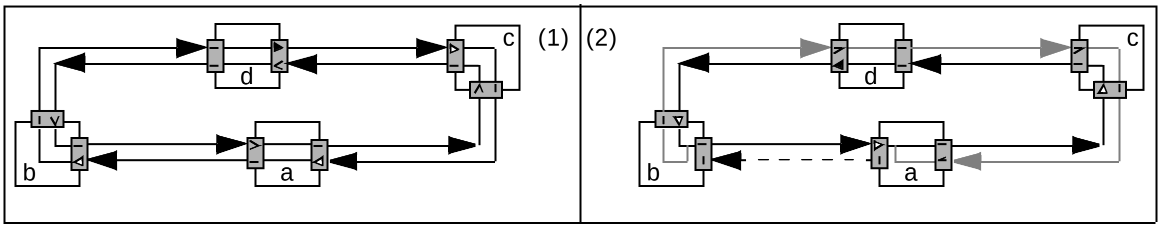

Figure 28—Secondary link fault induced reset

7.2.5 Forced tree reset

As another example, the side node may be attached to an existing ring, as illustrated in figure 29. The cable attachment initiates a bus reset; the aftermath of the bus resets changes the configured node topology, forming less efficient circumscribing ringlet connections.

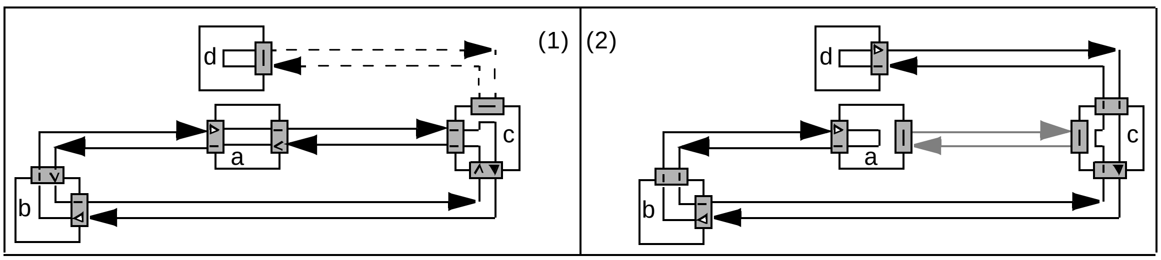

Figure 29—Forced tree reset

7.2.6 Redundant ring recovery

As another example, an active/redundant cable may be severed or removed, as illustrated in figure30. The cable detachment initiates a bus reset; the aftermath of the bus resets changes the configured node topology, forming an alternative (but equally efficient) dual-ringlet connections.

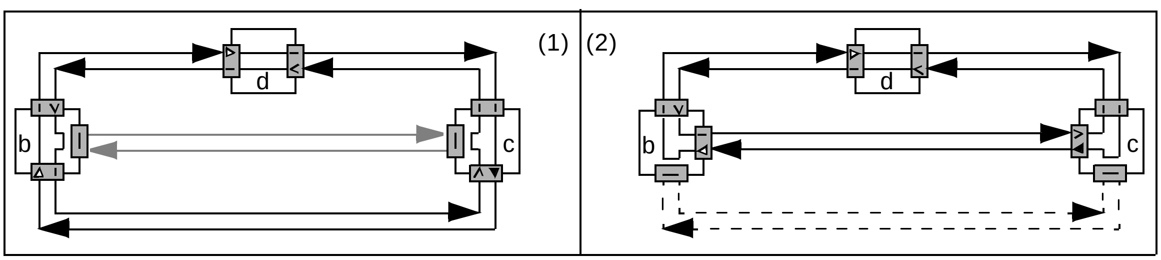

Figure 30—Redundant ring recovery