802.17 presentation

- Prepared for Orlando, May2001
- Dr. David V. James
Chief Architect
Lara Networks
110 Nortech Parkway
San Jose, CA 95134
+1.408.942.2010
dvj@alum.mit.edu

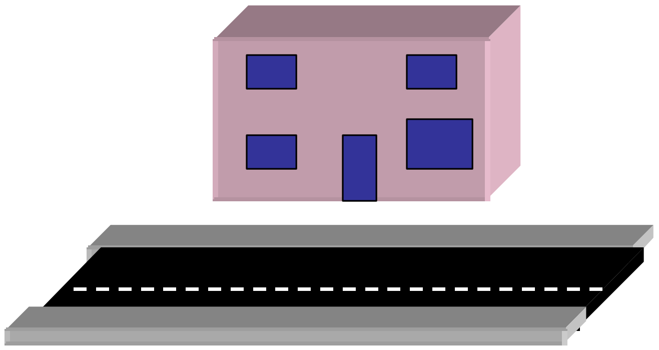
Lara Networks

Basic issues

- *Limits of scalability*
- Supported topologies
- Packet formats
- Transport services
- Arbitration
- Initialization (plug & play)

Lara Networks

Limits of scalability



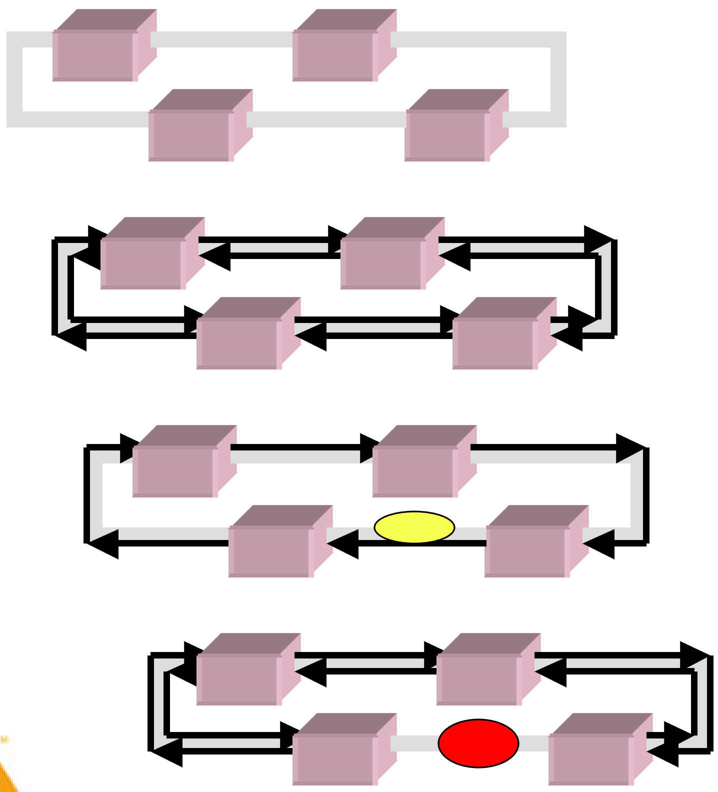
- Geosynchronous
  - Terrestrial
    - The metro area
      - To the curb
        - To the home

Lara Networks

Basic issues

- Limits of scalability
- *Supported topologies*
- Packet formats
- Transport services
- Arbitration
- Initialization (plug & play)

Supported topologies



- A physical ring
- Dual ringlets
- Single ringlet
- Duplex ringlet

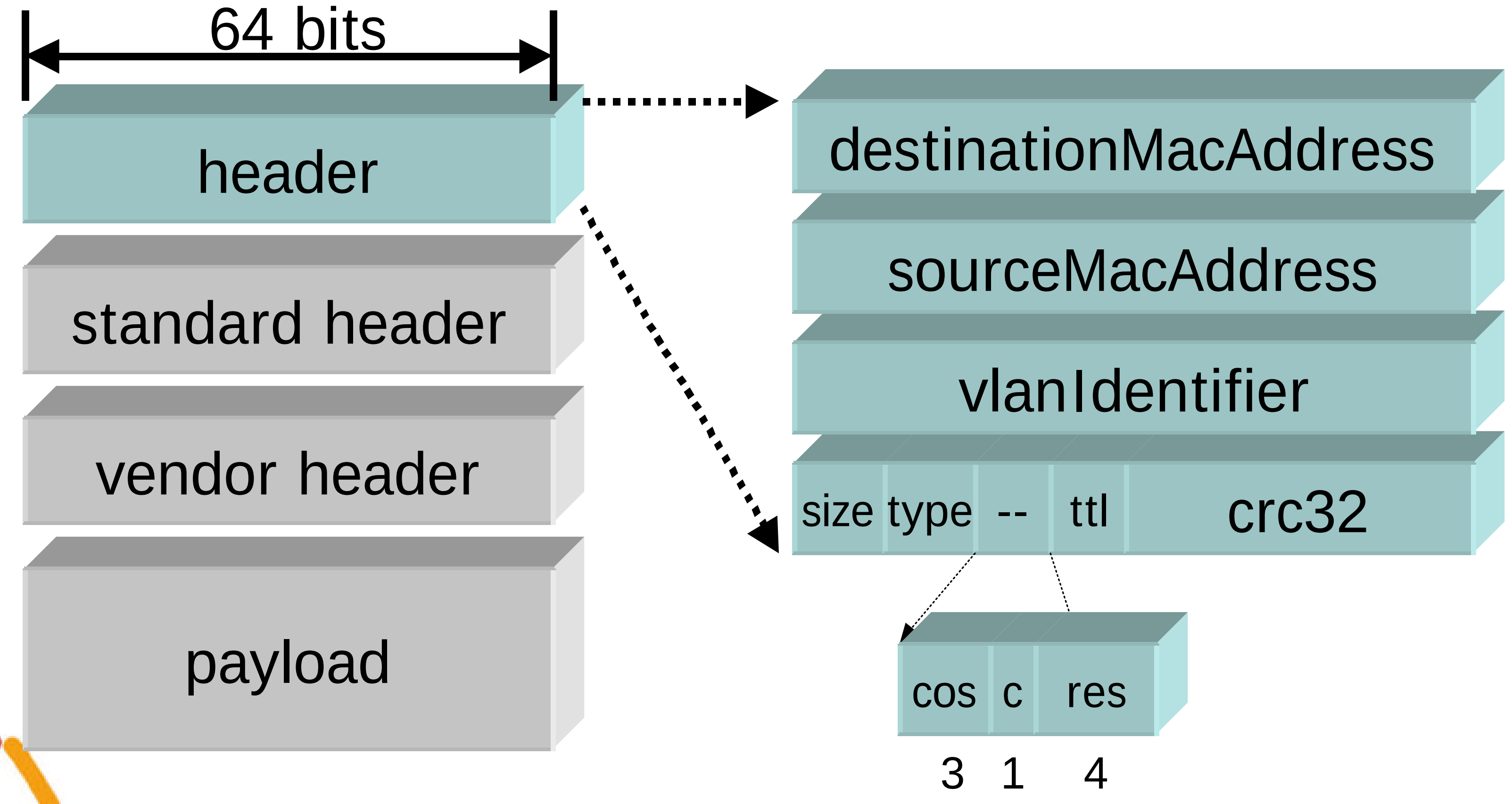
Lara Networks

Basic issues

- Limits of scalability
- Supported topologies
- *Packet formats*
- Transport services
- Arbitration
- Initialization (plug & play)

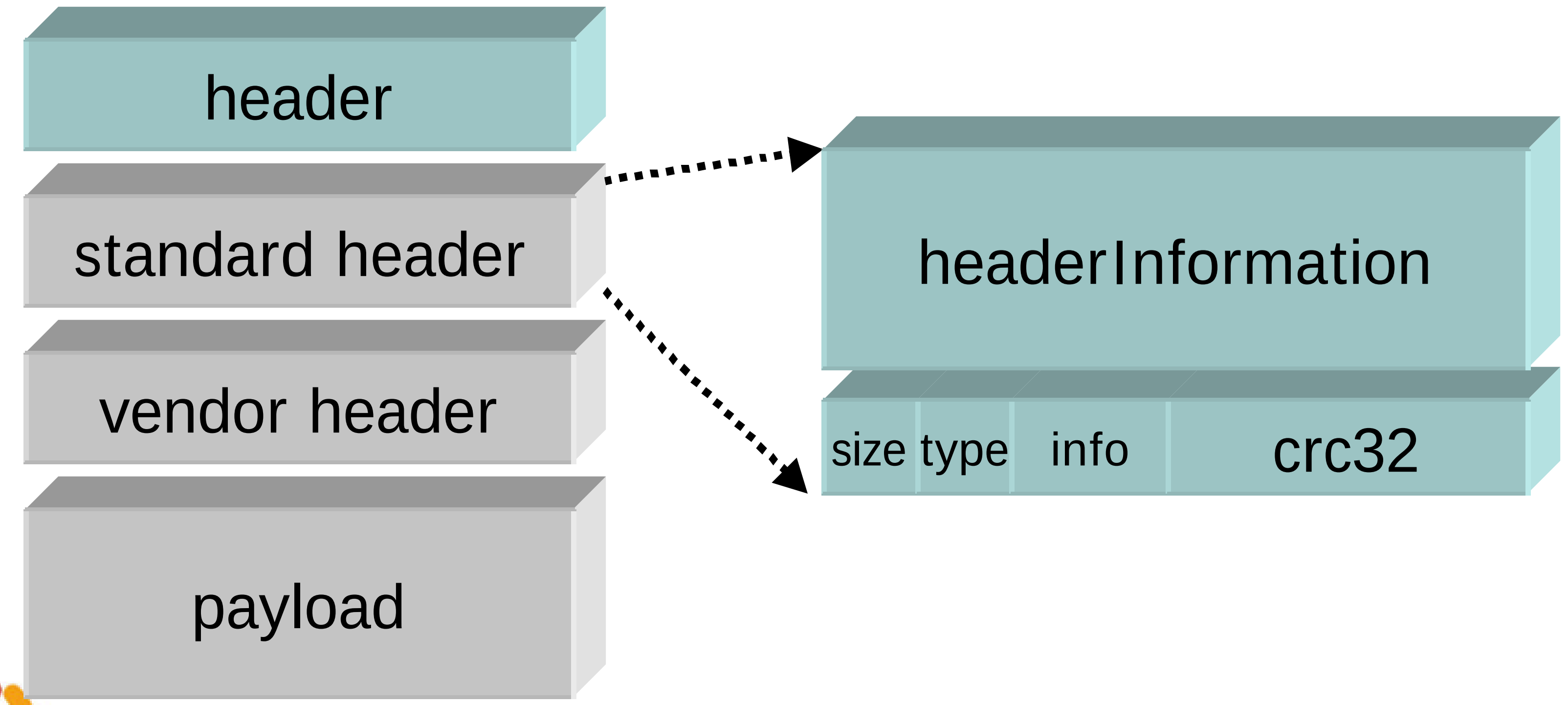
Lara Networks

Packet header



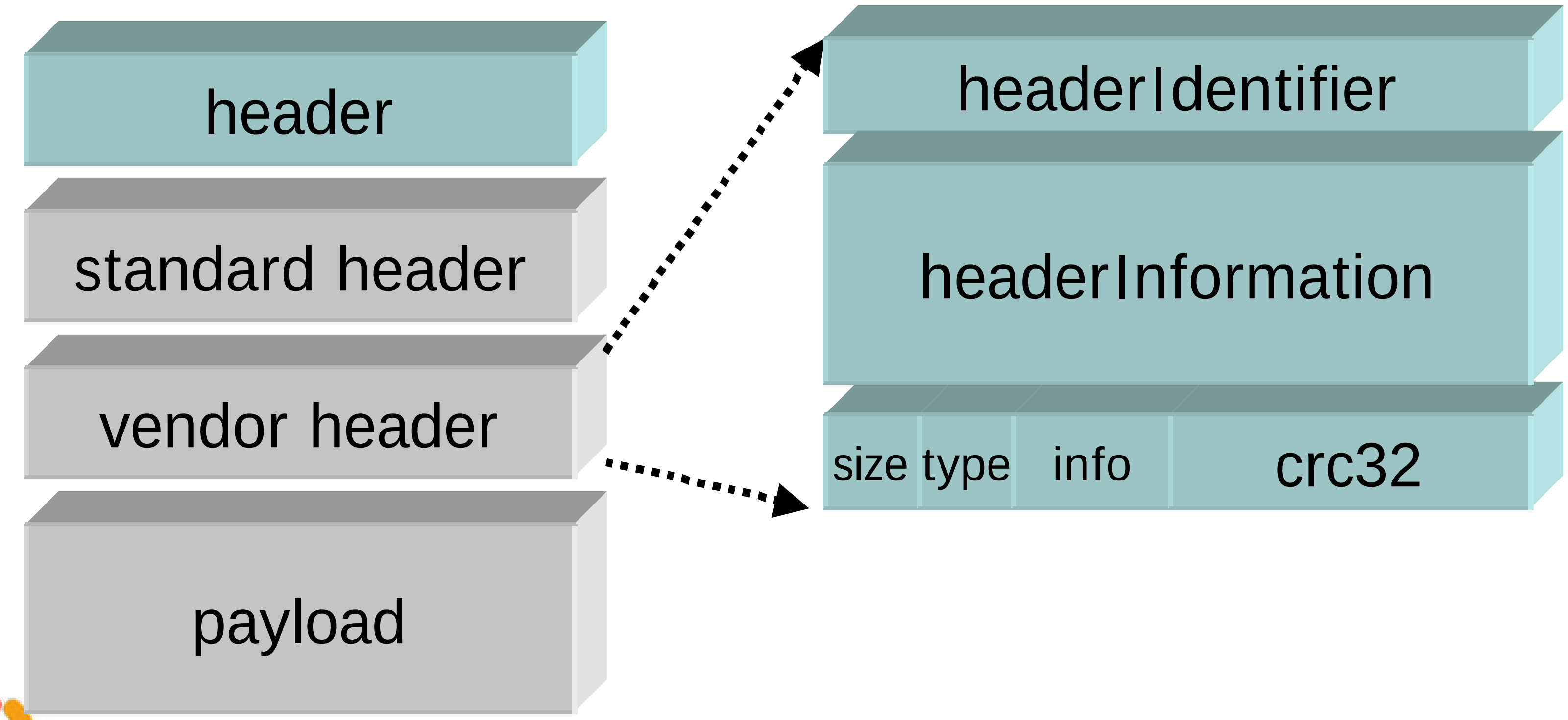
Lara Networks

Standard extended header



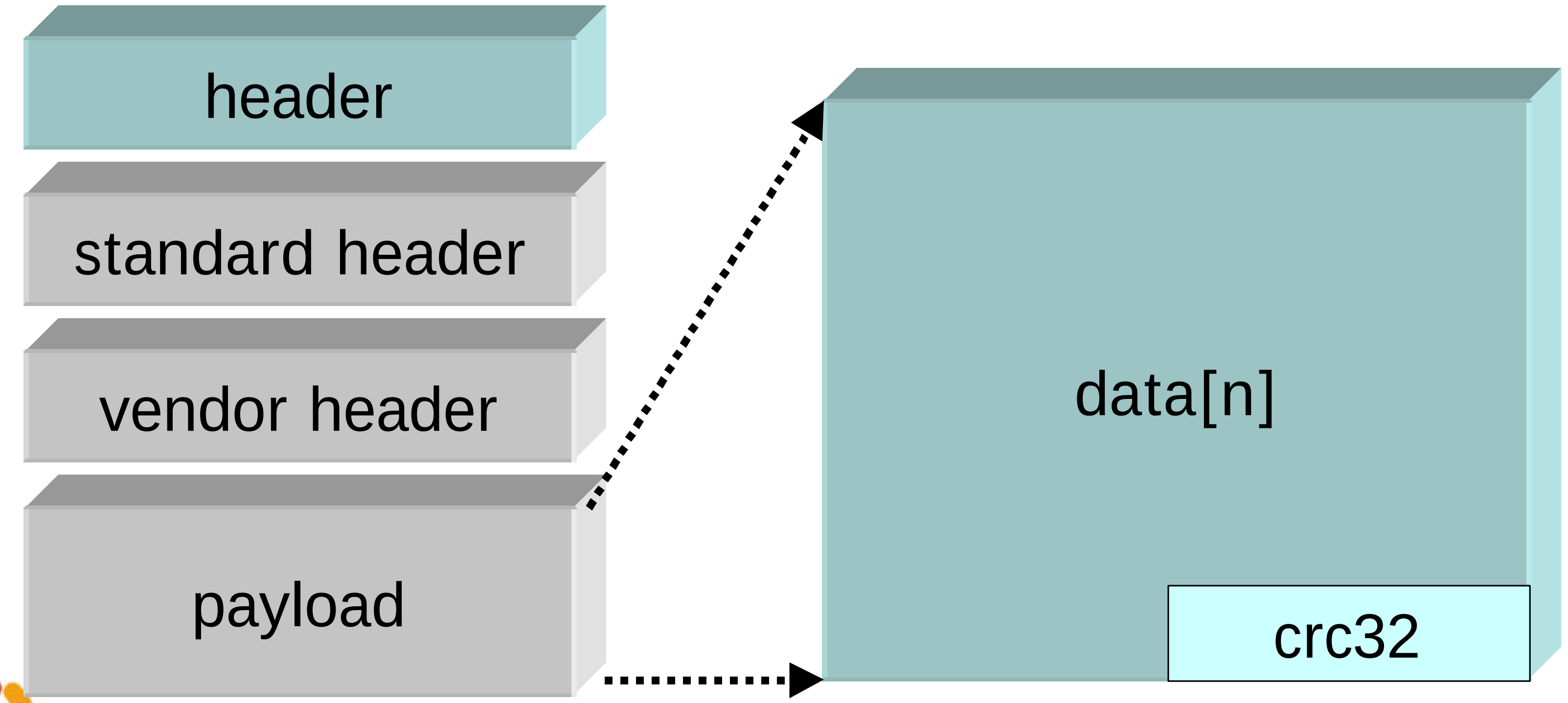
Lara Networks

Vendor dependent header



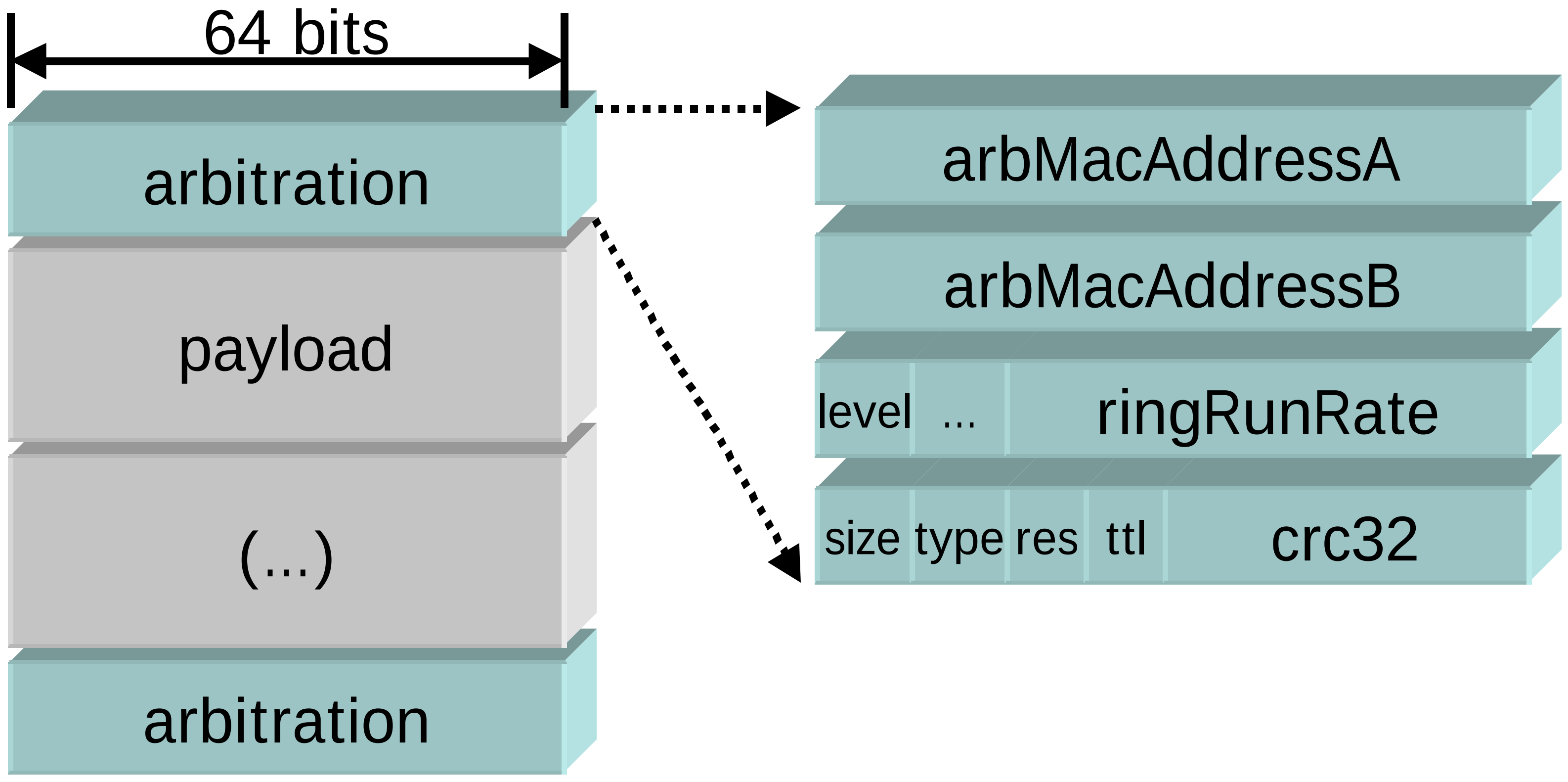
Lara Networks

Payload



Lara Networks

Arbitration packets



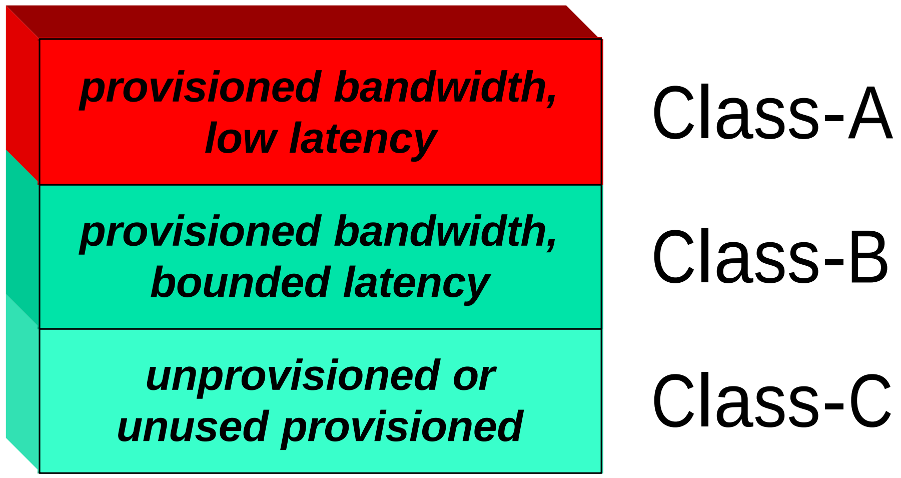
Lara Networks

Basic issues

- Limits of scalability
- Supported topologies
- Packet formats
- *Transport services*
- Arbitration
- Initialization (plug & play)

Lara Networks

Arbitration classes


provisioned bandwidth, low latency
Class-A
provisioned bandwidth, bounded latency
Class-B
unprovisioned or unused provisioned
Class-C

Lara Networks

Basic issues

- Limits of scalability
- Supported topologies
- Packet formats
- Transport services
- *Arbitration*
- Initialization (plug & play)

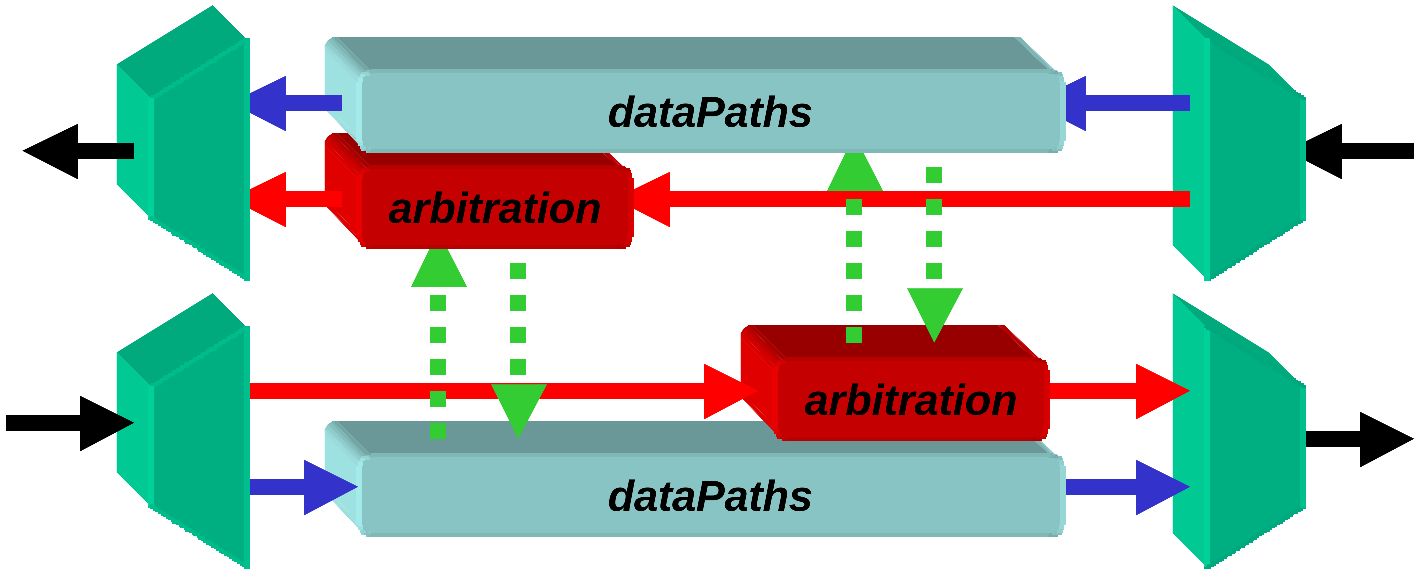
Lara Networks

Lessons of the past...

