

**BR062R01 (submitted for p1394.1 committee vote):
Isochronous connection management**

Dr. David V. James, Sony
3300 Zanker Road, MS SJ3C1
San Jose, CA 95134-4591
Phone: 408 955-6295
FAX: 408 955-4591
Email: dvj@alum.mit.edu

December 1, 1999

**This contribution is one of several, presented for review and incorporation.
An overall contribution, which provides an overall context for this and other contributions, is presented in BR047R10.**

Isochronous resources are managed through the use of messages sent between intermediate portals. Each intermediate portal has a proxy: the talker-bus proxy has a talker flag; the listener bus proxy has a listener tag. Use of these proxies, error recovery, message formats, and proxy storage are specified. The working group is requested to accept this content for incorporation into the draft.

1.12 Isochronous management sequences

1.12.1 Isochronous connect

Establishment of an isochronous connections begins with knowledge of the isochronous payload size. This normally involves a read of the talker node's oPCR (not illustrated) or ROM. Connection management does requires this information, but its location and format are beyond the scope of this standard.

The connection formation starts with the sending of a connect message (1) from the controller to the listener's talker-path portal, as illustrated in figure 1. Internal communications (not illustrated) allocate the necessary listener-bus isochronous resources. The connect message is forwarded (2) to the next talker-path portal, where lock transactions (3) are used to acquire isochronous resources. Internal communications (not illustrated) allocated the necessary talker-bus isochronous resources before the talker (4) is attached. The thick shaded arrow follows the path of connect messages, which flow in a listener-to-talker direction.

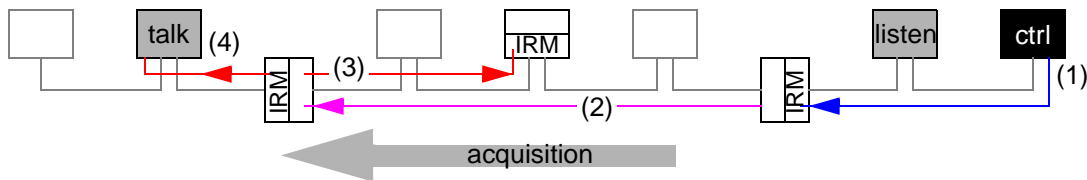


Figure 1—Connect acquisition

To simplify isochronous allocations, the IRM is forced to be in a portal or bridge-aware node. This is also necessary to trigger the generation of iso_change indications when bandwidth adjustments are made.

On the talker bus, the connect message is redirected in the reverse direction, traveling (5) in the talker-to-listener direction, as illustrated in figure 2. At the listener bus this triggers (6) the listener attachment, before the message (7) is returned to the controller.

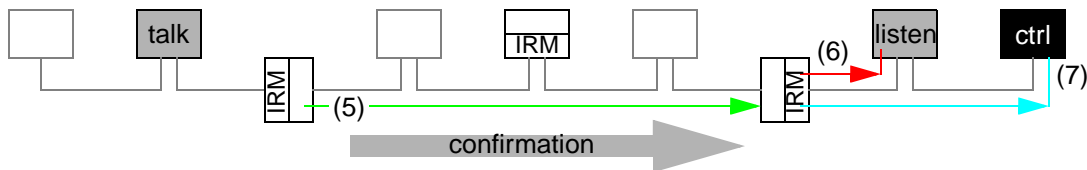


Figure 2—Connection confirmation

1.12.2 Isochronous disconnect

A disconnection starts with the sending of a disconnect message (1) from the controller to the listener's talker-path portal, as illustrated in figure 3. The listener (2) is detached before internal communications (not illustrated) release listener-bus isochronous resources. The connect message is forwarded (3) to the next talker-path portal, where lock transactions (4) release isochronous resources. The talker (5) is detached, internal communications release allocated isochronous resources, and a reply (6) is returned to the controller.

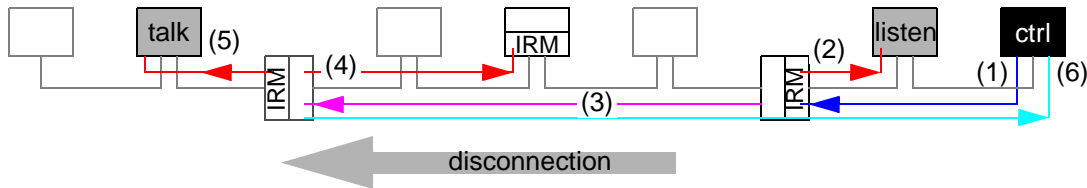


Figure 3—Isochronous disconnect

1.12.3 Bandwidth adjustments

A bandwidth adjustment starts with the talker's adjustment (1) of IRM-resident allocations, as illustrated in figure 4. This triggers distribution of a talker-bus-local isochronous-change message to local controllers (2), before the messages follow connection paths to the listeners on other buses (3,3), thereby informing all controllers of possible bandwidth changes. After receiving their notifications, controllers are expected to initiate a standard reconnection procedure, with revised bandwidth parameters.