- Flow control mandates 2-out-of-3
  - Low latency transmissions
  - Fair bandwidth allocation
  - High bandwidth utilization
- Feedback control systems
  - Low latency signaling
  - Control can pass class-B/C packets
  - Separate class-A queue is utilized
- Other observations
  - Local control => global perversions
  - Fairness is inherently “approximate”
  - Strange beating sequences DO OCCUR

Lara Networks

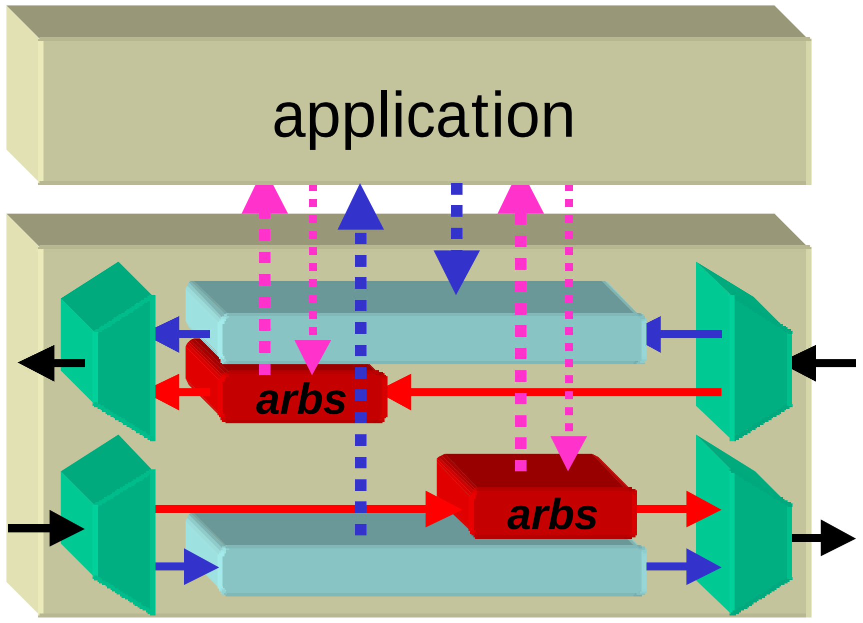
Internal MAC arbitration signals



- Arbitration affects opposing run
- My congestion affects upstream node
- Downstream congestion affects me

Lara Networks

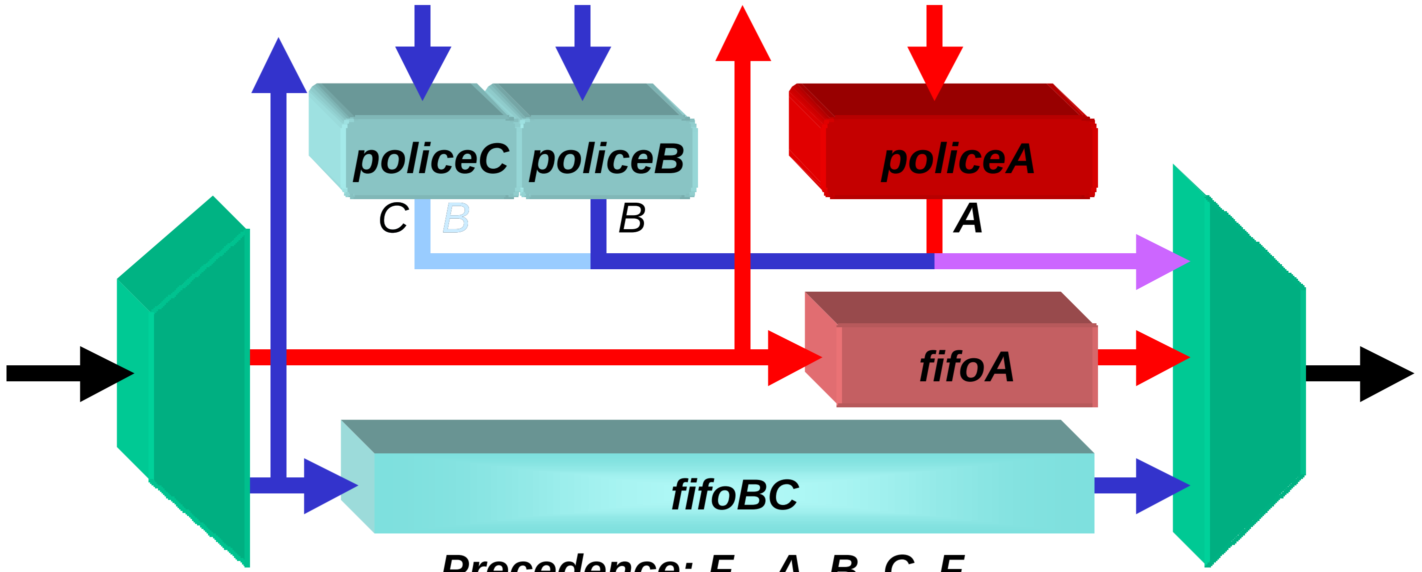
External MAC arbitration signals



- MAC receives application information
  - MAC FIFOs are $$, latency++, inflexible
- Application receives MAC information
  - Allows reordering and run selection

Lara Networks

Arbitration related components

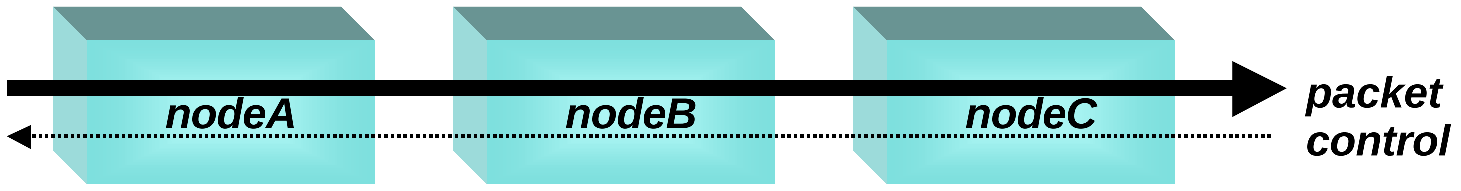


Precedence: F$_a$, A, B, C, F$_{bc}$

- Distinct class-A & class-B/C paths
- Load dependent policing

Lara Networks

Opposing arbitration



- Data packets flow in one direction
- Arbitration control flows in the other*

Lara Networks

Level dependent transmissions

level	depth	behind	row	enabled
>depth	—	—	1	Fa, A
—	≥3/4	—	2	Fa, A, Fbc
	>1/2	—	3	Fa, A,B, Fbc
	<1/2	0	4	
		1	5	Fa, A,B,C, Fbc
	0	—	6	

Lara Networks

Arbitration summary

- Dual levels
  - Class-A, pre-emptive low latency
  - Class-B, less latency sensitive
- Jumbo frames
  - Affect asynchronous latencies
  - NO IMPACT on synchronous latency
- Cut-through vs store-and-forward
  - Either should be allowed
  - Light-load latency DOES matter

Lara Networks

Basic issues

- Limits of scalability
- Supported topologies
- Packet formats
- Transport services
- Arbitration
- *Initialization (plug & play)*

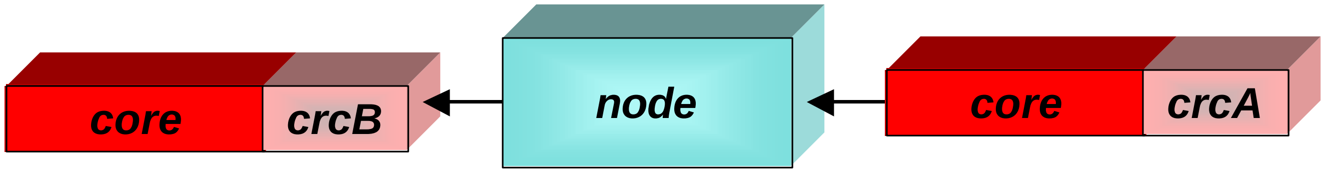
Lara Networks

Initialization

- TBD...
-
-
-
-
-

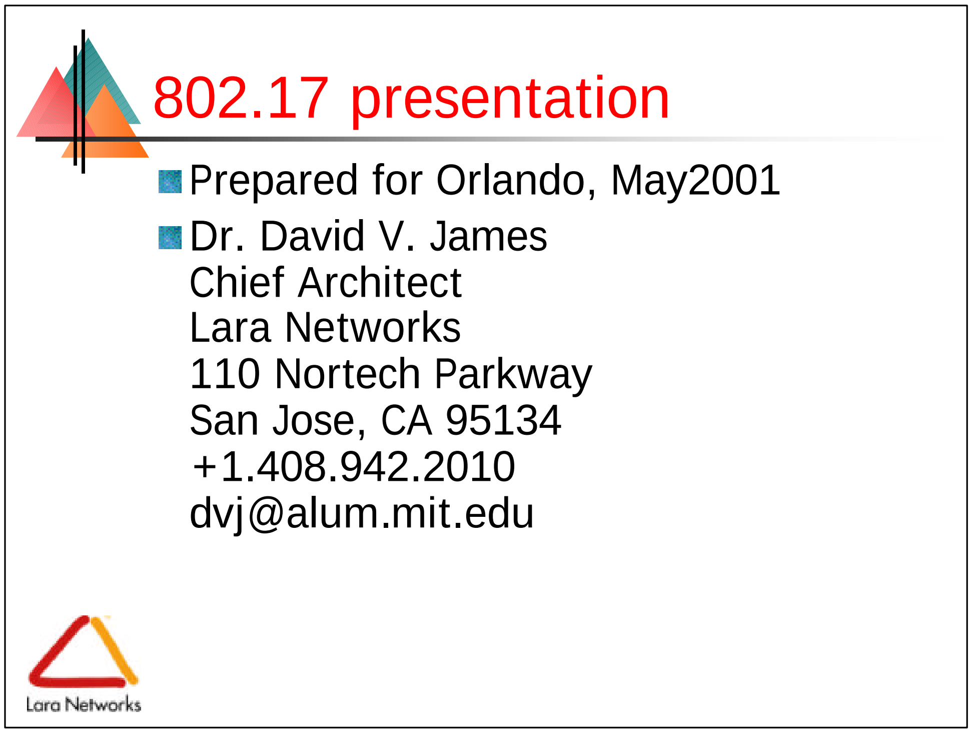
Lara Networks

Cut-through CRCs



- Corrupted packet remains corrupted
- Error logged when first detected
- if (crcA!=crc&&crcA!=crc^STOMP) {

```
    errorCount+= 1;
    crcB= crc^STOMP;
}
```



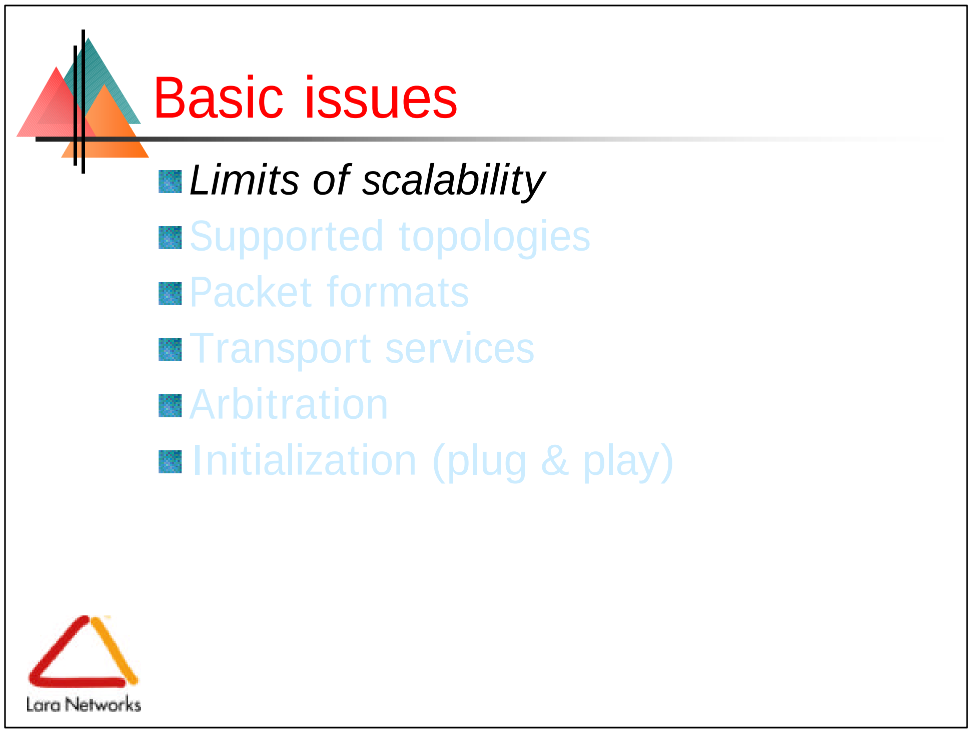
802.17 presentation

- Prepared for Orlando, May2001
- Dr. David V. James
  Chief Architect
  Lara Networks
  110 Nortech Parkway
  San Jose, CA 95134
  +1.408.942.2010
  dvj@alum.mit.edu

Lara Networks

Lara Networks makes TCAMs for the internet.

The author, David V. James, has been involved with point-to-point interconnects since the infancy of SCI in the early 1990s.



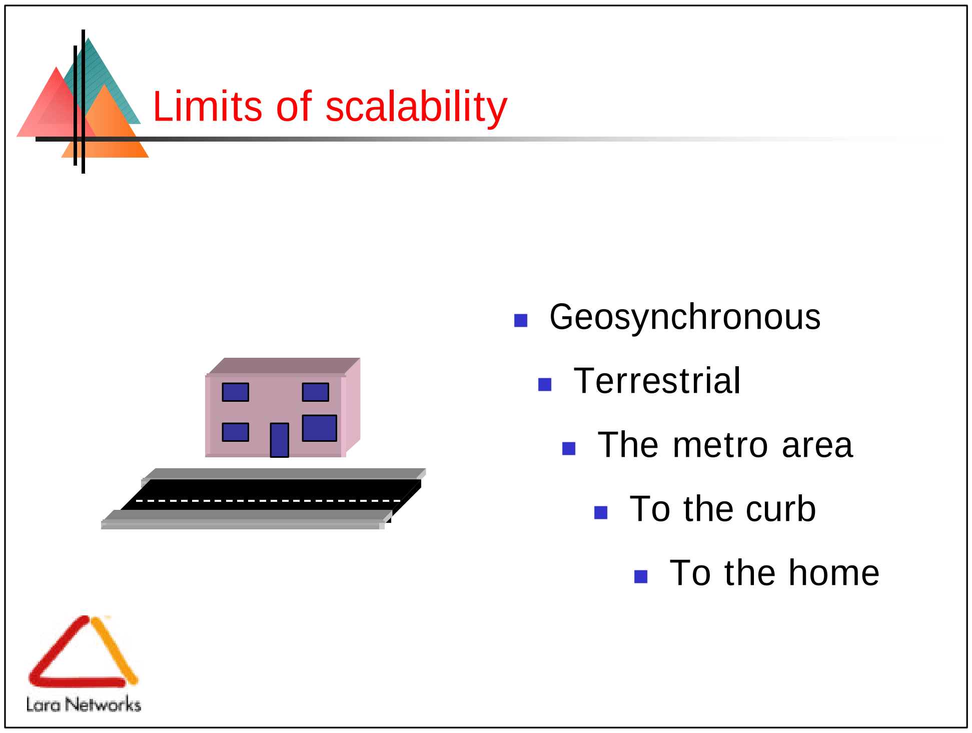
Basic issues

- *Limits of scalability*
- Supported topologies
- Packet formats
- Transport services
- Arbitration
- Initialization (plug & play)

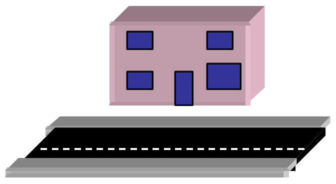
Lara Networks

Distances are long enough that most of the nonscalable solutions don’t work.

Although it may be harder to find the truly scalable solutions, the scalable solutions are likely to be the best ones (even if hard to find).



Limits of scalability



- Geosynchronous
  - Terrestrial
    - The metro area
      - To the curb
        - To the home

Lara Networks

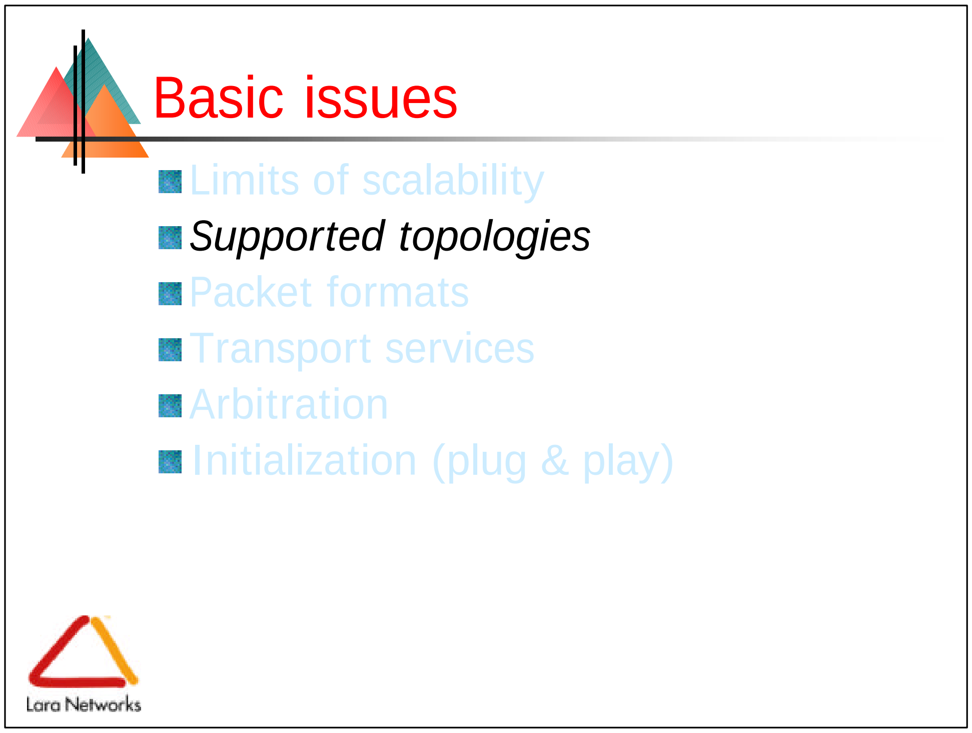
The robustness, plug-and-play, and deterministic provisioned bandwidth make 802.17 useful in the home, as well as the metropolitan area.

To transport telephony and video traffic, the gap between the home and MAN (the first mile) should also be handled, or this becomes the weak link.

Of course, the MAN is our primary focus.

Longer undersea terrestrial connections could have a similar distance/bandwidth parameters, so the same solution may be applicable.

Since there is limited outer-space communications, the limit would probably be geosynchronous applications.

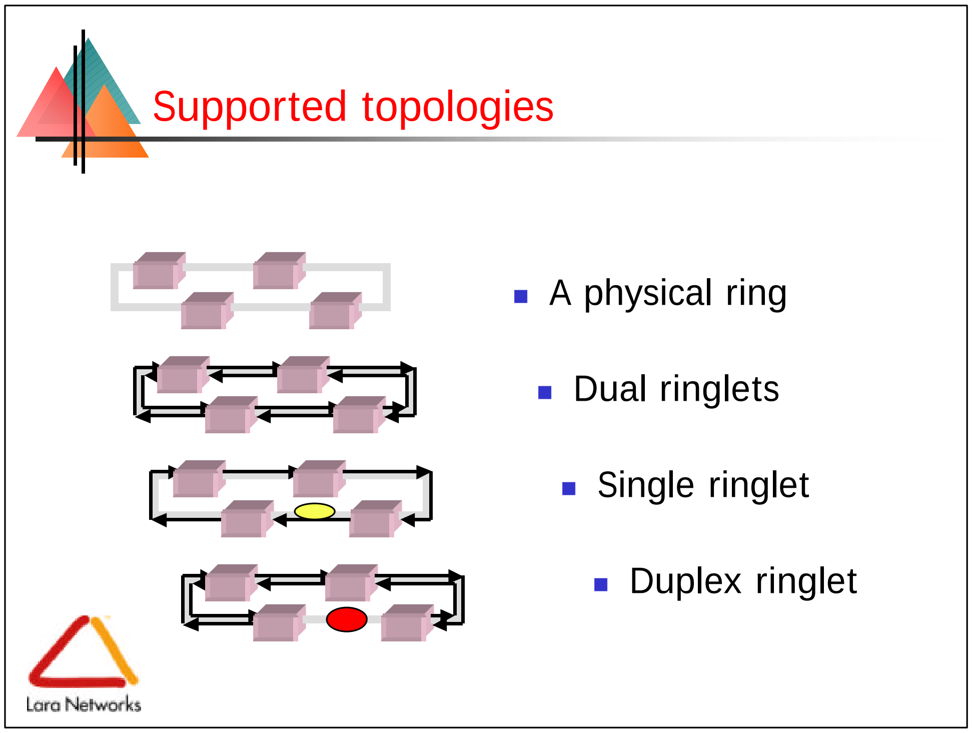


Basic issues

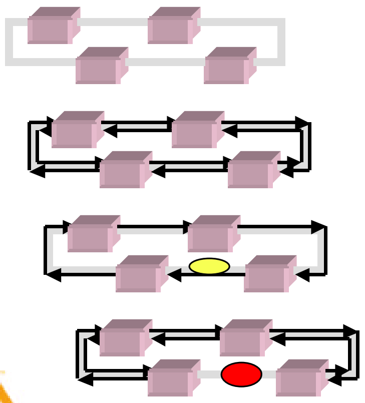
- Limits of scalability
- *Supported topologies*
- Packet formats
- Transport services
- Arbitration
- Initialization (plug & play)

Lara Networks

Supported physical topologies are limited, so that the duplex ring can be quickly and well defined…



Supported topologies



- A physical ring
- Dual ringlets
- Single ringlet
- Duplex ringlet

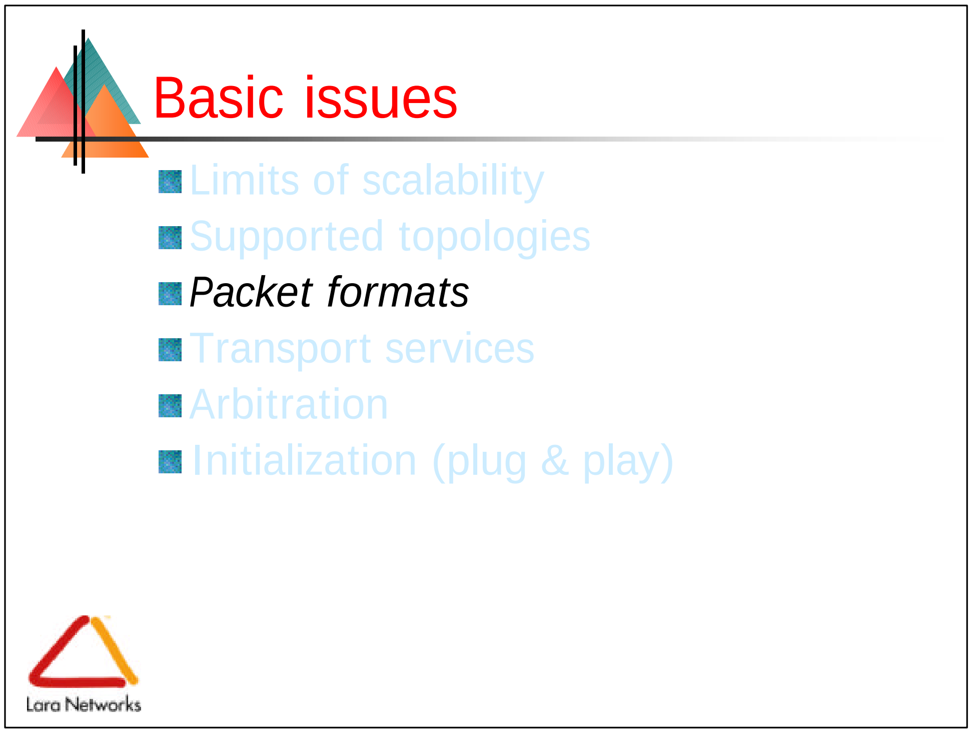
Lara Networks

The physical ring is the base cabling model.

However, each cable is assumed to provide a full-duplex connection. Thus, the dual ringlet model is fundamental for the normal operation.

A component (e.g. laser) failure may force this into a single ringlet configuration. This should be supported, as the other (nonfailed) link remains useful.

The duplex ringlet, of course, is necessary to support communications after a cable (e.g. backhoe cut) failure.



Basic issues

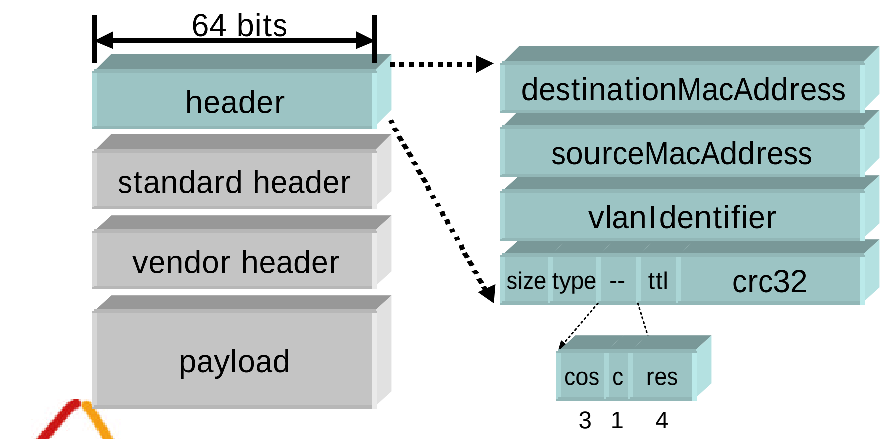
- Limits of scalability
- Supported topologies
- *Packet formats*
- Transport services
- Arbitration
- Initialization (plug & play)

Lara Networks

Packet framing is likely to be physical-layer dependent.

However, the packet formats themselves are useful to be defined, independent of the physical link details.

Packet header



Lara Networks

High-speed signaling is likely to be done on a 64-bit (or wider) granularity.

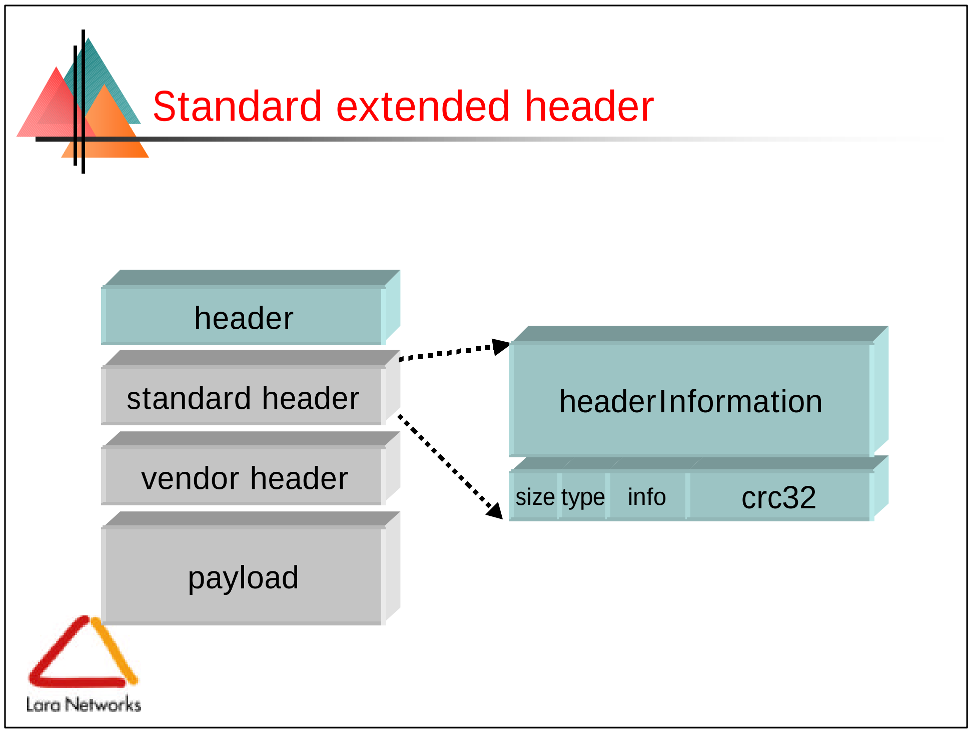
This granularity encourages use of a 64-bit destinationMacAddress; other reasons include:

a) New applications of (potential) high volume are recommended to use these, so that the number space will not be quickly consumed.

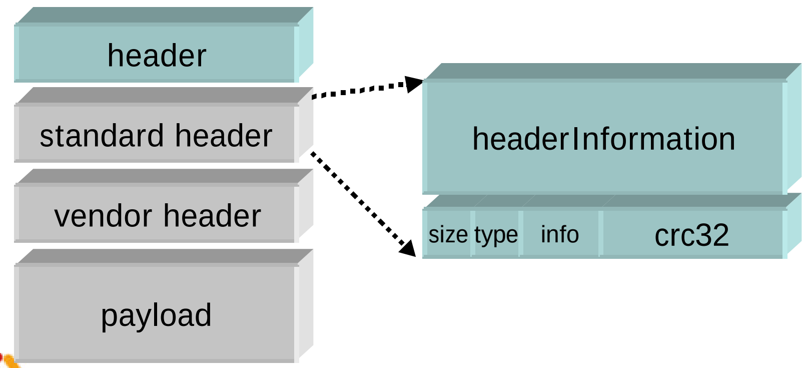
b) This provides sufficient flexibility to use the LSBs to specify the egress port of multiported attachment.

Similarly, a 64-bit *vlanIdentifier* value is easily self-administered, eliminating the need to provide any form of VLAN administration authority.

Packet headers also include size, type, quality-of-service, time-to-live and CRC32 values.



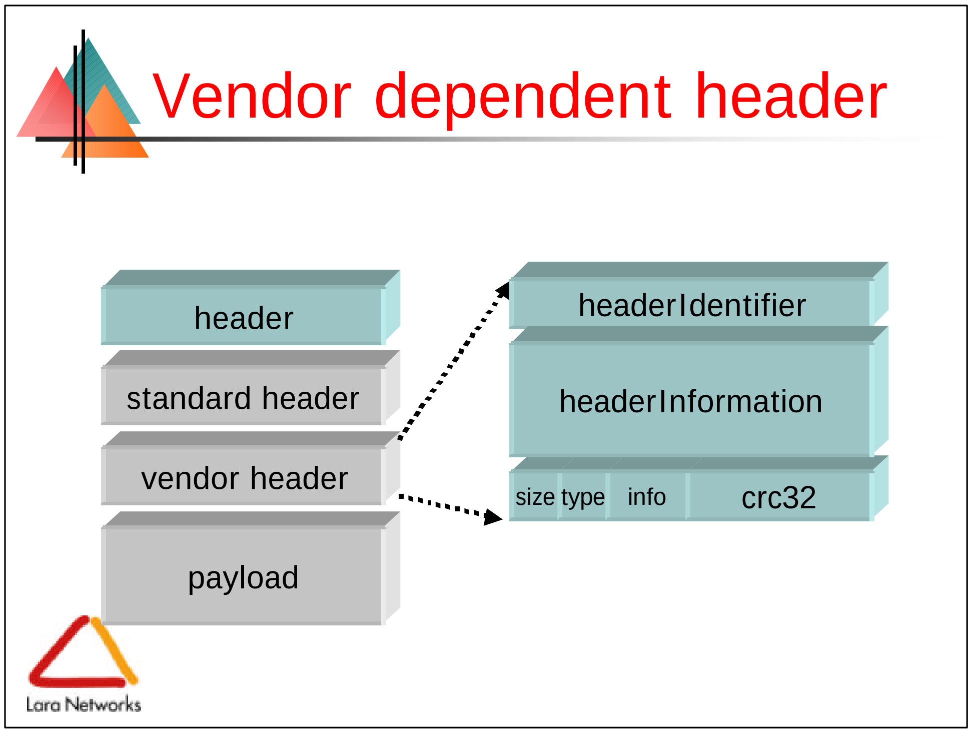
Standard extended header



Lara Networks

A standard extended header can be readily identified by the preceding 8-bit type field.

Like IP-V6, the header concludes with ***size*** and ***type*** specifiers for the next header and an independent crc32 value.



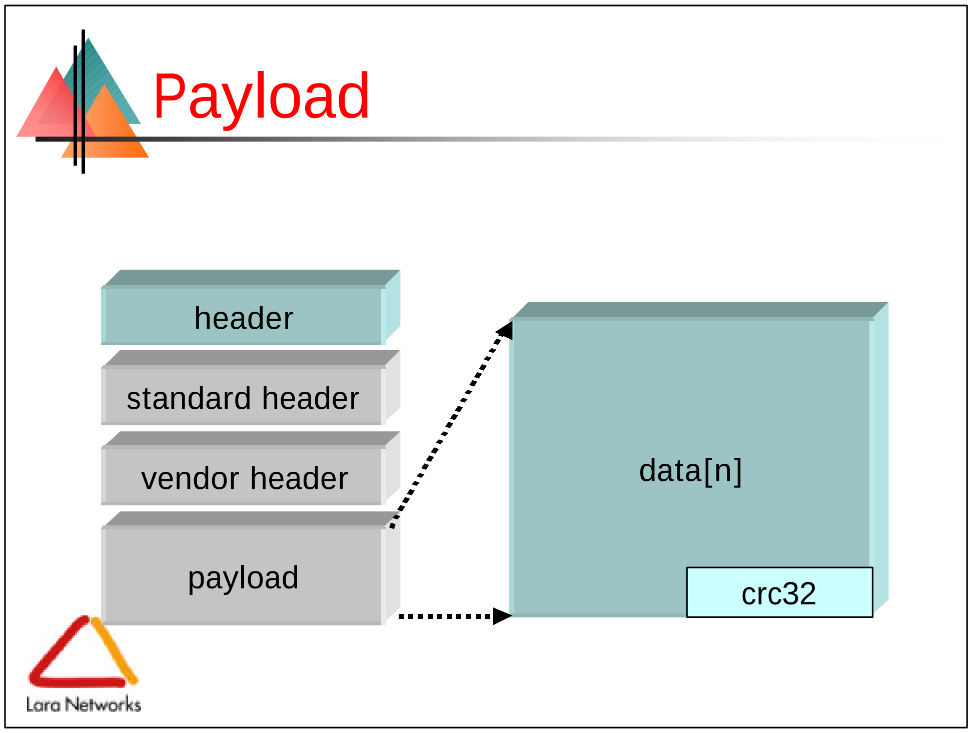
Vendor dependent header

The need to obtain distinct ***type***-value assignments is unacceptable for many companies, for several reasons:

a) The application-processing times for new number assignments can influence time-to-market.

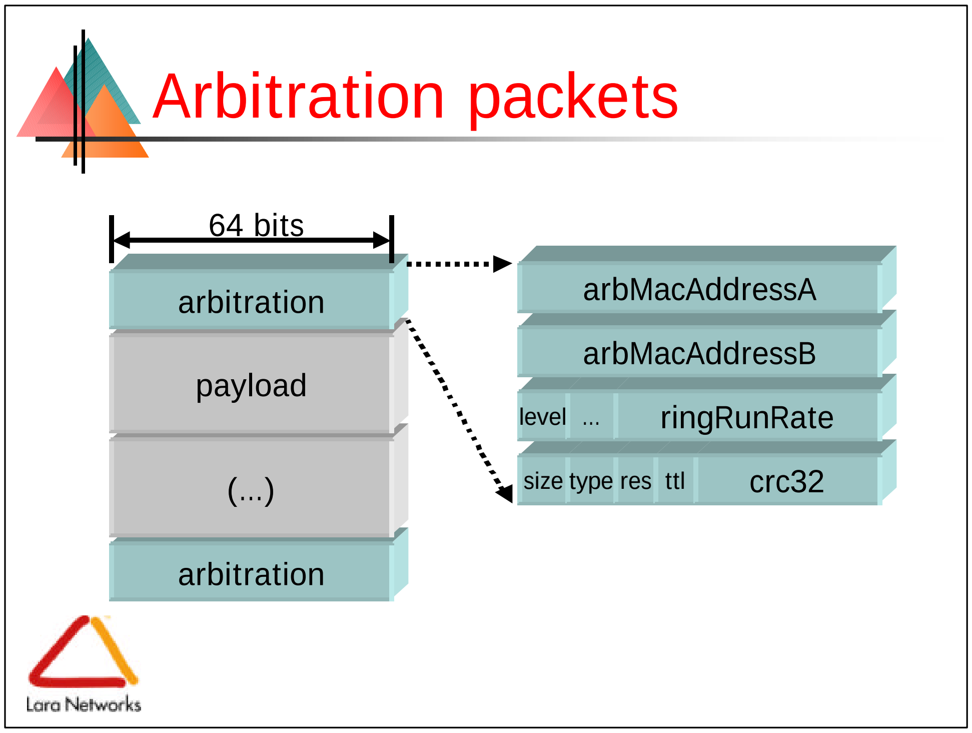
b) New companies may prefer to remain in “stealth” mode, rather than publicise their existence by requesting distinct number assignments.

So, instead, the content of a vendor-dependent header can be specified by the first 64 bits of the header, where that 64-bit value is the unique global identifier of the header-format specifying standard.



Payload

And, the payload has a distinct crc32 value.



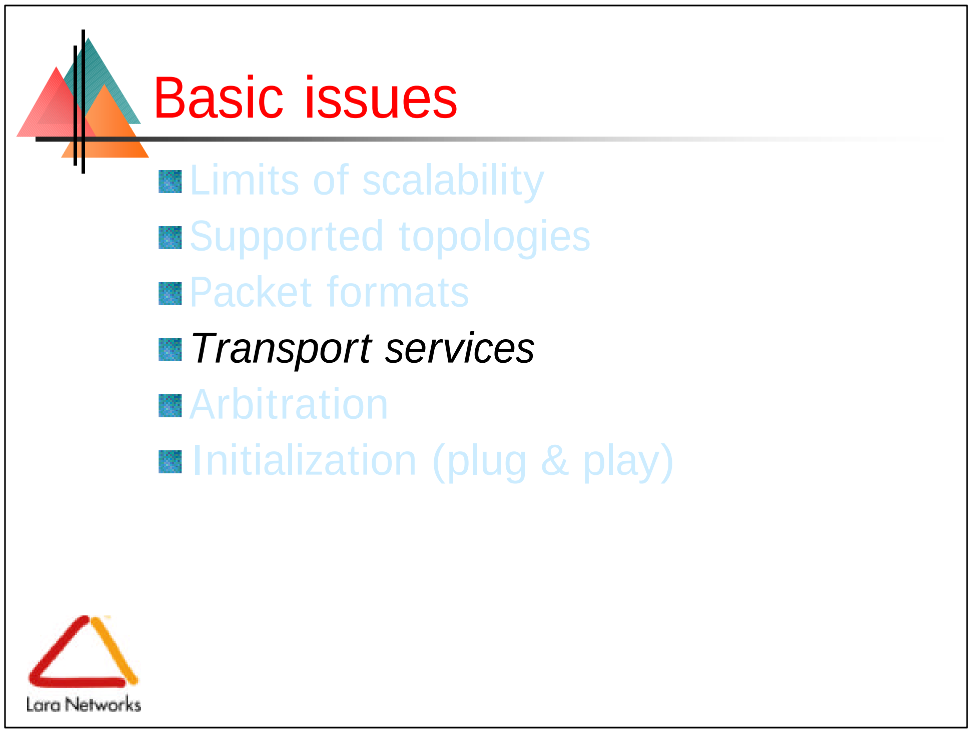
Arbitration packets

Arbitration involves sending of high-precedence control packets.

These control packets have *macAddress* identifiers, that identify primary and secondary arbitration nodes.

Also included in this packet are congestion “level” indications.

To ensure fairness, the most blocked node also asserts its *runRate*, so the *ringRunRate* ultimately selects the most blocked class-C node and ensures its progress.

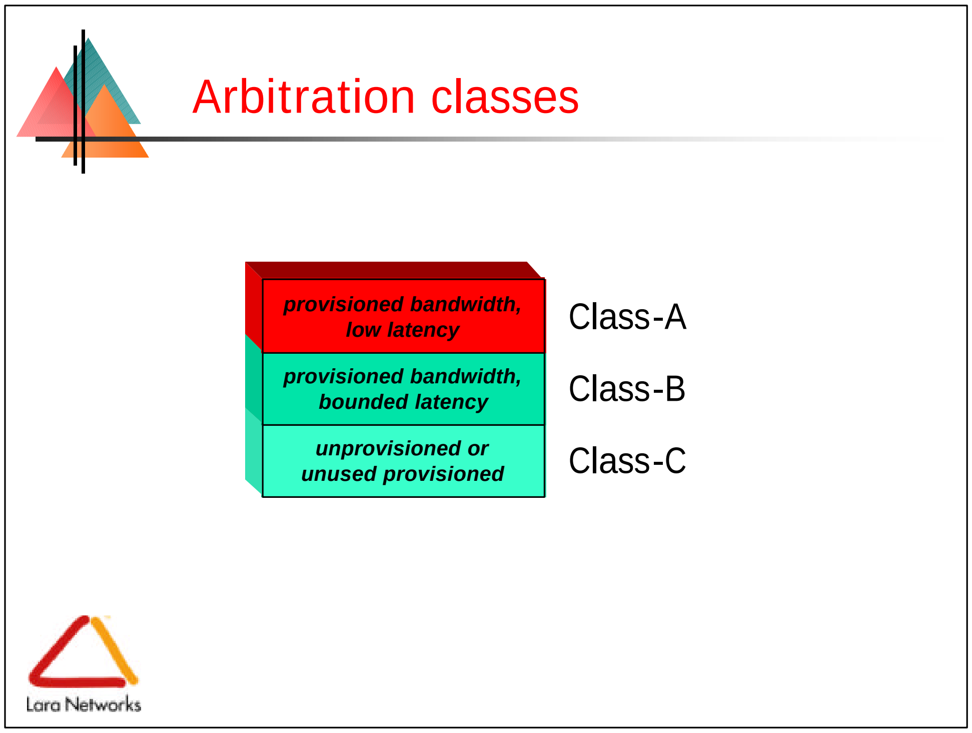


Basic issues

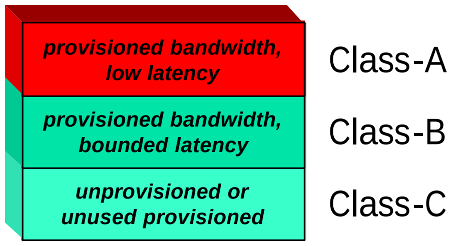
- Limits of scalability
- Supported topologies
- Packet formats
- *Transport services*
- Arbitration
- Initialization (plug & play)

Lara Networks

Classes of service are necessary, to support low latency (telephony), SLA specified bandwidth partitioning, and best-effort transmissions…



Arbitration classes


provisioned bandwidth, low latency
provisioned bandwidth, bounded latency
unprovisioned or unused provisioned
Class-A
Class-B
Class-C

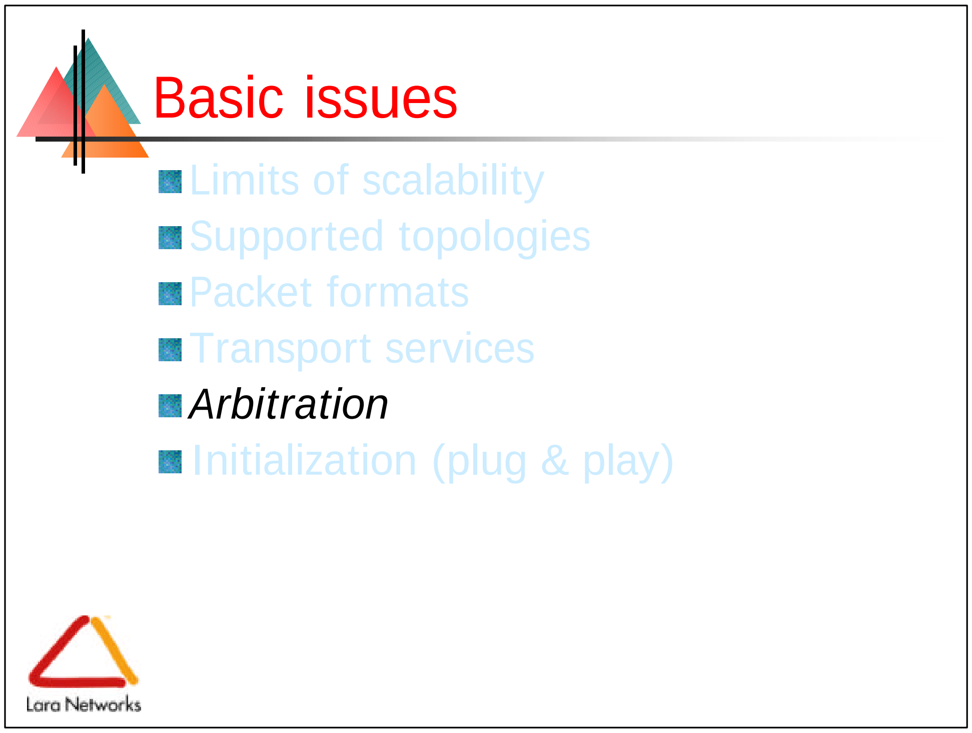
Lara Networks

Traffic classes are labeled as ‘A’, ‘B’, and ‘C’.

Class A: provisioned low-latency bandwidth, for telephone.
Supports “synchronous-like” transfers.

Class B: provisioned bounded-latency bandwidth, for SLA provisioned BW.

Class C: provisioned or unused provisioned bandwidth.
Supports ensured forward progress and fairness for the residual BW.

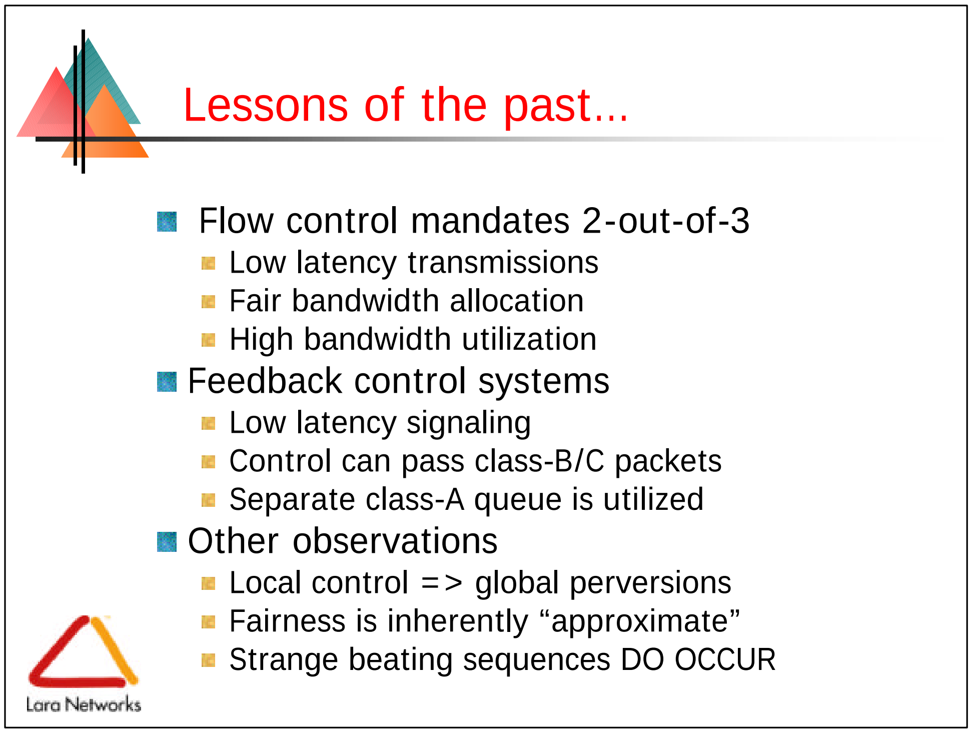


Basic issues

- Limits of scalability
- Supported topologies
- Packet formats
- Transport services
- *Arbitration*
- Initialization (plug & play)

Lara Networks

Arbitration protocols involve the stiffling of upstream nodes to ensure progress of the downstream node.



Lessons of the past…

- Flow control mandates 2-out-of-3
  - Low latency transmissions
  - Fair bandwidth allocation
  - High bandwidth utilization
- Feedback control systems
  - Low latency signaling
  - Control can pass class-B/C packets
  - Separate class-A queue is utilized
- Other observations
  - Local control => global perversions
  - Fairness is inherently “approximate”
  - Strange beating sequences DO OCCUR

Lara Networks

Note that there is inherently no optimal flow-control technique.
  Low latency transmission provides no decision-making time
  Fair allocation involves delays or BW losses
  High bandwidth utilization is easily achieved by sending when possible.
    However, this all too easily creates starvation.

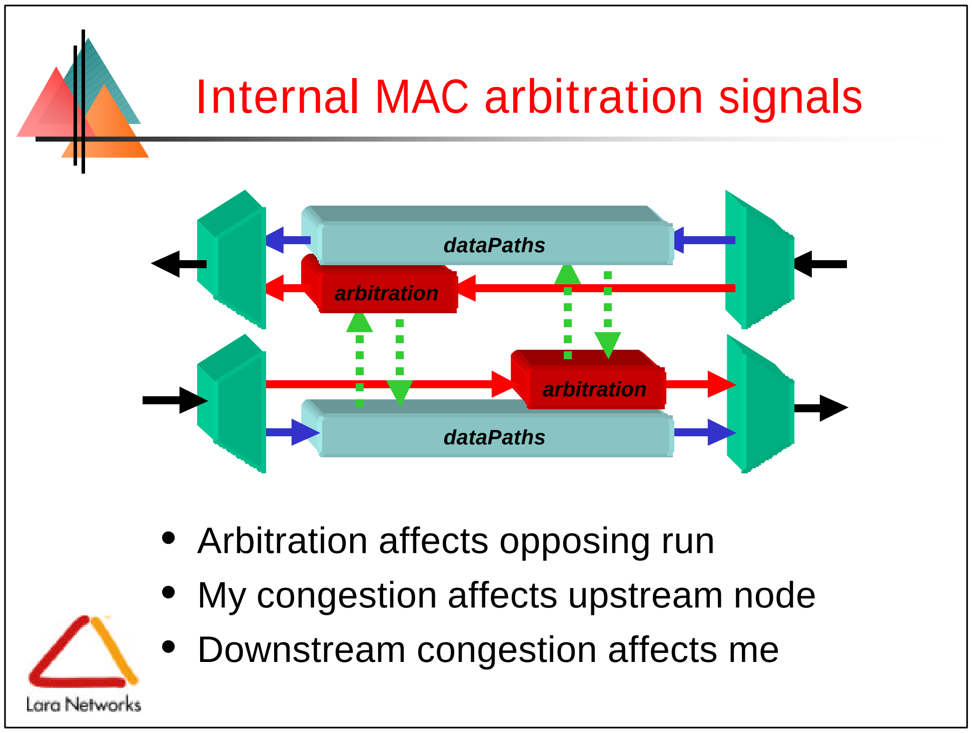
So, the basic problem is to be dynamic, in that low latency is ensured under light loading conditions and high-bandwidth (but longer latencies) are ensured under heavy loading conditions.

The dual-mode operation is a simple form of control system, where low-latency signaling is critical. So, control information is defined to be class-A traffic and this traffic-A traffic is allowed to bypass other class-B/class-C traffic.

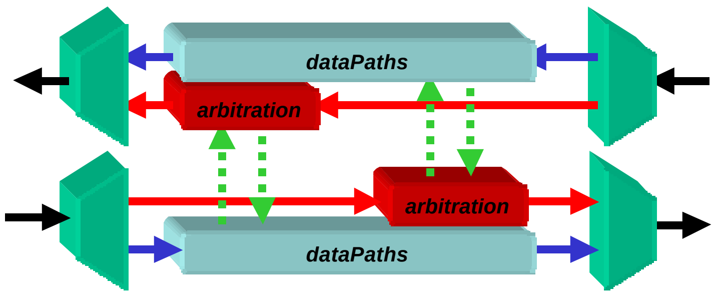
When designing this control system, control must be based on distribution of global, rather than link-level-local information. Otherwise, things like local fairness (1/2 of bandwidth for each) causes parking-lot problems and unfair $(½)^n$ servicing scenereos.

The global information, due to collection latencies, will inherently be approximate, and “fairness” will therefore also be approximate.

Algorithms should not rely on “typical” loading patterns, but should yield “provable” results. Otherwise, strange beating patterns (that often occur) can compromise latency and bandwidth guarantees.



Internal MAC arbitration signals

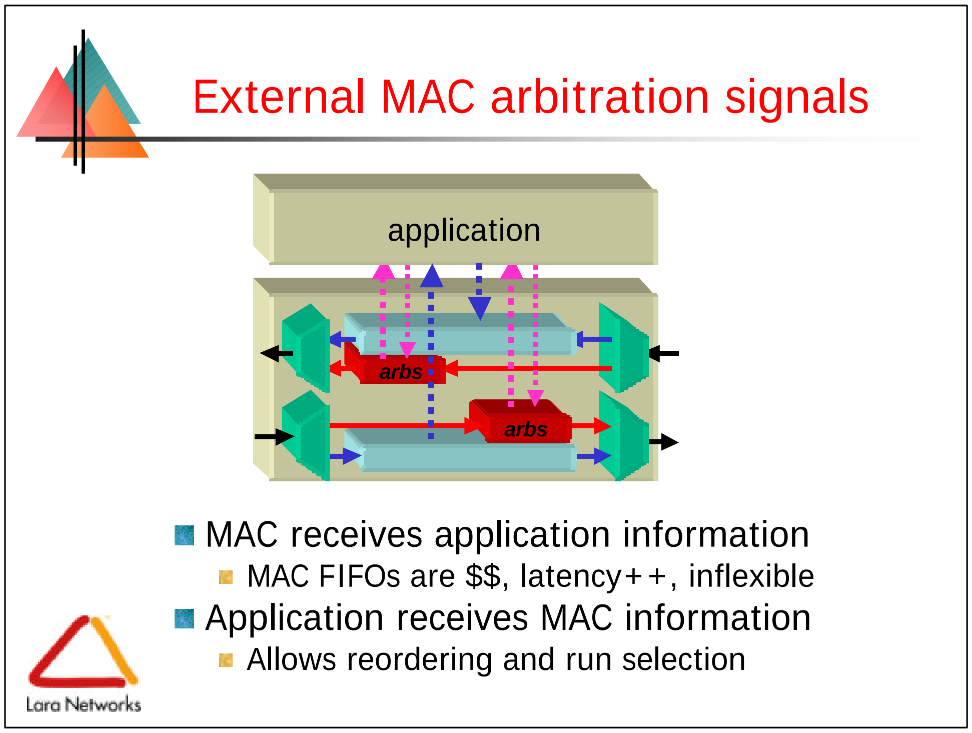


- Arbitration affects opposing run
- My congestion affects upstream node
- Downstream congestion affects me

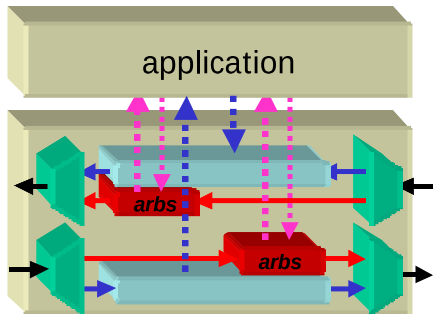
Lara Networks

Arbitration information from one link affects the transfers on the opposing link.

So, these packets are separated from the *dataPath* packets and their contents used to control the proposing of the opposing run *dataPath* processing.



External MAC arbitration signals



- MAC receives application information
  - MAC FIFOs are $$, latency++, inflexible
- Application receives MAC information
  - Allows reordering and run selection

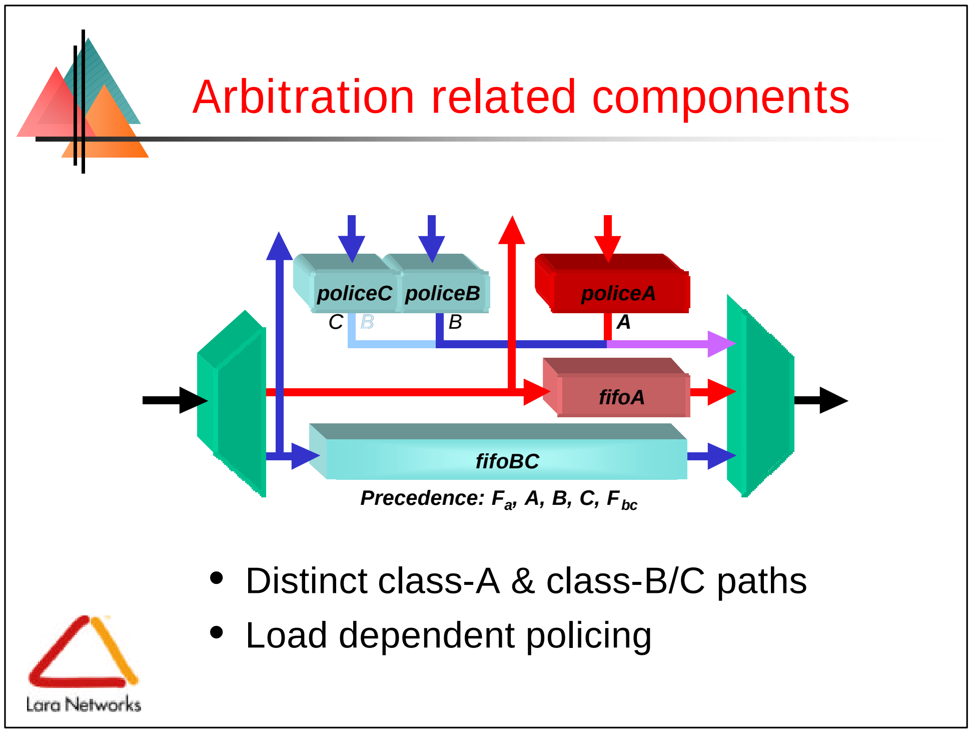
Lara Networks

For optimal performance, communications between the application and MAC is also required:

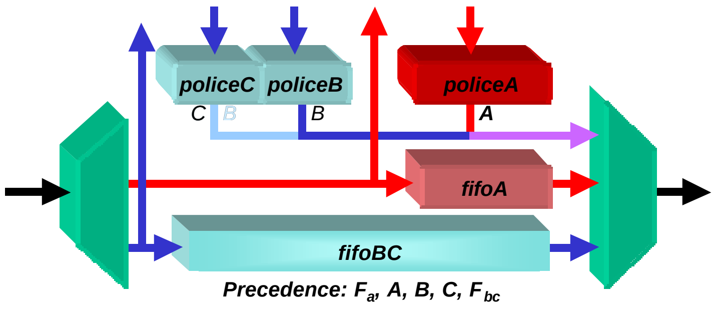
a) The application sends class-B queue depth information to the MAC, so that the application-congestion as well as MAC-FIFO depths can both influence the generation of upstream flow-control information. Communication of depth information most likely involves an out-of-band
signaling mechanism.

b) The links transports flow-control information used to limit the traffic on the opposing data run. This information is passed, within packets, to the application for (optional) fine-tuning of the transmission decisions. Communication of congestion information most likely involves sending of arbitration control packets to the higher-level application.

Note that the flow-control information for top-run data is passed to the bottom-run, and vice-vera.



Arbitration related components



- Distinct class-A & class-B/C paths
- Load dependent policing

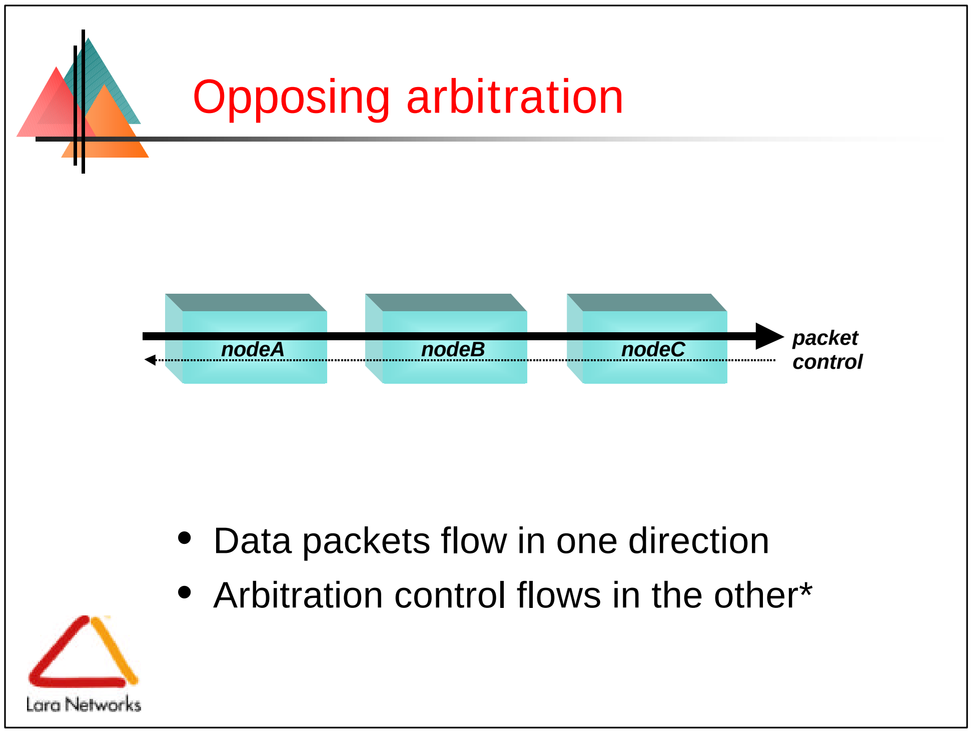
Lara Networks

A basic attach point has policing at the ingress.

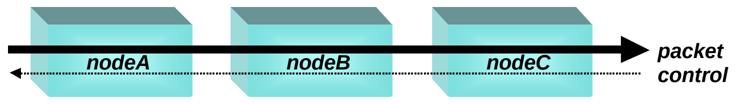
The intent is to limit the short-term class-A transfer rate, so that class-B bandwidth can be ensured. Also, a short-term bandwidth averaging requirement
is used to bound the worst-case jitter to the 125us averaging interval.

The queues are serviced in a given precedence order, but the apparent “presence” of any class effects whether this class can be sent based on policing history.

The fifoBC has to be large, so that data from the full-bandwidth traffic from an upstream node can be buffered (rather than lost) while this traffic is being throttled by a congested downstream node.



Opposing arbitration



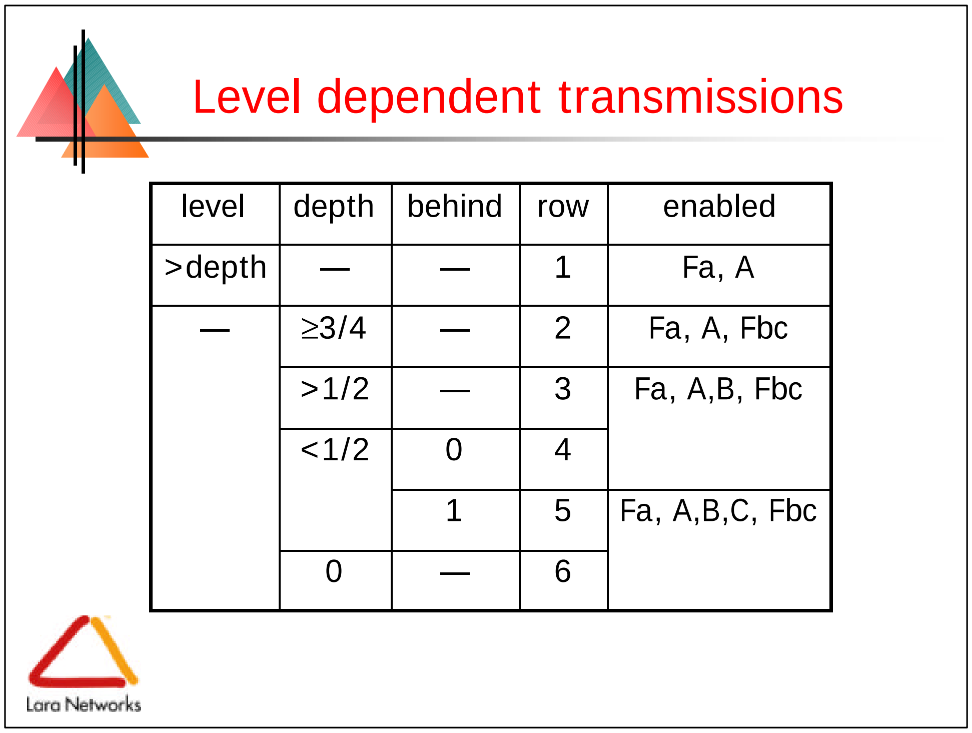
- Data packets flow in one direction
- Arbitration control flows in the other*

For a two-run connection, two ports are provided: top-run and bottom-run.

The top-run port supplies top-run data packets.
The application congestion information is actually passed to bottom-run HW.
The bottom-run congestion packets are merged with top-run data and returned through this port.

The bottom-run port supplies bottom-run data packets.
The application congestion information is actually passed to top-run HW.
The top-run congestion packets are merged with bottom-run data and returned through this port.

The opposite-run routing of data and arbitration reduces the delays between the onset and resolution of congestion conditions, by allowing control to be sent in the direction of the most-applicable excess-bandwidth offenders.



Level dependent transmissions

level	depth	behind	row	enabled
>depth	—	—	1	Fa, A
—	≥3/4	—	2	Fa, A, Fbc
	>1/2	—	3	Fa, A,B, Fbc
	<1/2	0	4	
		1	5	Fa, A,B,C, Fbc
	0	—	6	

Lara Networks

Queues are selectively enabled for transmission, based on the sensed queue-congestion level.

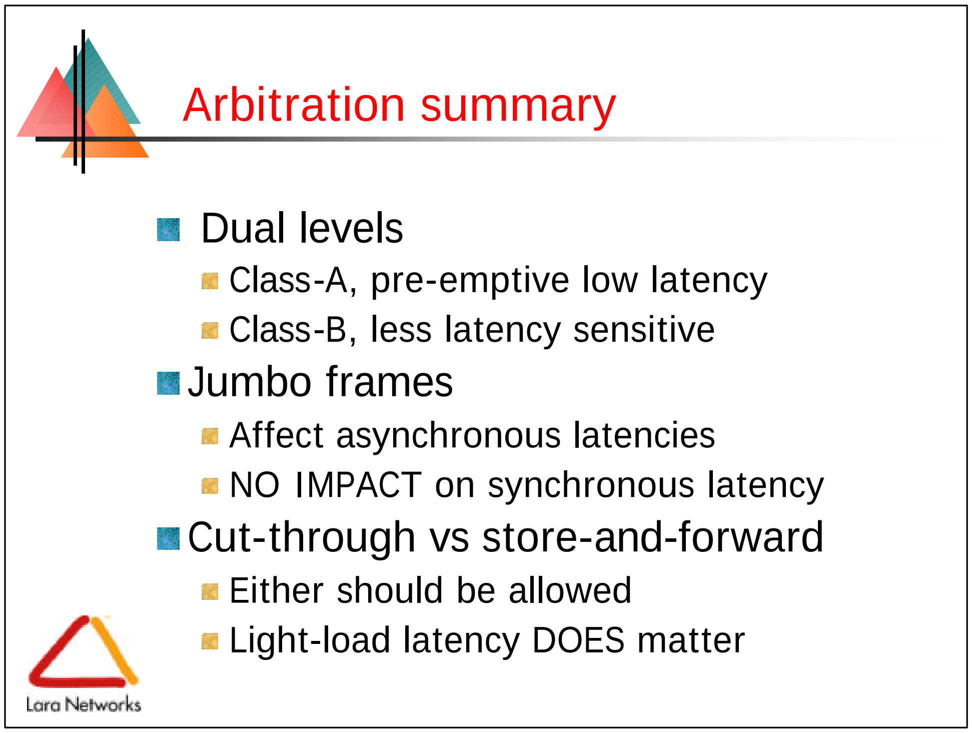
When assistance is requested, only class-A traffic can be sent. The intent is to provide idles for the congested downstream node, until a similar level of congestion is spread throughout the upstream nodes.

When assistance is not requested, a heavily congested node sends only class-A traffic while its fifoBC is being emptied. The intent is to relived this localized congestion condition by temporarily inhibiting transmissions of lower-class traffic.

When congestion levels decline, class-B (in addition to class-A) traffic can be sent.

When congestion levels decline below a threshold, nodes which are behind can also transmit their class-C traffic. The *behind* decision is made by comparing the node's *thisRunRate* parameter to a like-named field distributed in the arbitration control packet.

When congestion levels are relieved, meaning that fifoBC is entirely empty, then class-C traffic can also resume. This is opportunistic traffic that, in the absence of other controls, would not be distributed fairly. However, sending of this traffic increases the node's *thisRunRate* parameter, which places other nodes in a *behind* condition, which allows them to empty their class-C traffic. Thus, fairness is ultimately enforced.

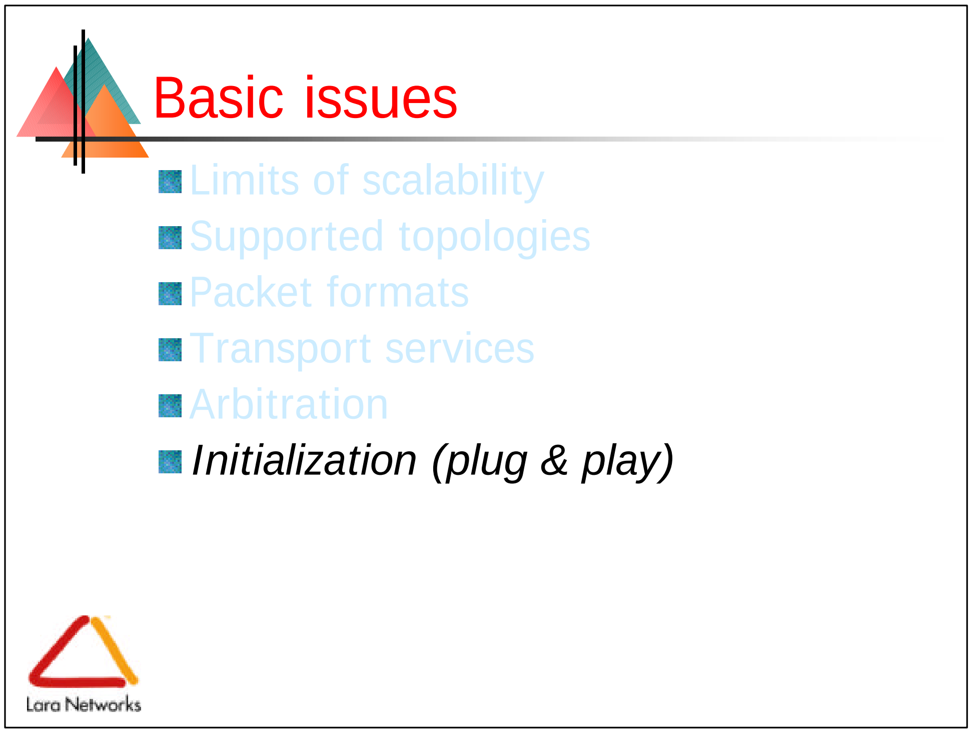


Arbitration summary

- Dual levels
  - Class-A, pre-emptive low latency
  - Class-B, less latency sensitive
- Jumbo frames
  - Affect asynchronous latencies
  - NO IMPACT on synchronous latency
- Cut-through vs store-and-forward
  - Either should be allowed
  - Light-load latency DOES matter

Lara Networks

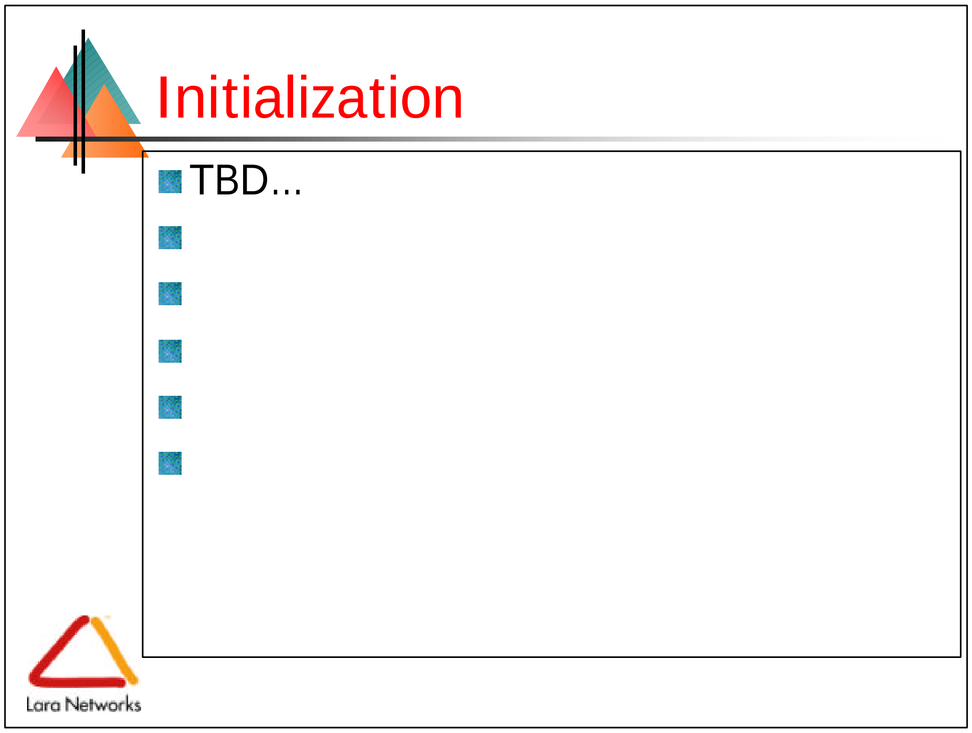
Observations…



Basic issues

- Limits of scalability
- Supported topologies
- Packet formats
- Transport services
- Arbitration
- *Initialization (plug & play)*

Lara Networks



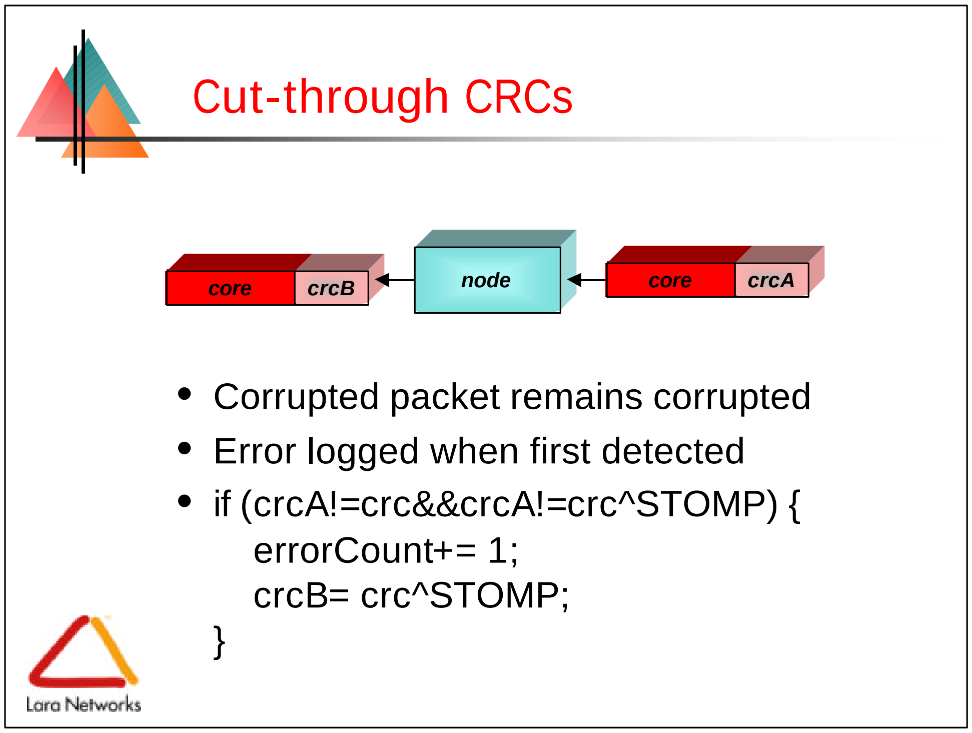
Initialization

- TBD...
-
-
-
-
-

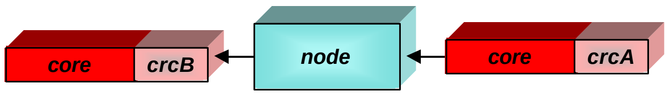
Lara Networks

Initialization protocols are responsible for constructing ringlets from arbitrary topologies, so that redundant connections can be present, but not utilized until a fault condition occurs.

Details are TBD.



Cut-through CRCs



- Corrupted packet remains corrupted
- Error logged when first detected
- if (crcA!=crc&&crcA!=crc^STOMP) {

```
    errorCount+= 1;
    crcB= crc^STOMP;
}
```

If cut-through is supported, then a node may begin retransmission on packet-A before the end of packet-A has been received. Thus, portions of packet-A may have already been sent before an invalid CRC is discovered.

To cope with such scenereos, while minimizing the impact on sequencing hardware, the remainder of the packet is always transmitted. However, the packet’s crc32 is “stomped”, but overwriting the found-to-be-invalid value with a distinctive known-to-be-invalid value, called the *stomp* value.

When the packet passes through additional nodes, the stomp value remains but is not logged as an error. In this manner, the error is normally on its first occurance, allowing the defective link to be more easily identified.

DVJ Proposal for Resilent Ring Protocol (RPR)

Draft 0.22
May 19, 2001

Comments on this proposal can be directed to the contributing editor:

David V. James, Proposal author
Lara Networks
110 Nortech Parkway
San Jose, CA 95134
Email: dvj@alum.mit.edu
Phone: +1.408.942.2010
Fax: +1.408.942.2099
Email: dvj@alum.mit.edu

Comments on the p802.17 Working Group status should be addressed to the Working Group Chair:

Mike Takefman
Cisco Systems, Inc.
365 March Road
Kanata, Ontario
Canada K2K 2C9
Phone: +1.613.271.3399
FAX: +1.613.271.3333
Email: tak@cisco.com

TBDs

The following items are TBDs:

1) Initialization. Needs functionality and coding details.
   a) Many. A n>2 ported hub is preferred over a 2-ported hub.
   b) Single. A one-ported node is not root capable, and therefore has no precedence.

Proposals for Resilient Packet Ring (RPR)

1. Overview

This is a very rough draft proposal for a Resilient Packet Ring Access Protocol (RPR). 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

NOTE—These objectives have not yet been updated to correspond to recent RPR Working Group decisions.

1.2.1 Objectives summary

The primary objective of RPR 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. RPR is easily partitioned to support distinct service level agreements (SLAs).
    b) Synchronized. Nodes can be accurately synchronized to standard global timers.
    c) Ordered. Ordered delivery is maintained within each packet flow.
    d) 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) Classified. Multiple classes of transport services are provided, including:
    a) Class-A: minimum latency prenegotiated bandwidth.
    Expected uses include telephony and video transfers.
    b) Class-B: bounded latency prenegotiated bandwidth.
    Expected uses include deterministic SLA bandwidth partitioning.
    c) Class-C: unprovisioned and unused-provisioned bandwidth.
    This residual traffic is partitioned in an approximately fair fashion.
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: RPR is applicable between within-the-home and within-the-planet distances.
    b) Bandwidth: RPR is applicable to low-rate as well as multiple gigabit/second rate media.

1.2.2 Objectives, requirements, and strategies

1.2.2.1 Compatible

Objective:	Ethernet, ATM, and other protocol packets should be sent in an unmodified fashion.
Requirement:	Shall be capable of encapsulating Ethernet packets.
Strategy:	The 48-bit MAC addresses are encapsulated within 64-bit MAC addresses.
	Standard and vendor-dependent headers enable identification of encapsulated formats.

Objective:	Should be physical layer independent, allowing use any byte transfer media.
Requirement:	Shall be capable of using any byte and control transfer media.
Strategy:	Control information is sent within small class-A packets.

1.2.2.2 Quality of service

Objective:	Should support eight or more quality-of-service levels.
Requirement:	Shall support three class-of-service categories:
	1) Class-A: minimum latency guaranteed bandwidth.
	2) Class-B: bounded latency guaranteed bandwidth.
	3) Class-C: unprovisioned and unused provisioned bandwidth.
Strategy:	Separate bypass FIFOs for class-A and class-B/class-C traffic.
	Allow class-A mid-packet preemption of class-B or class-C traffic.

Objective:	Should support SLA (service level agreements) of bandwidth and quality of service.
Requirement:	Shall support SLA (service level agreements) of guaranteed bandwidth and latency.
Strategies:	Bandwidth guarantees for provisioned low-latency and bounded-latency traffic.
	Provide 1 ms latency guarantees for low-latency traffic.

Objective:	Non-negotiated class-C bandwidth should be fairly allocated.
Requirement:	Non-negotiable class-C traffic shall have bounded delivery delays.
Strategies:	The node with the lowest consumption count sends prioritized C_b traffic.

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:	Packets are transmitted and received in the same order, if they are sent over the same ringlet and have the same A/B/C class identifiers.

1.2.2.3 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:	If not otherwise supported by the physical layer, 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.

1.2.2.4 Scalable

Objective:	The link should be applicable to within-the-home through within-the-planet applications.
Requirement:	The link shall 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 class-BC traffic, so the length of class-BC packets doesn't effect class-A packet latencies.

1.2.2.5 Resilient

Objective: The link should recover from topology changes within 50 ms.
Requirement: The link shall recover from topology changes within several 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.
Miss-addressed packets are stripped within several ringlet-circulation times.

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.

1.2.2.6 Efficient

Objective: The unicast destination or final multicast destination strips the returning packet.
Requirement: The unicast destination or the multicast destination strips the returning packet.
Strategies: The unicast packets are stripped based when they reach their destination.
The multicast packets are stripped based when they return to their source.

Objective: Packets should be transmitted in the most-optimal direction.
Requirement: Packets shall be allowed to be sent on the more optimal counter-rotating ringlet.
Strategies: Allow a time-stamped acknowledge indications to be returned.
The transmitter has the option of changing ringlets, based on acknowledge information.

1.3 MAC interface functionality

The MAC interface definition provides an ingress data path and application-depth indication, as illustrated in figure 1. In a similar fashion, the MAC provides an egress data path and interleaves link-congestion arbitration-control packets on that link.

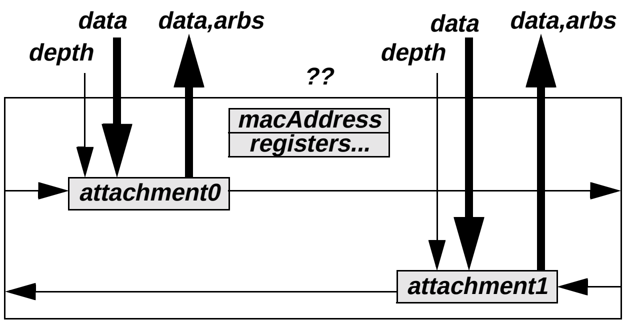


Figure 1—MAC interface signals

1.4 RPR topologies

1.4.1 Ring topologies

RPR is targeted for cable-ring topologies and maximizes bandwidth capabilities through the use of full-duplex cabling, as illustrated in figure 2. The full-duplex cable infrastructure normally allows concurrent transmissions on the clockwise and counter-clockwise rings, as illustrated in the left of figure 2. After a single link failure, communication continues (but at a reduced rate) over the remaining ring, as illustrated in the center figure 2. After a duplex cable failure, communication continues (but at a further reduced rate) over the daisy-chained connection, as illustrated in the right of figure 2.

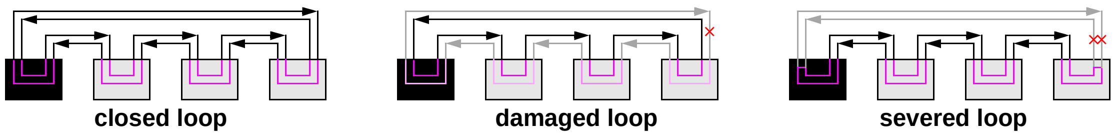


Figure 2—Ring topologies

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 class-A 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].

1.4.2 Hub topologies

An RPR topology can include larger multiported nodes, called hubs, as illustrated in figure 2. An N-ported hub can be used to expand the number of lower-cost single-ported nodes, as illustrated in the left of figure 2. Alternatively, hubs can support redundant fault tolerant connections, as illustrated in the right of figure 2.

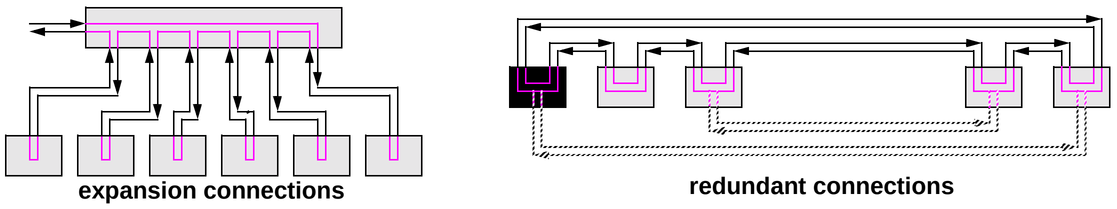


Figure 3—Ring topologies

[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

1.5 Node data paths

1.5.1 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 figure 4

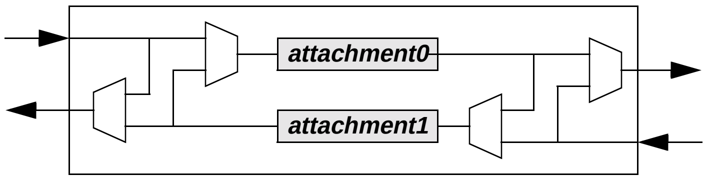


Figure 4—Selectable data paths

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

1.5.2 Traffic classes

RPR supports three types of delivery services, as illustrated in figure 5 and listed below. Each node is required to police its class-A, class-B, and class-C traffic to avoid exceeding its provisioned limits.

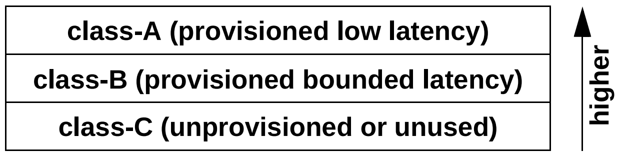

class-A (provisioned low latency)
class-B (provisioned bounded latency)
class-C (unprovisioned or unused)
higher

Figure 5—Attach point routing

1) Class-A: Provisioned guaranteed-latency bandwidth.
   This service is expected to be used for transmission of streaming audio and/or video traffic.
2) Class-B: Provisioned bounded-latency bandwidth.
   This service is expected to be used for transmission of contracted nonstreaming traffic.
3) Class-C: Fairly assigned unprovisioned or unused provisioned bandwidth.
   This service is expected to be used for transmission of nonprovisioned traffic.

1.5.3 Ordering constraints

To implement multiple classes of service, the RPR transport allows packet reordering. Sufficient ordering constraints, listed below, limit the reordcering effects on higher level applications.

1) Stream order. Within the context of one ringlet, packets with the same source-MAC-address, destination-MAC-address, and class (A, B, or C) shall maintain their releative ordering.
2) Flow order. Within the context of any ringlet span, class ordering constraints apply:
   a) Lower class limits. Lower-class traffic shall not advance past higher-class traffic:
      i) Class-B traffic cannot pass classA traffic.
      ii) Class-C traffic cannot pass class-B traffic.
   b) Higher class allowance. Higher-class traffic may advance past lower-class traffic:
      i) Class-B traffic may pass classC traffic.
      ii) Class-A traffic may pass class-B or class-C traffic.

1.5.4 Class-A packet queues

A node's attach point has multiple queues in each attach component, as described in 5.2. Each attach point has a receive interface, a transmit interface, and two bypass buffers for resolving concurrent reception/transmission conflicts, as illustrated in frame-1 of figure 6. Use of these queues depends on the passing through packet type; class-A and class-B/class-C packets are routed through *fifoA* and *fifoBC* components respectively.

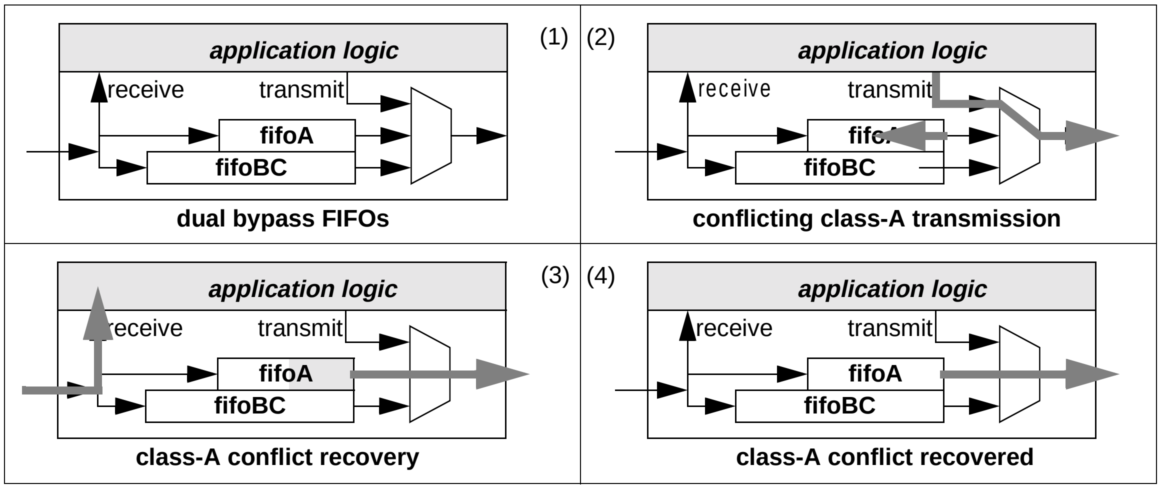


Figure 6—Class-A packet queues

From the perspective of class-A packets, the packet-transmission protocols are described below:

1) Setup. An outbound packet is readied for transmission.
2) Transmit. The class-A packet is transmitted, as illustrated in frame-2 of figure 6.
   During the transmission interval, incoming class-A traffic is saved in the *fifoA* buffer.
3) Recover. The *fifoA* is allowed to empty, as illustrated in frame-3 of figure 6, in several ways:
   a) Incoming packets are stripped, while *fifoA* symbols are sent.
   b) Incoming idle symbols are discarded, while *fifoA* symbols are sent.

1.5.5 Preemptive bypass queues

The purpose of these two queues is to reduce the transit time of class-A packets, by allowing class-A packets to bypass conflicting in-transit class-B/class-C packets, as described in the remainder of this subclause.

A class-B packet may arrive during a class-A packet transmission, as illustrated in frame-2 of figure 7. The presence of an active class-A packet transmission causes the class-B packet to be saved in the *fifoBC*, with the effect of increasing the passing-through class-B/class-C packet latencies.

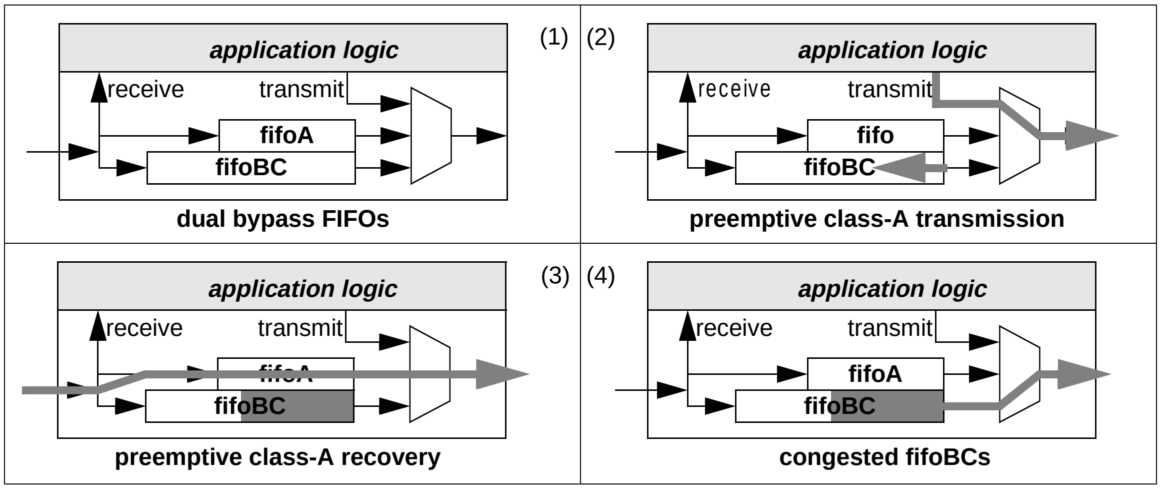


Figure 7—Class-B/class-C queue preemption

To reduce the class-A delivery latencies associated with partially filled *fifoBC* components, class-A packets are routed through a distinct (and much smaller) *fifoA* component, as illustrated in frame-3 of figure 7.

Between packet gaps eventually allows the *fifoBC* queue to be emptied, as illustrated in frame-4 of figure 7. Before this occurs, additional class-A packets may be transmitted, further increasing the *fifoBC* depth. Alternatively, additional class-A packets may be transmitted or routed through the lower-latency *fifoA* component.

The use of a separate preemptive *fifoA* queue increases the MAC design complexity, but if viewed as unavoidable. The (less desirable) alternatives would be one of the following:

1) Block class-A. The class-A packet transmissions could be delayed, while signaling the upstream node to cease transmission of unexpected class-BC traffic. However, this would increase class-A packet-transmission latencies.
2) Toss class-BC. The class-BC packets could be discarded, while signaling the upstream node to cease transmission of insufficiently throttled class-BC traffic. However, this would increase class-BC packet transmission losses.
3) Less class-BC. Statistical throttling of the upstream node transmissions could be utilized to provide bandwidth for expected downstream class-A and class-B packet transmissions. However, potential class-C bandwidth is wasted when the actual level of class-A/class-B traffic is less than expected.

1.6 Queuing options

1.6.1 Cut through and store-and-forward

The terms cut-through and store-and-foward describe options for the processing of pass-through traffic. Cut-through processing allows the initial portion of a packet (a) to be forwarded before the final portion of the packet (b) has been received, as illustrated in the left of figure 8. Store-and-forward processing delays packet forwarding until after the final portion of the packet has been received, as illustrated in the right of figure 8.

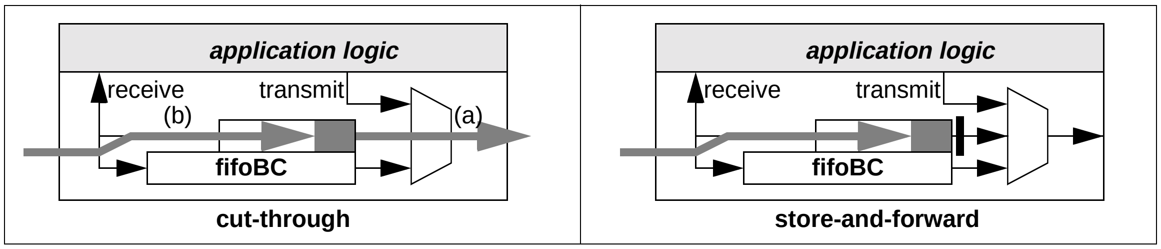


Figure 8—Cut-through and store-and-forward processing options

The choice of using cut-through or store-and-forward processing is implementation dependent and beyond the scope of this standard.

NOTE—Although cut-through implementations can reduce transport latencies, this feature may imply additional design complexities, such as special circuitry to support the “stomping” of CRCs (see 6.1) on passing through packets.

1.6.2 Preemption

The term preemption describes a technique for suspending the transmission (1) of pass-through lower-class traffic for sending (2a) one or more class-A packets, as illustrated in the top left and right of figure 9. The continuation of the lower-class (class-B or class-C) packet is queued (2b) while the class-A packet is being sent.

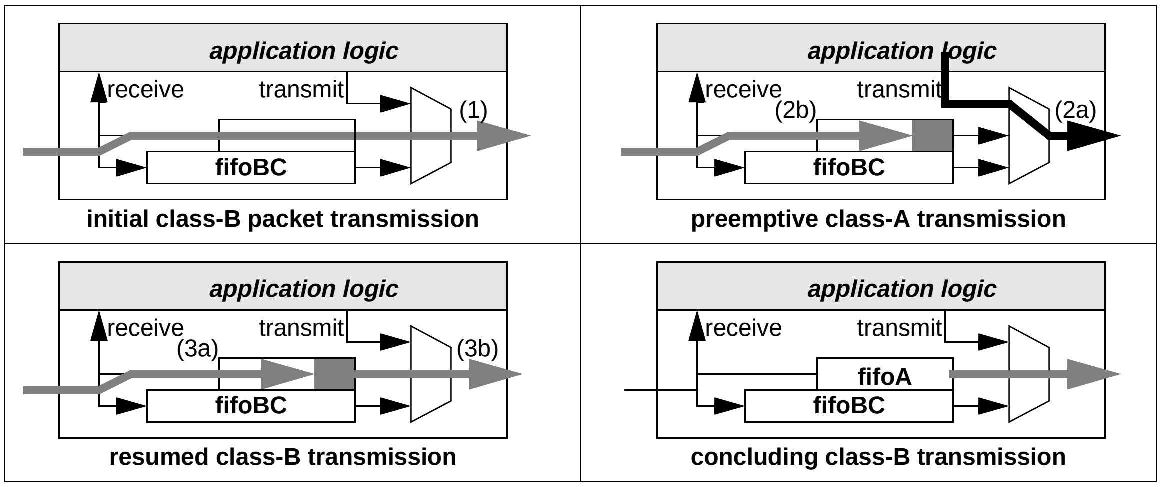


Figure 9—Preemptive packet processing

After the preemption, the lower-class packet reception (3a) and retransmission (3b) continues. The remainder of the lower-class packet is then transmitted, as illustrated in the bottom right of figure 9.

Preemptive packet processing has the effect of splitting lower-class packets, as illustrated in the left of figure 10. If packetA is stripped before packetX, a between-packet void can also be generated, as illustrated in the right of figure 10.

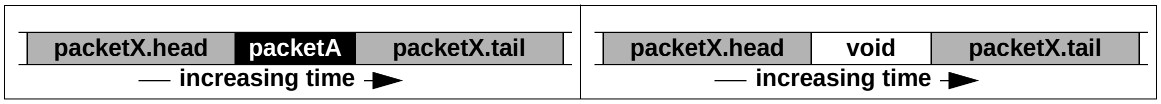


Figure 10—Preemptive packet sequences

A node may discard these void symbols while transmitting idles or packetized symbols, with the intent of reducing the depth of packets (or packet fragments) within fifoA or fifoBC components. The void symbols are distinct from other between-packet idles, however, in that their transmissions shall not be interrupted by the transmission of other lower-class components.

If not discarded previously, the void symbols are converted into the normal between-packet idles when the preceding packet component (*packetX.head*, in figure 10) is stripped at its destination node.

NOTE—Preemptive packet processing relies on distinct code symbols to identify the border between packetized lower-class data and the start of preemptive class-A packets. Because these codes may not be supported by all physical layers, this feature is physical-layer dependent and not mandated by this standard. Supportive physical layers may mandate preemption, may prohibit preemption, or may define an interoperable preemption option.

Preemption mandates additional hardware, including store-and-forward sequencing of receive queues, to isolate the application from the packet fragments generated on the interconnect. To ensure interoperability, preemptive transmitters connected to downstream non-preemptive receivers shall use store-and-forward processing techniques to merge received packet fragments before their retransmission, as illustrated in figure 11.

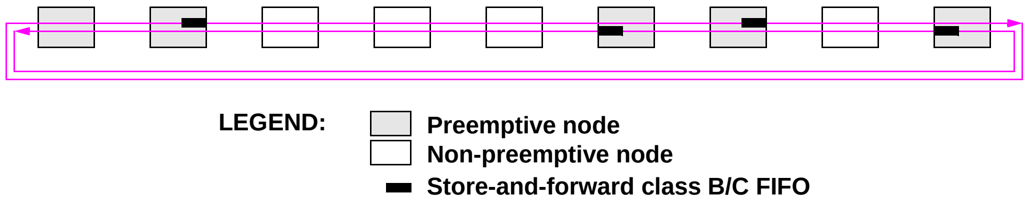


Figure 11—Store-and-forward components for preemption interoperability

1.7 Arbitration protocols

1.7.1 Arbitration principles

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 opportunistic 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.

1.7.2 Arbitration signal flow

For efficiency, arbitration signals and data packets normally flow in the opposite direction, as illustrated in the top of figure 12. Top-run data-packet transmissions (sold lines) are associated with bottom run arbitration-control-packet transmission (dotted lines), as illustrated in the top left of figure 12. Similarly, bottom-run data-packet transmissions (sold lines) are associated with top-run arbitration-control-packet transmission (dotted lines), as illustrated in the top right of figure 12.

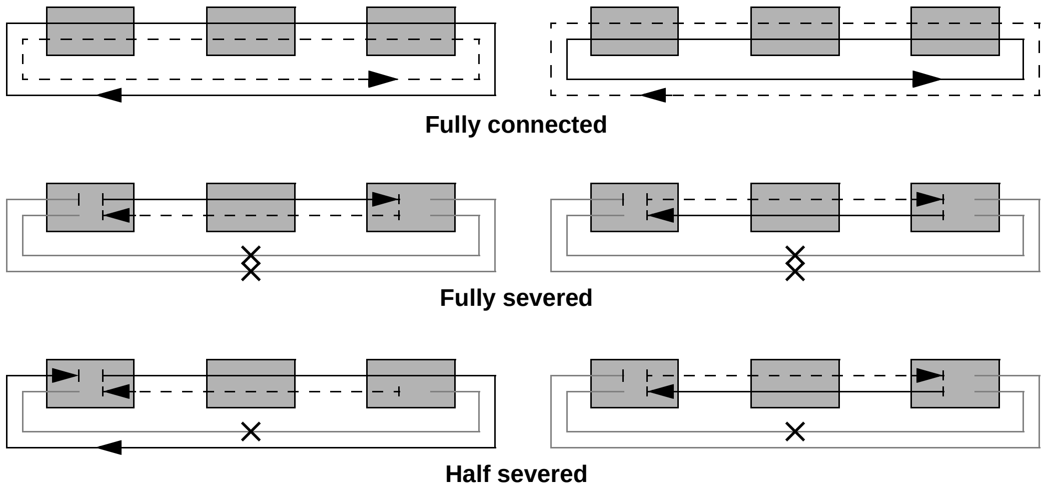


Figure 12—Opposing packet and arbitration paths

When a link is fully severed, packets are steered to the top or bottom runs, based on their right-side or left-side destination, as illustrated in the center of figure 12. Top-run data-packet transmissions (sold lines) are associated with bottom run arbitration-control-packet transmission (dotted lines), as illustrated in the left center of figure 12. Similarly, bottom-run data-packet transmissions (sold lines) are associated with top-run arbitration-control-packet transmission (dotted lines), as illustrated in the right center of figure 12.

When a link is half severed, packets are once gain steered to the top or bottom runs, based on their right-side or left-side destination, as illustrated in the bottom of figure 12. However, one of the boundary nodes (in this example, the leftmost node) can be reached from either side, improving the available link bandwidth.

1.7.3 Arbitration connectivity

The ***data*** and ***arbs*** (arbitration control) packets are interleaved and transferred over the same link. The arbitration control packets on one run are affiliated with the data flows on the opposing run, as illustrated in the left of figure 13. Within this figure, the dotted lines indicate arbitration-control-information flows.

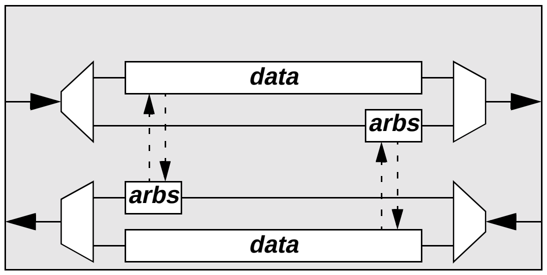


Figure 13—Opposing-run arbitration effects

The dotted lines from *dataFlows* into *arbs* facilitate the sending of upstream arbitration warnings when queues become congested. The dotted lines from *arbs* into *dataFlows* allows facilitates the throttling of traffic within the upstream node when such congestion warnings are received.

The application may elect to receive port-related arbitration, for the purposes of fine-tuning its transmit-packet selections. When so enabled, the arbitration from the opposing run are interleaved with the data packets returned through the port interface, as illustrated in figure 14.

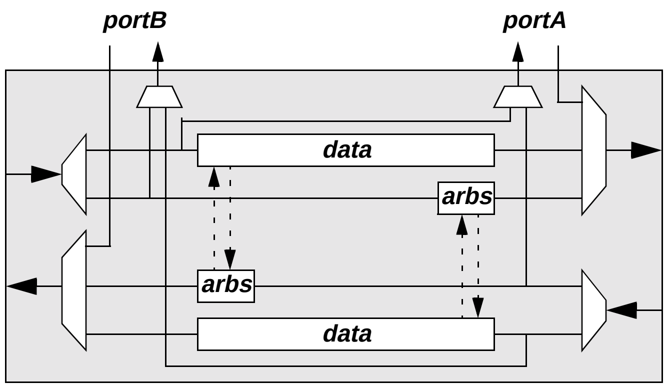


Figure 14—Arbitration packet deliveries

1.8 Future possibilities

1.8.1 Excess-conductor communications

Future links could theoretically be composed of more than two conductors, as illustrated in figure 15. However, this could complicate flow control specifications (normally based on opposing run signaling), because not all runs would have an affiliated opposing runs.

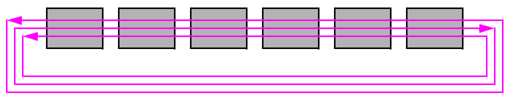
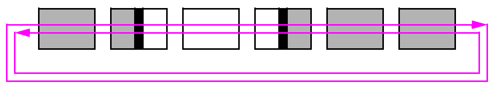


Figure 15—Excess-conductor communications

1.8.2 Heterogeneous link bandwidths

Future extensions to this standard could support rings constructed from different-speed links, as illustrated in figure 16. This could affect provisioned bandwidth allocation protocols, because the traffic's fractional-bandwidth allocations differ on the higher-speed and lower-speed links. Also, larger latencies may be incurred when passing through speed-matching FIFOs.



LEGEND:

░	**Higher speed transfers**
□	**Lower speed transfers**
▮	**Speed matching buffers**

Figure 16—Heterogeneous link bandwidths

1.8.3 Mesh topologies

Mesh topologies can support higher bandwidths, due to the additional link and a reduced number of links in most source-to-destination path, as illustrated in figure 17. However, additional routing and buffering specifications could be required to support transfers of packets from one ring to another.i

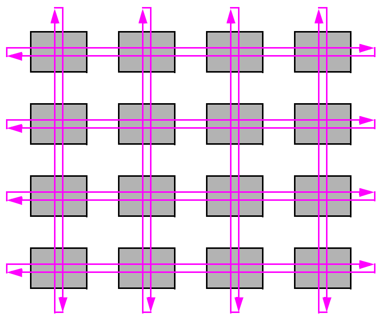


Figure 17—Mesh topologies

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^{2}I 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^{2}I 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 RPR 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 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.

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

3.1.2.4 class-A packet: A component of class-A traffic.

3.1.2.5 class-A traffic: Data traffic for which the transmission bandwidth is provisioned and low latency is ensured by assignment of the maximum effective transmission priority.

3.1.2.6 class-B packet: A component of class-B traffic.

3.1.2.7 class-B traffic: The collection of data packets with provisioned-bandwidth and bounded-latency characteristics, whose transmission can only be preempted by higher priority class-A traffic.

3.1.2.8 class-BC packet: A component of class-B or class-C traffic.

3.1.2.9 class-BC traffic: The combined class-B and class-C traffic.

3.1.2.10 class-C packet: A component of class-C traffic.

3.1.2.11 class-C traffic: Data traffic for which the transmission bandwidth is unprovisioned or represents unused portions of other provisioned bandwidths. Fairness protocols attempt to allocate this traffic in an equitable fashion.

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

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

3.1.2.14 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.

3.1.2.15 node: The entity associated with a particular unique identifier and identifier-affiliated resources.

3.1.2.16 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.

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

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

3.1.2.19 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.

3.1.2.20 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.

3.2 Acronyms

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

RPR	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 table 1:

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
Maximum(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

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

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.

4. Packet formats

4.1 Packet formats

4.1.1 Header formats

A packet consists of a basic header, zero or more optional extended header, and payload components. The basic header identifies the type and size of following header components, as illustrated in figure 18.

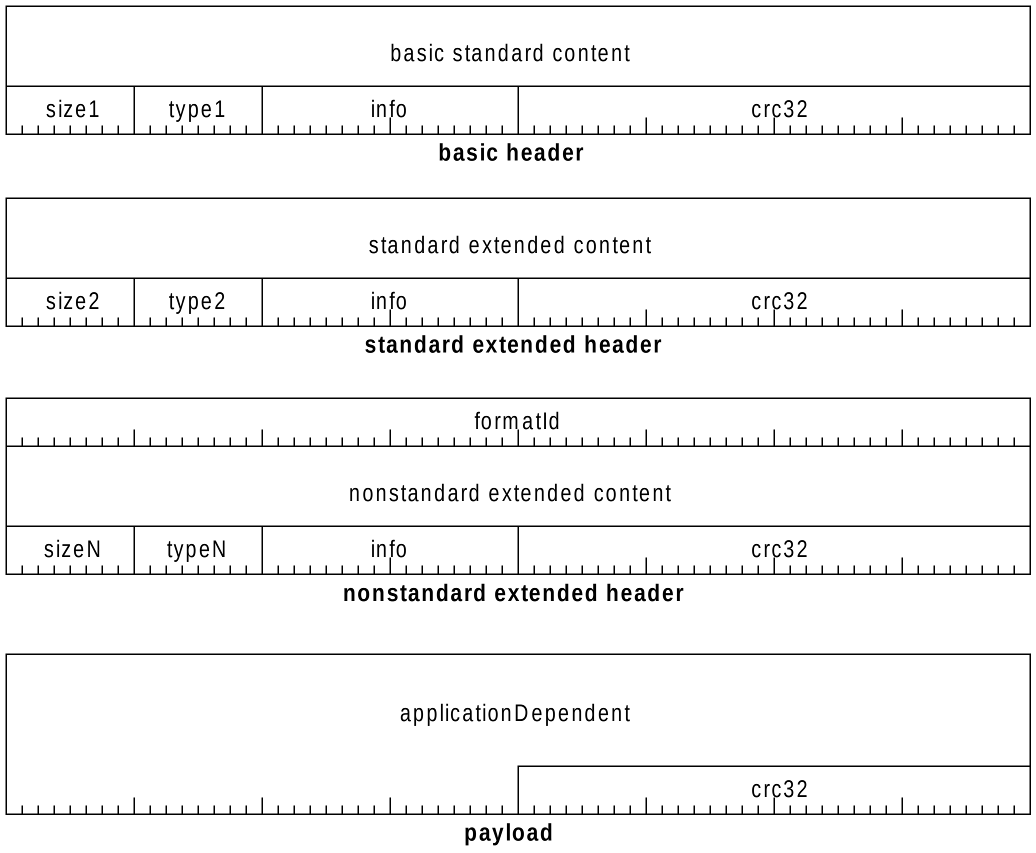


Figure 18—Packet formats

Within the initial header, the ***size1*** and ***type1*** fields specify the size and type of the following component. A zero valued ***size1*** field indicates the presence of a following payload component.

Within the following extended header, the ***size2*** and ***type2*** fields specify the size and type of the following component. A zero valued ***size2*** field indicates the presence of a following payload component.

Within the final extended header, the ***sizeN*** and ***typeN*** fields specify the size and type of the following component. A zero valued ***sizeN*** field indicates the presence of a following payload component.

All header and payload components are terminated with a final 32-bit ***crc32*** value, which independently protects the contents of each component.

4.1.2 Standard initial header format

The standard initial header starts with 64-bit *destinationMacAddress* and *sourceMacAddress* components, as illustrated in figure 19.

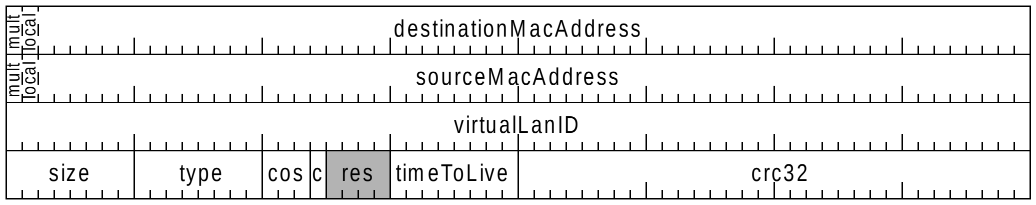


Figure 19—Standard initial header format

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 8-bit ***size***, 8-bit ***type*** field, and the 32-bit ***crc32*** field are specified in 4.1.1.

The 3-bit ***cos*** (class of service) field communicates high resolution class of service from one application to another. The intent is to facilitate inexpensive queue-placement and/or reordering at the MAC-to-application interface, based on sophisticated flow classification information provided by the application layer of the transmitter. Although transported through the MAC layer, this field shall be otherwise ignored by the MAC-layer transport.

A 1-valued ***c*** bit identifies class-A traffic; the 0 valued ***c*** bit identifies other (class-B or class-C) traffic. The 4-bit ***res*** field shall be reserved.

The 8-bit ***timeToLive*** field is decremented when packets pass through nodes, marking stale packets to facilitate their timely demise.

4.1.3 Standard extended header

The formats of the standard-initial and standard-extended headers are similar, as illustrated in figure 20.

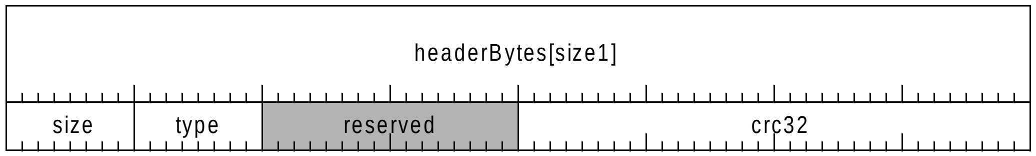


Figure 20—Standard extended header

The 8-bit ***size*** field, the 8-bit ***type*** field, and the 32-bit ***crc32*** field are specified in 4.1.1.

4.1.4 Nonstandard extended header

The formats of the standard-initial and standard-extended headers are similar, as illustrated in figure 21.

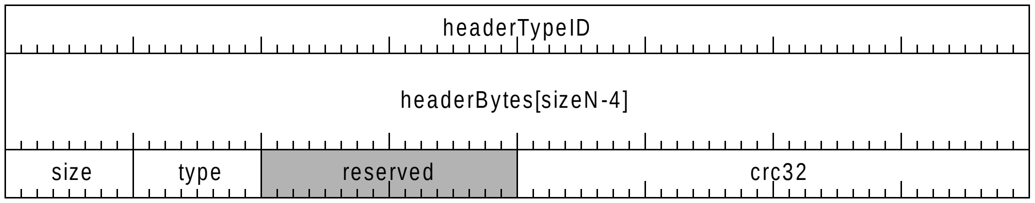


Figure 21—Nonstandard extended header

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

The 8-bit ***size*** field, the 8-bit ***type*** field, and the 32-bit ***crc32*** field are specified in 4.1.1.

4.1.5 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 22.

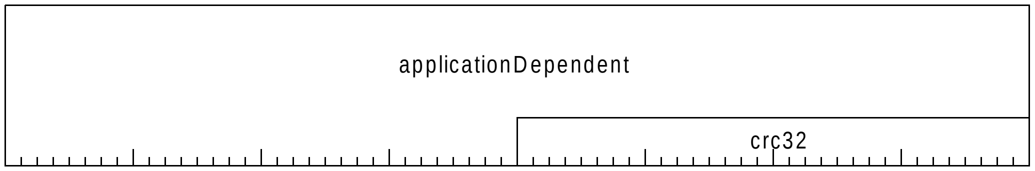


Figure 22—Payload format

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.

4.1.6 Arbitration packets

4.1.7 Arbitration packets

An arbitration packet is a standardized instance of the basic header, which starts with 64-bit *resultMacAddress* and *requestMacAddress* components, as illustrated in figure 23.

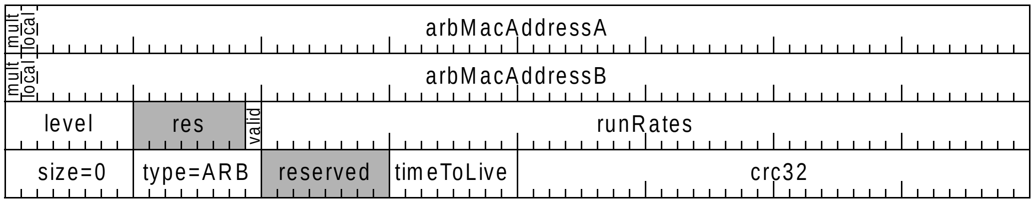


Figure 23—Arbitration packet format

The 64-bit ***arbMacAddressA*** field identifies the *selected* node, the source and target node for this packet. The 64-bit ***arbMacAddressB*** field serves two purposes, as follows:

low	For the lower levels, *arbMacAddressB* identifies the highest-precedence node. The intent is to identify the assertive node with the lowest internal *runRate* parameter.
high	For the higher levels, *arbMacAddressB* identifies the upstream node (the transmitter link). The intent is to readily transfer control to the recently-higher-precedence upstream node.

The 8-bit ***level*** (priority level) field identifies the asserted priority level, in terms of equivalent *fifoBC* fill depth. The ***valid*** bit is 1 or 0, when the following *runRates* value is valid or void respectively. The 7-bit ***reserved*** field shall be reserved.

The 48-bit ***runRates*** field is a copy of the asserted *runRate* field; typically the node with the most lagging class-C traffic.

The 8-bit ***size*** field shall be zero; the 8-bit ***type*** field specifies the type of this header, as specified in tablexx. The 8-bit ***reserved*** field shall be reserved. The 8-bit ***timeToLive*** field and the 32-bit ***crc32*** field are specified in xx.

5. Arbitration protocols

Arbitration protocols are based on the policing of offered traffic, based on the class of the traffic and the provisioned 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 Arbitration protocols

5.1.1 Arbitration signal flow

For efficiency, arbitration signals and data packets normally flow in the opposite direction, as illustrated in figure 24. Arbitration signals for top-run transmissions (solid lines) are sent on the bottom run (dotted lines), as illustrated in the left half of figure 24. Arbitration signals for bottom-run transmissions (solid lines) are sent on the top run (dotted lines), as illustrated in the right half of figure 24.

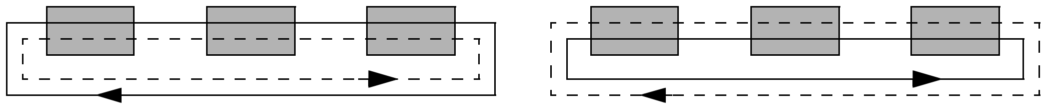


Figure 24—Opposing packet and arbitration paths

Arbitration on nonloop topologies attempts to send arbitration and data in opposite directions, but these attempts are less effective, as illustrated in figure 25. 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.

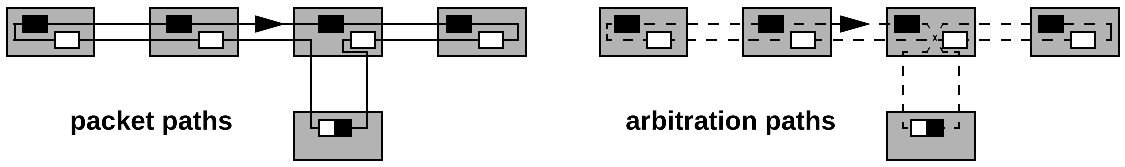


Figure 25—Reversed packet and arbitration flows

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.

5.1.2 Arbitration signals

The key element in the design of the arbitration protocols is the low-latency transmission of arbitration-related information between nodes. Arbitration information includes the following:

1) *level*. An 8-bit congestion indication transmitted from a congested upstream node.
2) *runRate*. An indication running transmission rate of class-C traffic of a class-C congested node.

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 arbitration indications are fault tolerant, in that special fault-restoration protocols are unnecessary.

5.2 Attach point queues

A typical attach point has class-A and class-B/class-C components, as illustrated in the top and bottom halves of figure 26 respectively. The *sendA* component is the class-A transmit queue, whose servicing is rate limited by the *policeA* circuitry. The *sendB* component is the class-B transmit queue, whose servicing is rate limited by the *policeB* circuitry.

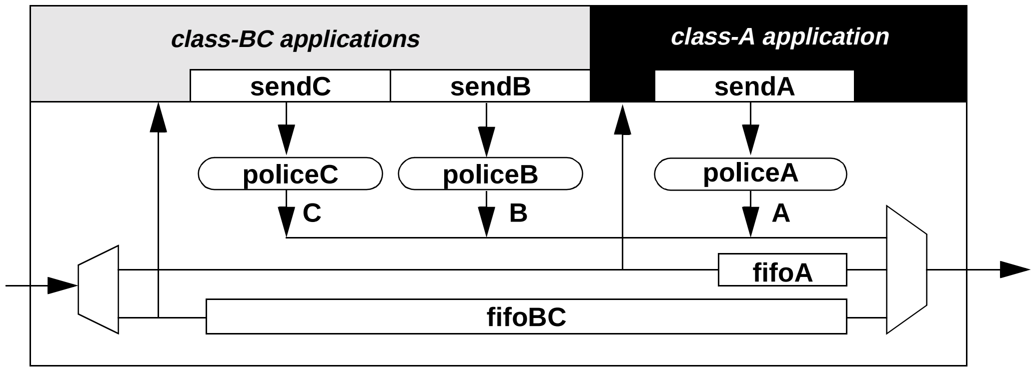


Figure 26—Attach point routing

The *sendC* component is the class-C transmit queue, whose servicing is rate limited by the *policeC* circuitry. The *policeC* component identifies the prioritized C_b and residual C_c traffic respectively.

When passing-through class-A traffic cannot be immediately retransmitted, this traffic is queued in *fifoA*. When passing-through class-B or class-C traffic cannot be immediately retransmitted, this traffic is queued in *fifoBC*. The intent is to reduce the class-A latencies by allowing their passage to preempt previously queued class-B or class-C traffic.

The send buffers are located in the client, as opposed to the MAC, so that packets can be conveniently shuffled, inserted, or deleted while awaiting transmission. The queue management protocols are application dependent and beyond the scope of this standard, but queue-level indications are passed across the Client-to-MAC interface.

5.2.1 Bypass FIFOs

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

The purpose of the ***fifoBC*** buffer is to hold portions of class-BC packets that are received during class-A packet transmissions. Multiple class-A packet can be sent before the *fifoBC* can be emptied. Therefore, an optimally sized *fifoBC* buffer depends on the arbitration parameters and ringlet circulation times (typically much larger than the *fifoA* buffer size.

NOTE—When used within large metropolitan or small-state environments, the *fifoBC* 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.

5.2.2 Policing

The ***policeA*** gating blocks the class-A traffic ready-to-send indication when transmissions would exceed their provisioned limits, as follows:

```
if (Status(macPtr->sendA)==EMPTY&&macPtr->creditSync>0)
  macPtr->creditA= 0;
else
  macPtr->creditA+= macPtr->rateOfClassA*timeDelay-sizeOfTransfer;
```

The ***policeB*** gating blocks the class-B traffic ready-to-send indication when transmissions would exceed their provisioned limits, as follows:

```
if (Status(nodePtr->sendB)==EMPTY&&nodePtr->creditSync>0)
  macPtr->creditB= 0;
else
  macPtr->creditB+= macPtr->rateOfClassB*timeDelay-sizeOfTransfer;
```

The ***policeC*** circuit provides a class-P (prioritized) indication when this node is lagging behind (i.e., others have sent more class-C traffic) and a class-C indication otherwise. This is the key towards ensuring fair distribution of class-C traffic, by publishing the lagging class-C opportunistic traffic rates. Others with higher class-C transmission rates are then inhibited from sending these as prioritized class-C_b traffic, until the published traffic rate has been reached.

```
    macPtr->thisRunRate+= sizeOfTransfer;
```

5.3 Queue selection protocols

5.3.1 Revised arbitration prarameters

The arbitration parameters contained within the arbitration control packets have *macAddressA* and *ringLevel* values. The arbitration packet is normally distributed to others, but can also be returned to (and stripped by) its source. The generation of the effective incoming *macAddress* and *ringLevel* values therefore requires adjustments to estimate the upstream-node values, when control arbitration circulates to its source, as listed in table 3.

Table 3—Revised arbitration parameters

conditions		**Row**	**effective incoming parameters**	
arbMacAddressA	**ringLevel**		**macAddress**	**level**
thisMacAddress	≥1/2	1	arbMacAddressB	MINIMUM(0,MAXIMUM(depth,less))
	—	2	thisMacAddress	
—	—	3	arbMacAddressA	ringLevel

#define less (ringLevel-DEPTH/4)

Row 1: When arbitration control returns to self, and the previous node is readily identified, that node becomes the arbiter and the level is reduced by 1/4 of the *fifoBC* depth.

Row 2: When arbitration control returns to self, and the previous node cannot be identified, this node becomes the arbiter and the level is reduced by 1/4 of the *fifoBC* depth.

Row 3: When arbitration control comes from another, that node's identifier and assertion level are used as input values.

5.3.2 Arbiter selection priorities

The node with the highest asserted *level* is called the arbiter, since its assertions affect transmissions by others. The observed *level* value is related to the effective *depth* of the fifoBC component, as listed in table 4. The intent is to update the circulating *arbsMacAddressA* as well as *ringLevel* values, so that the arbitration control packet can be identified when its returns to the arbiter.

Table 4—Arbiter selection priorities

conditions	Row	outgoing values	
depth		**arbsMacAddressA**	**ringLevel**
depth≥level	1	thisMacAddress	depth
—	2	macAddress	MIN(level,MAX(depth+DEPTH/4,0))

#define depth MAXIMUM(depthBC, MININIM(debthBC+depthB, DEPTH/2))

Row 1: This node's fifoBC depth exceeds the previously observed level, so its identify and *depth* values become the asserted *arbsMacAddressA* and *ringLevel* values respectively.

Row 2: This node's fifoBC depth remains below the previously observed level, so the observed identity and revised *level* values become the asserted *arbsMacAddressA* and *ringLevel* values respectively. The revised *level* value equals the observed *ringLevel* value, unless this node's *depth* is dramatically smaller.

5.3.3 RunRate assertion priorities

The RunRate assertion protocols distribute the smallest of the observed run rates on the attached nodes, as listed in table 5.

Table 5—Secondary selection protocols

conditions				Row	outgoing parameters		
ringRunLevel	**arbsMacAddressB**	**behind**	**classA**		**ringRunValid**	**ringRunRate**	**arbsMacAddressB**
≥DEPTH/2	—	—	—	1	classC	thisRunRate	thisMacAddress
<DEPTH/2	thisMacAddress	—	—	2			
	—	1	1	3			
		—		4	-unchanged-		

Row 1: The previous node is so congested that the asserted *ringRunRate* value has no meaning, so this node's *thisRunRate* value is asserted.

Row 2: The asserted *ringRunRate* value is known to be stale, because its source is this node, so this node's current *thisRunRate* value is asserted.

Row 3: The asserted *ringRunRate* value indicates this node is behind, and class-C traffic is ready to send, so this node's current *thisRunRate* value is asserted.

Row 4: Another node's transmission is preferred, so the observed runRate value is passed through.

5.3.4 Secondary selection priorities

When no class-C traffic is ready to be sent, the MAC's *thisRunRate* value tracks advancing the observed *ringRunRate* values, as listed in table 6. The intent is to avoid accumulation of transmission credits while operating in the not-ready-to-transmit condition.

Table 6—Secondary selection protocols

current values			Row	updates
ringRateValid	**ringRunRate**	**classA**		**thisRunRate**
1	>thisRunRate	0	1	ringRunRate
—	—	—	2	-none-

Row 1: This node is behind others but has nothing to send, so update its internal thisRunRate value.

Row 2: Until conditions allow, the node's *thisRunRate* value remains unchanged.

5.3.5 Queue selections

Queue selections, as listed in table 7. The intent is to begin filling nearby upstream *fifoBC* buffers to provide idles for congested downstream nodes.

Table 7—Queue selections

conditions			Row	queue enabled				
level	**depth**	**behind**		**FIFO_A**	**SEND_A**	**SEND_B**	**SEND_C**	**FIFO_BC**
>depth	—	—	1	yes	yes	-	-	-
—	depth≥¾	—	2	yes	yes	-	-	yes
	depth<¾	—	3	yes	yes	yes	-	
	depth<½	0	4					
		1	5	yes	yes	yes	yes	
	depth=0	—	6					

#define behind ...

Row 1: When the downstream node appears more congested, assistance (in the form of unused symbols) is provided. This assistance involves the transmission of only class-A traffic, effectively replacing class-B or class-C traffic with idle symbol transmissions.

Row 2: A node with a mostly full fifoBC is disallowed to transmit class-B or class-C traffic. The intent is to stiffle all non-class-A transmissions while congested class-A traffic cannot be transmitted.

Row 3: A node with a half full fifoBC is disallowed to transmit class-C traffic, regardless of its lagging class-C traffic transmissions. The intent is to stiffle all class-C transmissions while congested class-B traffic cannot be transmitted.

Row 4: A node that is not behind, and has a partially filled fifoBC, is disallowed to transmit class-C traffic. The intent is to enable transmissions from nodes' that are behind in their class-C transmissions.

Row 5: A behind node is allowed to transmit class-C traffic when its fifoBC is mostly empty, because further fifoBC symbols remain to provide assistance (if need be) for others' future class-B or class-A transmissions.

Row 6: When fifoBC is empty, this node is uncongested and other assistance is not requested. Transmission of all traffic (including class-C) is allowed. Although these opportunistic transmissions may be unfair, class-C transmissions update the local *thisRunRate* value, with the effect of ensuring fairness of the cumulative packet transmission rates.

6. Maintenance activities

6.1 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).

Annexes

Annex A

(informative)

Bibliography

The following publications are recommended as background material for understanding the objectives behind this standard:

[B1] IEEE Std 1596-1992, Scalable Coherent Interface.[2]

[B2] IEEE Std 1394-1995, High Performance Serial Bus.[1]

Annex B

(informative)

Initialization sequences

B.1 Initialization overview

NOTE—This subannex is highly preliminary and subject to revision. This initial writeup attempts to communicate the concept of reset packets, which force nodes out of their normal operational state, and flush packets that confirm the absence of residual reset packets.

Reset sequences are based on the selection of a distinctive node, called the root, and the formation of a tree topology beneath the root. During the reset, multiple nodes (called candidates) may attempt to become the root, but only the node with the largest ringID identifier succeeds. Unsuccessful candidates quickly become proxies for the currently active candidates, forwarding candidates' ringID identifiers to others. The remaining ringID identifiers are sent between the active candidates, allowing unsuccessful candidates to withdraw, until only one candidate remains and the topology associated with that candidate has been established.

The reset sequence is followed by a flush sequence, which propagates flush indications through the recently established ringlet(s). The flush indications restore nodes to their operational states while ensuring the removal of residual reset indications. The root node is responsible for starting ringlet communications after the flush indications have circulated.

Although the reset state machines are relatively simple to define, the reset/flush sequences can be somewhat difficult to understand. For that reason, these sequences are illustrated for a variety of representative topologies. Although the set of representative topologies is relatively complete, more complex topologies are often expected to be implemented.

B.2 Clean-slate initialization

A clean-slate initialization, called a net reset, is necessary when node have unknown or potentially inconsistent states, which could occur after a power loss or major communication errors. Changing from unknown to initialized states involves the transmission of reset messages between nodes. For simplicity, the following discussions assume that nodes have entered the reset state concurrently.

B.2.1 Duplex ring initialization

B.2.1.1 Duplex ring reset

In this example, nodes on the ring send initial reset messages to their neighbors, as illustrated in figure B.1. The intent of the reset packets is to establish a ringlet topology while forcefully placing other nodes into a known initialized state.

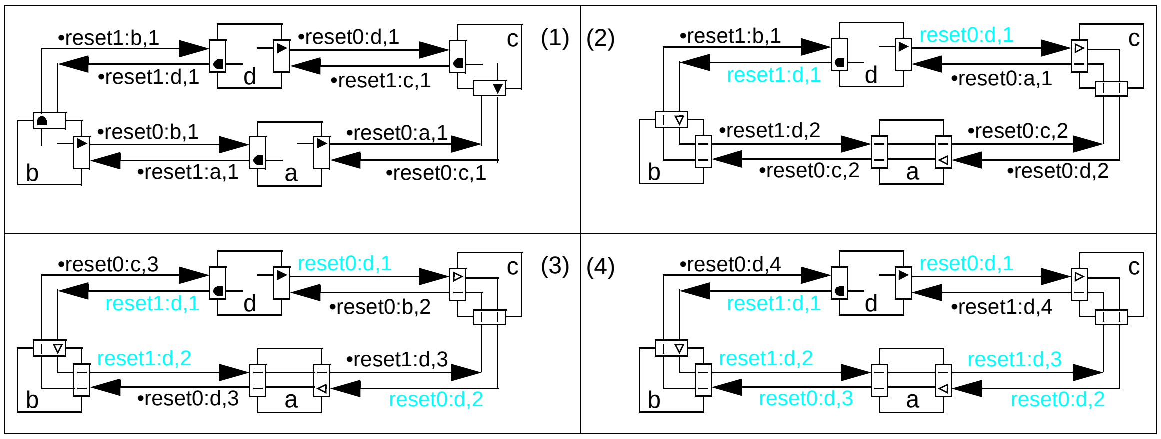


Figure B.1—Duplex ring reset

Each node's behavior is biased by its distinctive ringID identifier. Shortly after the resets (2) begin to propagate, node-C, node-B, and node-A have deferred to node-D, node-D, and node-C respectively. Having observed the presence of a higher authority, these nodes simply forward reset packets on behalf of that authority.

Shortly thereafter (3), resets propagate through their adjacent neighbors: node-A also defers to node-D. Having observed the presence of a higher authority, all but node-D now forward reset packets on its behalf.

After the reset packets (4) have fully circulated, the root node senses that condition and generates *flush* indications to remove the disruptive *reset* indications, as described in the following subclause.

B.2.1.2 Duplex ring flush

Before normal operations can begin, the reset indications must be flushed from the interconnect. Fushing begins at the root node, where flush packets are generated until flush packets have been returned, as illustrated in figure B.2.

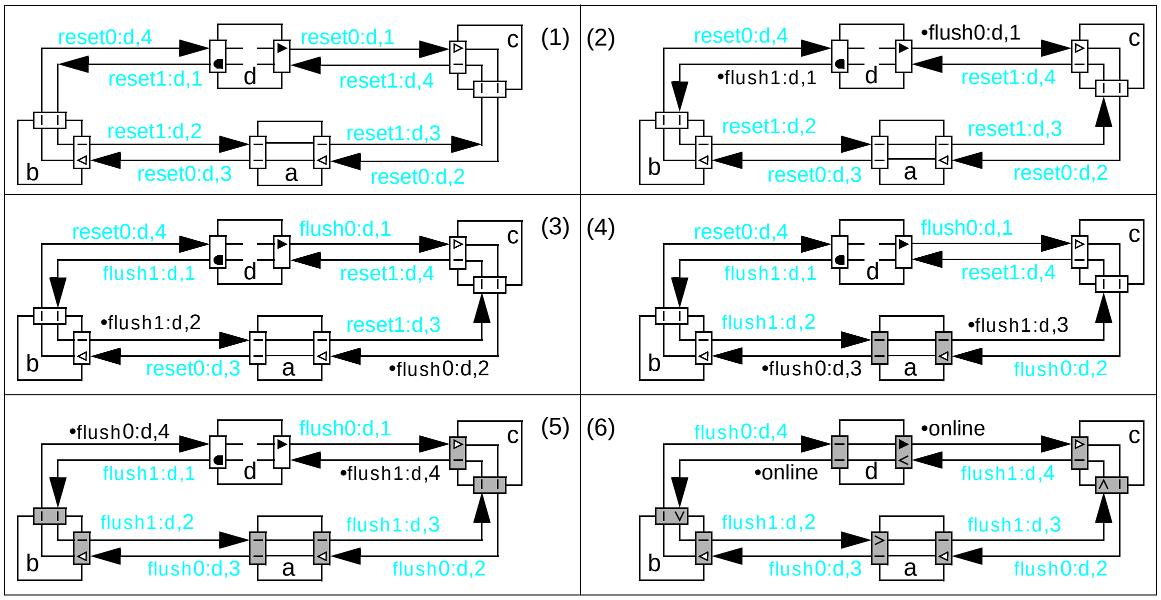


Figure B.2—Duplex ring flush

B.2.2 Severed-ring initialization

B.2.2.1 Severed-ring reset

In this example, nodes on a severed ring send initial reset messages to their neighbors, as illustrated in figure B.3. Again, the intent of the reset packets is to establish a ringlet topology while forcefully placing other nodes into known initialized states.

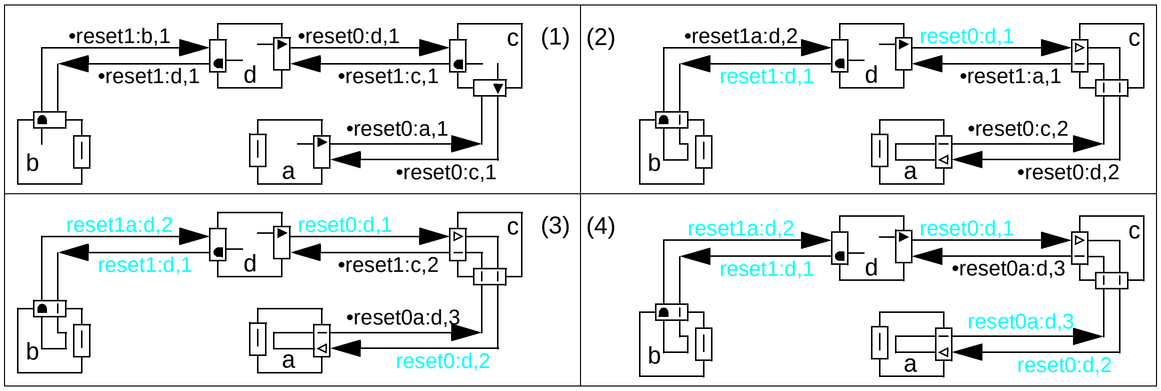


Figure B.3—Severed-ring reset

Each node’s behavior is biased by its distinctive ringID identifier. Shortly after the resets (2) begin to propagate, node-C, node-B, and node-A have deferred to node-D, node-D, and node-C respectively. Having observed the presence of a higher authority, these nodes simply forward reset packets on behalf of that authority.

Shortly thereafter (3), resets propagate through their adjacent neighbors: node-A also defers to node-D. Having observed the presence of a higher authority, all but node-D now forward reset packets on its behalf.

After the reset packets (4) have fully circulated, the root node senses that condition and generates *flush* indications to remove the disruptive *reset* indications, as described in the following subclause.

B.2.2.2 Severed-ring flush

When the returned reset indications match those reset indications sent from the root, the reset is known to have stabilized. Before normal operations can begin, the reset indications must be flushed from the interconnect. Fushing begins at the root node, where flush packets are output until flush packets are returned, as illustrated in figure B.4.

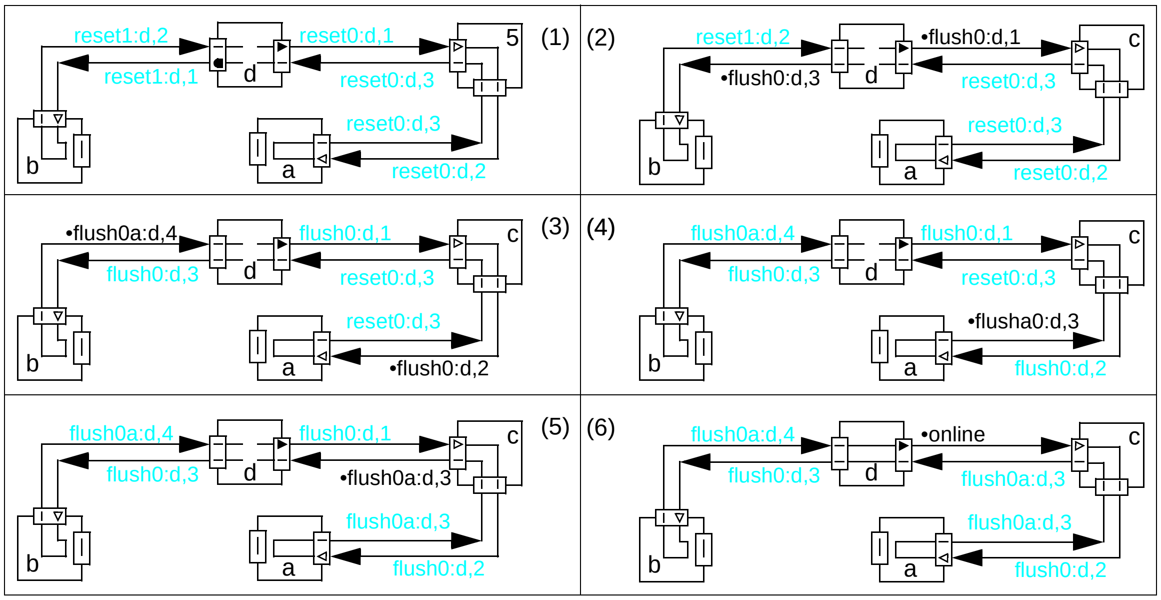


Figure B.4—Severed-ring flush

B.2.3 Half-severed initialization

B.2.3.1 Half-severed reset

In this example, nodes on a half-severed ring send initial reset messages to their neighbors, as illustrated in figure B.5. Again, the intent of the reset packets is to establish a ringlet topology while forcefully placing other nodes into known initialized states.

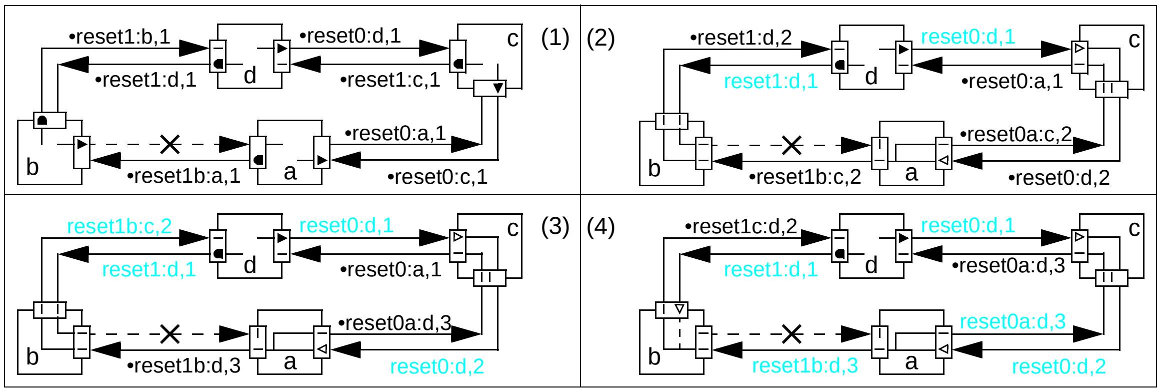


Figure B.5—Half-severed reset

Each node’s behavior is biased by its distinctive ringID identifier. Shortly after the resets (2) begin to propagate, node-C, node-B, and node-A have deferred to node-D, node-D, and node-C respectively. Having observed the presence of a higher authority, these nodes simply forward reset packets on behalf of that authority.

Shortly thereafter (3), resets propagate through their adjacent neighbors: node-A also defers to node-D. Having observed the presence of a higher authority, all but node-D now forward reset packets on its behalf.

After the reset packets (4) have fully circulated, the root node senses that condition and generates *flush* indications to remove the disruptive *reset* indications, as described in the following subclause.

B.2.3.2 Half-severed flush

When the returned reset indications match those reset indications sent from the root, the reset is known to have stabilized. Before normal operations can begin, the reset indications must be flushed from the interconnect. Fushing begins at the root node, where flush packets are output until flush packets are returned, as illustrated in figure B.6.

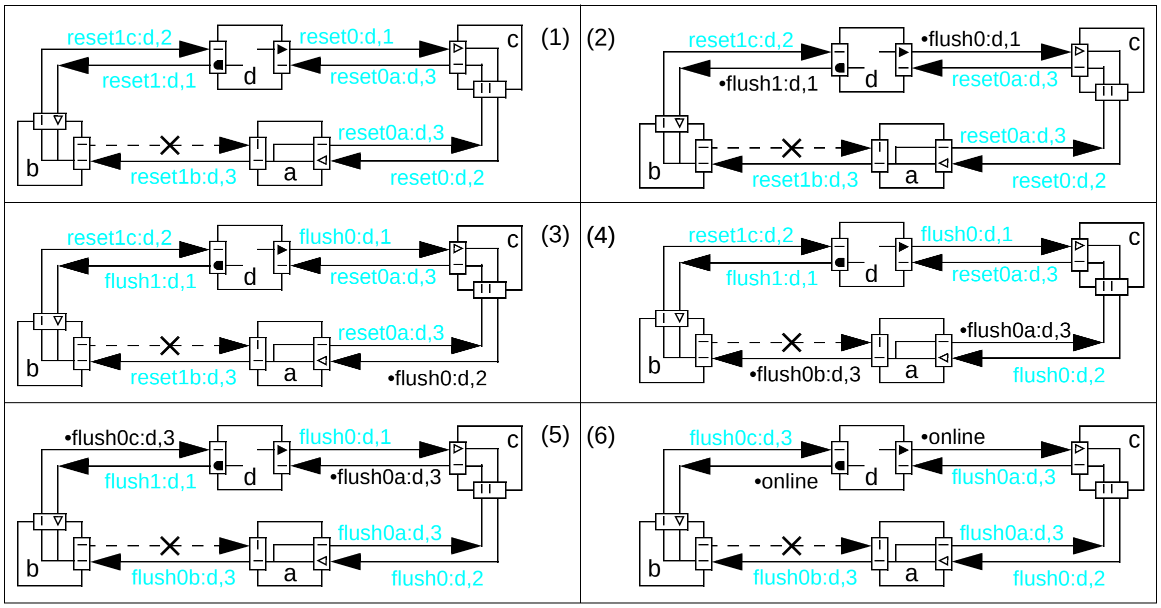


Figure B.6—Half-severed flush

B.3 Merged topologies

TBD—Two nets are joined into one.

B.4 Split topologies

TBD—One net is split into two.

B.5 Failed link

TBD—A link fails, inducing wrapping and steering.

B.6 Recovered link

TBD—A link recovers; steering changes to ringlet selection.

Annex C

(normative)

Physical layer dependent services

C.1 Wallclock synchronization

High quality isochronous data transfers, such as voice, audio, or video, rely on operating the initial source and final destination with the same (or close to the same) digital sampling frequencies. To facilitate such synchronized end-to-end behaviors, the transport is assumed to be capable of transferring an accurate global time reference. Some physical layers may provide the global time reference directly; others may not. This subannex illustrates how a global time referenc can be readily provided by MAC-level hardware, when not directly supported by the physical layer.

C.1.1 Wallclock calibration

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 specialized class-A packets, called clockSync packets, as described in this subclause.

The root node is responsible for generation of clockSync packets. All nodes (root as well as nonroot) are responsible for measuring the clockSync propagation time through themselves. Clock deviations are sampled in cycle N and calibrations are performed in cycle N+1. Clock sampling involves through-node delays measurements and sampling of the nodes *clockTime* value at its transmitter, as illustrated in figure C.1.

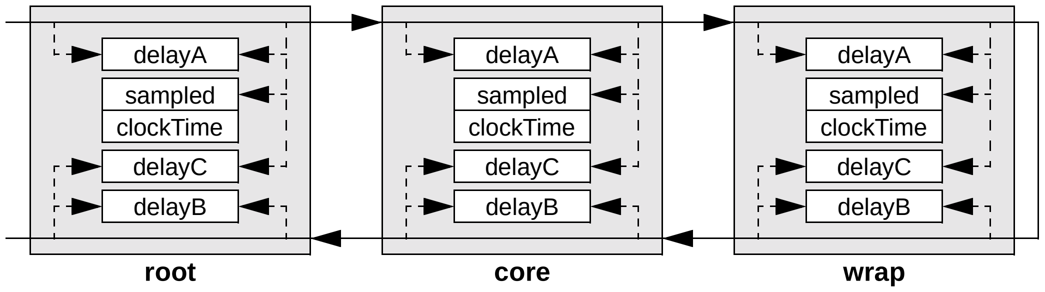


Figure C.1—Clock and delay measurements

The behaviors on transmissions to the right is as follows:

1) Root. The root forwards its right-side and left-side averaged transmit/receive time estimates:
   a) Right. The root sends the average of the observed right-side times, as follows:
      $timeSend_{right}= sampled+(delayC–delayB)/2$
   b) Left. The root sends the average of the observed left-side times, as follows:
      $timeSend_{left}= sampled+(delayA+delayB)/2$
2) Core. The core computes its clock deviations and forwards an adjusted time estimate:
   a) Deviation. The node computes its clock deviation, as follows:
      timeSink= sampled+(delayC–delayA)/2;
      timeDiff= timeSend–timeSink;

b) Core. The core sends the average of the observed right-side times, as follows:
$timeSend_{right} = timeSend_{left} - (delayA + delayB)/2$

3) Warp. The node computes its clock deviation, as follows:
$timeSink = sampled + (delayC - delayA)/2;$
$timeDiff = timeSend - timeSink;$

C.1.2 Wallclock adjustments

The wallclock measurements in cycle N are used to adjust the clock-slave wallclock values in cycle N+1, as specified in equation 2. Initial synchronization involves setting the node's *clockTime* value, to minimize the clock-value lock-up delays. Maintaining synchronization involves clock-rate adjustments, to avoid *clockTime* discontinuities.

```
#define THRESHOLD ONE_SECOND/8000       // Adjust after 8KHz interval
#define TICK (CLOCK_NOMINAL/5000)       // 200PPM overcomes 100PPM inaccuracy
delta= timeSink-timeSend;
if (Magnitude(delta)>(THRESHOLD/2))
  clockTime+= delta;
else
  clockRate+= difference>0 ? TICK:-TICK;                                      (2)
```

C.2 Power reduction shutdown sequences

C.2.1 Child initiated shutdown

On an operational network (1), a link shutdown may be triggered by a child, by sending *shutDown0* indications (2) to its parental neighbor, as illustrated in figure C.2. The parent initiates (3) the shutdown handshake (4) with the child. After the handshake, the parental disables (5) its leaf-side link; the shutdown concludes when both of the link transmitters (6) have been disabled.

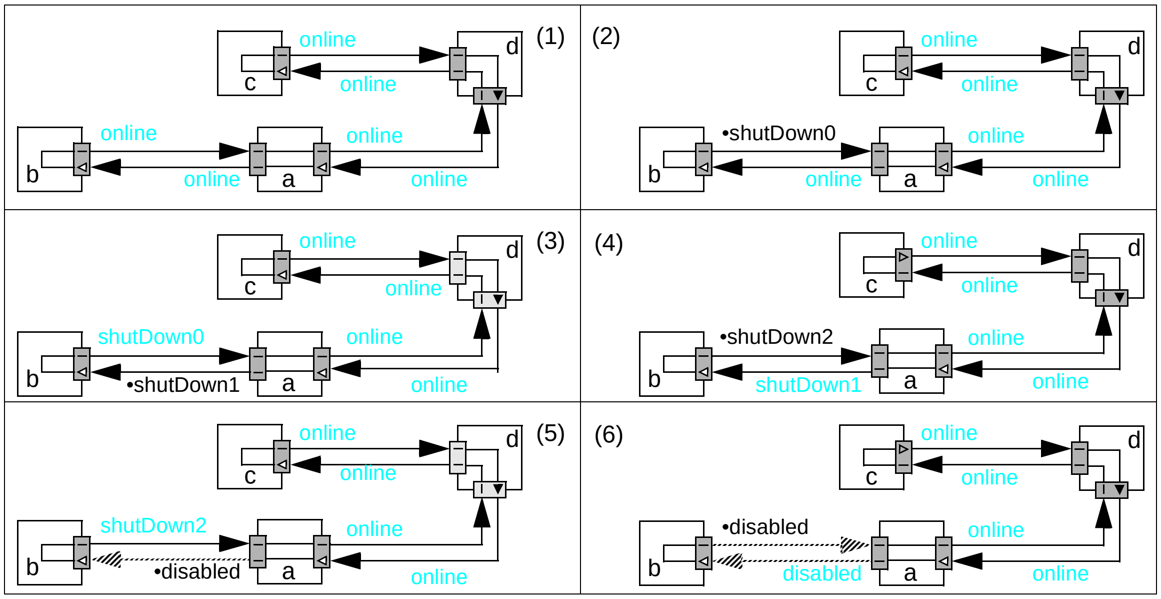


Figure C.2—Shutdown activation (child initiated)

Although the link transmitters (including their crystal clocks) can be fully disabled, a minimal portion of a disabled link is required to remain active for the purposes of monitoring incoming signals. If the monitor detects cable-present and signal-present conditions, additional logic must be activated, to decode possibly valid signals from one's neighbors.

C.2.2 Parent initiated shutdown

On an operating system (1), the parent initiates (2) the shutdown handshake (3) with the child, as illustrated in figure C.3. After the handshake, the parental node disables (4) its child-bound link; the shutdown concludes when both of the link transmitters (5) have been disabled.

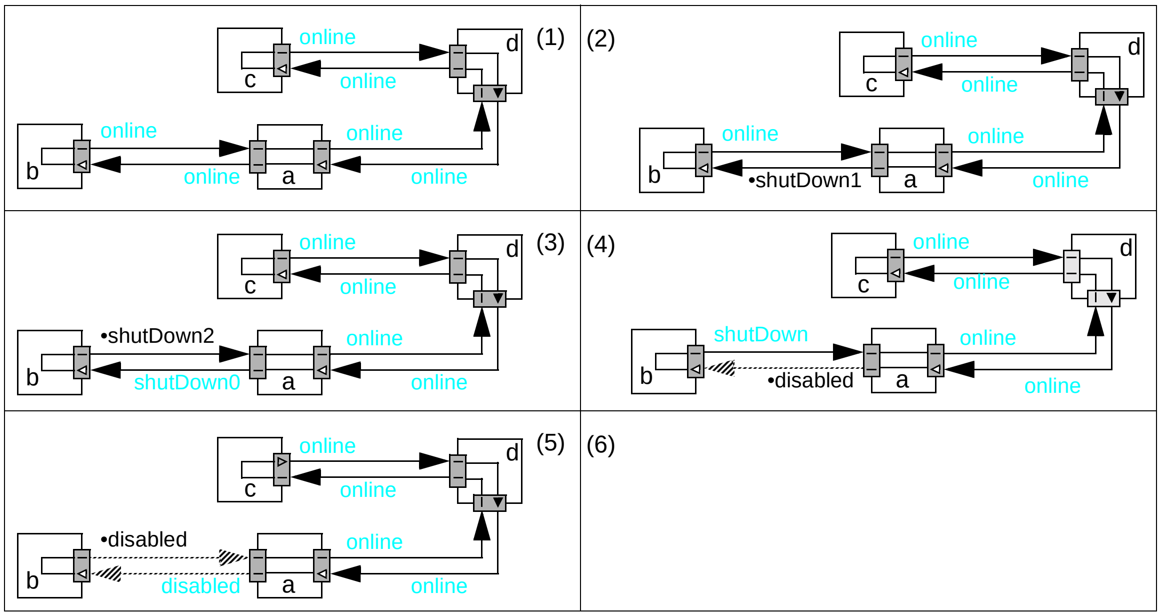


Figure C.3—Shutdown activation (parent initiated)

C.2.3 Child initiated recovery

Shutdown recovery (1) may be triggered by a child, by sending *wakeup0* indications (2) to its parental neighbor, as illustrated in figure C.4. The parent initiates (3) the wakeup handshake (4) with the child. After the handshake, the parental enables (5) its leaf-side link; the wakeup concludes when both of the link transmitters (6) have been enabled.

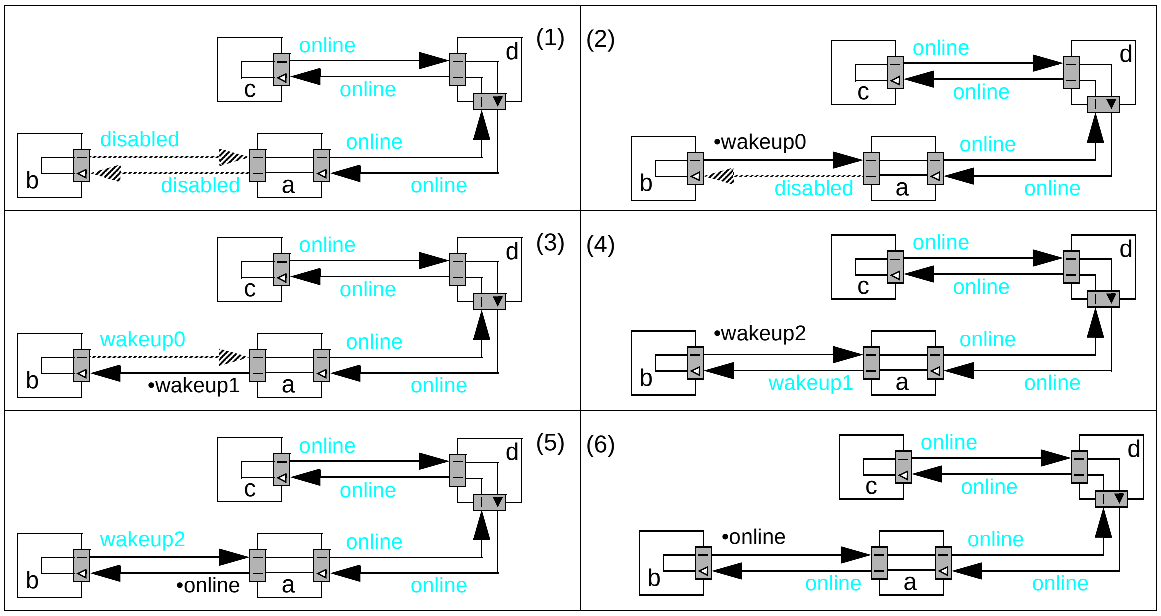


Figure C.4—Shutdown recovery (child initiated)

C.2.4 Parent initiated shutdown recovery

With a shutdown link (1), the parent may initiate (2) the wakeup handshake (3) with the child, as illustrated in figure C.5. After the handshake, the parental node enables (4) its child-bound link; the shutdown concludes when both of the link transmitters (5) have been enabled.

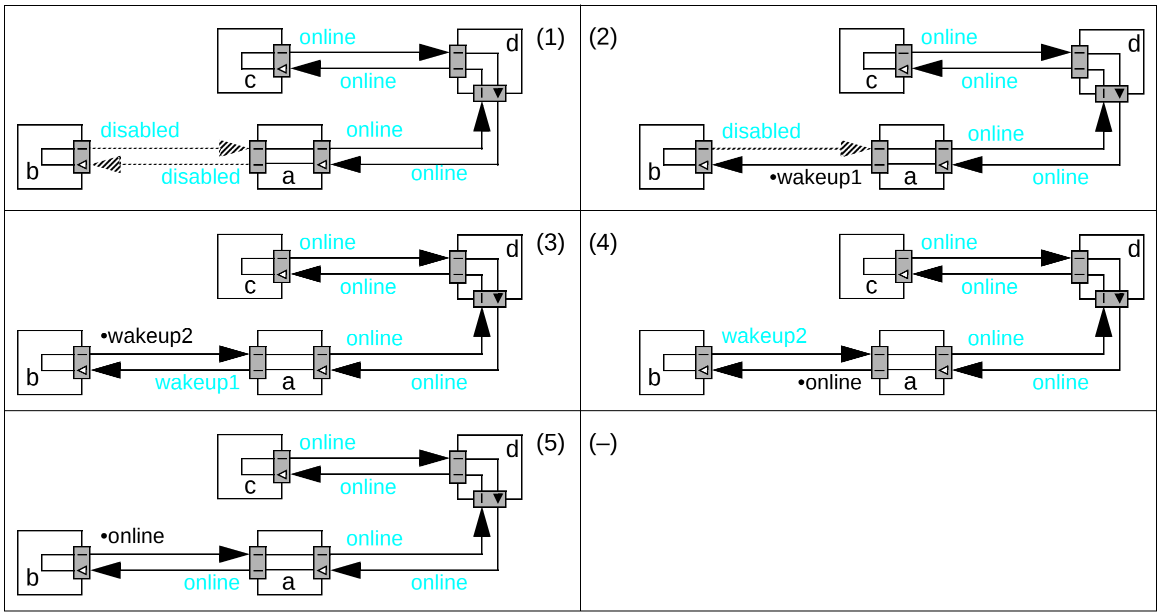


Figure C.5—Parent initited shutdown recovery

Annex D

(informative)

Physical encoding

D.1 8/10 coding

D.1.1 Idle symbol format

The between packet 32-bit idle symbols provide packet framing and control information, as illustrated in figure D.1.

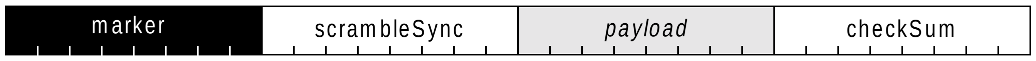


Figure D.1—Idle format

The ***marker*** is a physical-layer dependent control character that distinctively identifies the idle symbol. The marker also provides a start-of-symbol framing indication.

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

The 8-bit ***idlePayload*** field value transports the standard protocol bits, as specified in xx.

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

Annex E

(informative)

Protocol encapsulation

E.1 IEEE Std 1394 SerialBus encapsulation

A protocol for efficiently encapsulationg SerialBus packets is considered. The intent is to validate the flexibility of these RPR protocols to support such uses, not to restrict the encapsulation method.

E.1.1 SerialBus extended headers

The SerialBus extended header preceds the SerialBus header a distinctive protocol identifier and *timeOfDeath* label, as illustrated in figure E.1. A more efficient encoding would use a typeM identifier in the previous header to identify the extended header, if such a code value could be defined.

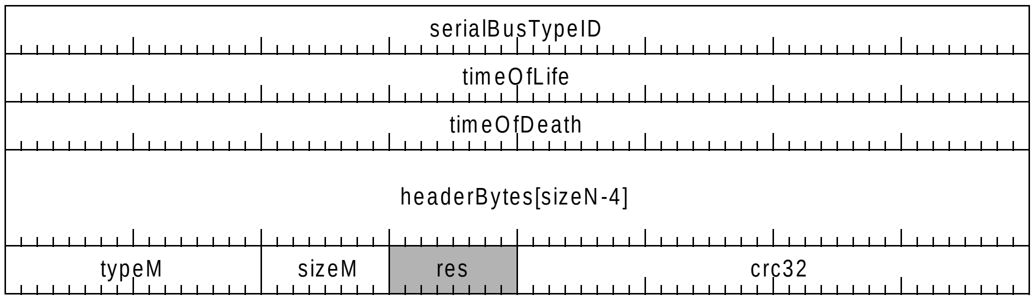


Figure E.1—SerialBus extended header

The 64-bit ***serialBusTypeID*** is an EUI-64 value that distinctively identifies this as a SerialBus extended header.

The 64-bit ***timeOfLife*** is a time value, represented in seconds and fractions-of-seconds, that specifies the time at which this packet's affiliated transaction was first initiated.

The 64-bit ***timeOfDeath*** is a time value, represented in seconds and fractions-of-seconds, that specifies the time after which this packet shall be discarded.

The following ***headerBytes[]*** are used to encapsulate the SerialBus header.

E.1.2 SerialBus directed flow control

TBD.

E.1.3 SerialBus multicast flow control

TBD.

Annex F

(normative)

Code listing

```
//                                                                                                   1         1         1         1
//        1         2         3         4         5         6         7         8         9         0         1         2         3
// 3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012

#define BYTES_PER_SECOND     (1250000000)                                        // Tranmission rate in bytes/second, 10 Gb/s example
#define BYTES_PER_SYMBOL     (8)                                                 // Number of bytes per symbol (credits are per symbol)
#define LONG_CLASSA_BYTES     (1600)                                             // Number of bytes in long class-A packet
#define LONG_CLASSB_BYTES     (18000)                                            // Number of bytes in longest class-B packet

#define MAX_NODES_ON_RING    (256)                                               // Maximum number of nodes attached to each ringlet
#define LOOP_DELAY_BYTES     (1250000)                                           // Number of loop speed-of-light delays, 1ms example

#define FIFOS_CLASSA_BYTES   (LONG_SYNCH_BYTES*MAX_NODES_ON_RING)                // Ringlet delays due to class-A packet lengths
#define TOTAL_CLASSA_BYTES   (FIFOS_SYNCH_BYTES+LOOP_DELAY_BYTES)                // Overall delays due to packets and speed-of-light
#define LIMIT_CLASSA_BYTES   (TOTAL_SYNCH_BYTES*SYNCH_RATE_LIMIT)                // Synchronous traffic credit limitations

#define FIFOS_CLASSB_BYTES   (LONG_ASYNC_BYTES*MAX_NODES_ON_RING)                // Ringlet delays due to class-A packet lengths
#define TOTAL_CLASSB_BYTES   (FIFOS_ASYNC_BYTES+LOOP_DELAY_BYTES)                // Overall delays due to packets and speed-of-light

typedef unsigned char       uInt1;
typedef unsigned short      uInt2;
typedef unsigned int        uInt4;
typedef unsigned long long  uInt8;
typedef signed short        sInt2;
typedef signed int          sInt4;
typedef signed long long    sInt8;

#define MINIMUM(a,b) ((a)<(b) ? (a):(b))
#define MAXIMUM(a,b) ((a)>(b) ? (a):(b))

enum {FIFO_BC=0X01, CLASS_C=0X02, CLASS_B=0X04, CLASS_A=0X08, FIFO_A=0X10};
#define DEPTH4 256                                                               // Normalized full-depth FifoBC
#define DEPTH3 ((DEPTH4*3)/4)
#define DEPTH2 ((DEPTH4*2)/4)
#define DEPTH1 ((DEPTH4*1)/4)
#define SCALED 65536

typedef struct {
    uInt1 presenceOfClassA;                                                      // Presence of class-A traffic
    uInt1 depthOfClassB;                                                         // Applicationís class-B congestion level
    uInt1 presenceOfClassC;                                                      // Presence of class-C traffic
} AppInfo;
typedef struct {
    uInt4 thisRunRate;                                                           // The weighted running byte-transfer count
    uInt1 depthBC;                                                               // The current fifoBC depth
    sInt8 creditA;                                                               // Total class-A credits, SCALED units
    sInt4 rateOfClassA;                                                          // Fractional class-A rate, SCALED units
    sInt8 creditB;                                                               // Total class-B credits, SCALED units
    sInt4 rateOfClassB;                                                          // Fractional class-B rate, SCALED units
} RunInfo;
```

```
// The arbitration information contained within each node
typedef struct {
    uInt8 thisMacAddress;                                                         // The node's globally unique EUI-64
    RunInfo runInfo[2];                                                           // Run specific information
    AppInfo *appInfoPtr[2];                                                       // Fractional class-B rate, SCALED units
} MacInfo;

// The arbitration contained within opposing-run class-A control packets
typedef struct {
    uInt8 arbsMacAddressA;                                                        // The MACaddress of the focal node
    uInt8 arbsMacAddressB;                                                        // A ringLevel-dependent 2nd MAC address
    unsigned ringLevel:       8;                                                  // The worst sampled congestion level
    unsigned ringRateValid:   1;                                                  // A valid ringRunRate indication
    unsigned reserved:        7;                                                  // Reserved, for future extendions
    unsigned ringRunRateHi:  16;                                                  // Worst running byte-transfer count, MSBs
    unsigned ringRunRateLo:  32;                                                  // Worst running byte-transfer count, LSBs
} ArbInfo;

void PoliceA(MacInfo *, int, uInt4, uInt4);                                       // Policing updates for class-A
void PoliceB(MacInfo *, int, uInt4, uInt4);                                       // Policing updates for class-B
void PoliceC(MacInfo *, int, uInt4);                                              // Policing updates for class-C
int ArbUpdates(MacInfo *, int, ArbInfo *, ArbInfo *);                             // Arbitration packet updates

// Argument values:
//            macPtr - Pointer to the mac information
//               run - Run on which update applies
//         timeDelay - Time since this routine was previously called
//    sizeOfTransfer - Length of packet that was sent
//
void
PoliceA(MacInfo *macPtr, int run, uInt4 timeDelay, uInt4 sizeOfTransfer)        // Class-A preemptive rate-limited negotiated streams
{   uInt8 transferSize= (uInt8)sizeOfTransfer*SCALED;
    AppInfo *appPtr= macPtr->appInfoPtr[run];
    RunInfo *runPtr= &(macPtr->runInfo[run]);

    if (appPtr->presenceOfClassA==0 && runPtr->creditA>0)                         // If no data is ready to be sent,
        runPtr->creditA= 0;                                                       // the excessive credits are discarded
    else
        runPtr->creditA+= runPtr->rateOfClassA*timeDelay-transferSize;          // Credit adjustment, based on taken symbols
}
```

```
// Argument values:
//            macPtr - Pointer to the mac information
//               run - Run on which update applies
//         timeDelay - Time since this routine was previously called
//    sizeOfTransfer - Length of packet that was sent
//
void
PoliceB(MacInfo *macPtr, int run, uInt4 timeDelay, uInt4 sizeOfTransfer)        // Class-A preemptive rate-limited negotiated streams
{   uInt8 transferSize= (uInt8)sizeOfTransfer*SCALED;
    AppInfo *appPtr= macPtr->appInfoPtr[run];
    RunInfo *runPtr= &(macPtr->runInfo[run]);

    if (appPtr->depthOfClassB==0 && runPtr->creditB>0)                          // If no data is ready to be sent,
        runPtr->creditB= 0;                                                     // the excessive credits are discarded
    else
        runPtr->creditB+= runPtr->rateOfClassB*timeDelay-transferSize;          // Credit adjustment, based on taken symbols
}

// Argument values for class-C traffic policing:
//            macPtr - Pointer to the mac information
//               run - Run on which update applies
//    sizeOfTransfer - Length of packet that was sent
//
void
PoliceClassC(MacInfo *macPtr, int run, uInt4 sizeOfTransfer)
{   RunInfo *runPtr= &(macPtr->runInfo[run]);

    runPtr->thisRunRate+= sizeOfTransfer;
}
```

```
// Specifies arbitration forwarding and transmit-queue selections
// Arguments:
//     macPtr - a pointer to the node's MAC information
//     appPtr - a pointer to the node's higher level application
//    sinkPtr - the contents of the last incoming arbitration packet
//    sendPtr - the contents of the next outgoing arbitration packet
// Returns allowed transmit-queue selections
//
int
ArbUpdates(MacInfo *macPtr, int run, ArbInfo *sinkPtr, ArbInfo *sendPtr)
{   AppInfo *appPtr= macPtr->appInfoPtr[run];
    RunInfo *runPtr= &(macPtr->runInfo[run]);
    uInt1 depthA= appPtr->presenceOfClassA;
    uInt1 classC= appPtr->presenceOfClassC;
    uInt1 depthB= MAXIMUM(MINIMUM(appPtr->depthOfClassB, runPtr->creditA),0);
    uInt1 depthBC= runPtr->depthBC;
    sInt2 sinkLevel;
    uInt8 sinkAddress, thisMacAddress = macPtr->thisMacAddress;
    uInt8 ringRunRate= ((uInt8)sinkPtr->ringRunRateHi<<32)|(sinkPtr->ringRunRateLo);
    uInt8 thisRunRate= runPtr->thisRunRate, runRate;
    int equal, depth, level, lowest, behind, backMacAddress;

    if (sinkPtr->arbsMacAddressA==thisMacAddress) {                       // Detect packets returning to self
        sinkAddress= sinkPtr->ringLevel>DEPTH2 ? sinkPtr->arbsMacAddressB:sinkPtr->arbsMacAddressA;
        sinkLevel= MINIMUM(sinkPtr->ringLevel-DEPTH1,0);
    } else {
        sinkAddress= sinkPtr->arbsMacAddressA;
        sinkLevel= sinkPtr->ringLevel;
    }

    depth= MAXIMUM(depthBC, MINIMUM(depthBC+depthB, DEPTH2));             // Packet depth includes
    if (depth>sinkLevel) {
        sendPtr->ringLevel= depth;                                        // Higher level values are asserted,
        sendPtr->arbsMacAddressA= thisMacAddress;                         // along with focal-node identification
    } else {
        sendPtr->ringLevel= MINIMUM(sinkLevel, MAXIMUM(depth+DEPTH1,0));  // Higher level values are observed
        sendPtr->arbsMacAddressA= sinkAddress;                            // and are eventually passed through
    }

    // At the lower priorities:
    //   arbsMacAddressB - identifies the arbitration focal point
    //          runRate - the smallest one has precedence
    if (sinkLevel>=DEPTH2) {                                              // For higher priorities, arbsMadcAddressB
        sendPtr->arbsMacAddressB= thisMacAddress;                         // identifies the upstream node.
    } else {
        runRate= thisRunRate+(MAXIMUM(0,DEPTH1-thisRunRate));               // The effective runRate depends on fifoBC depth
        lowest= (LowestRate(runRate, ringRunRate)==runRate);                // A ringRunRate arithmetic comparison value
        behind= sinkPtr->ringRateValid&&(classC==0||lowest==0);             // An effective higher ringRunRate value
        if (behind&&sinkPtr->arbsMacAddressB!=thisMacAddress) {
            sendPtr->ringRateValid= 1;                                      // Higher precedence ringRate values from
            sendPtr->ringRunRateHi= sinkPtr->ringRunRateHi;                 // upstream nodes are passed through
            sendPtr->ringRunRateLo= sinkPtr->ringRunRateLo;
            sendPtr->arbsMacAddressB= sinkPtr->arbsMacAddressB;
        } else {
            sendPtr->ringRateValid= (classC!=0);                            // The node with the best (e.g. lowest) runRate
            sendPtr->ringRunRateHi= runRate>>32;                            // sets the sampled ringRunRate value
            sendPtr->ringRunRateLo= (uInt4)runRate;                         // sets the sampled ringRunRate value
            sendPtr->arbsMacAddressB= thisMacAddress;
        }
        if (sinkPtr->ringRateValid&&classC==0&&lowest!=0)                   // When class-C traffic is ready, track ringRunRate
            runPtr->thisRunRate=  ((uInt8)sinkPtr->ringRunRateHi<<16)|(sinkPtr->ringRunRateLo);
    }

    if ((sinkLevel-depth)>0)                                                // Assisting: only class-A traffic is sent.
        return(FIFO_A|CLASS_A);
    if (depth>=DEPTH3)                                                      // High congestion: send class-A and fifoBC
        return(FIFO_A|CLASS_A|FIFO_BC);
    if (depth>=DEPTH2||(depth!=0&&lowest==0))                               // Half congestion: send class-A/B and fifoBC
        return(FIFO_A|CLASS_A|CLASS_B|FIFO_BC);
    return(FIFO_A|CLASS_A|CLASS_B|CLASS_C|FIFO_BC);                         // Mild congestion: send class-A/B/C and fifoBC}
}
```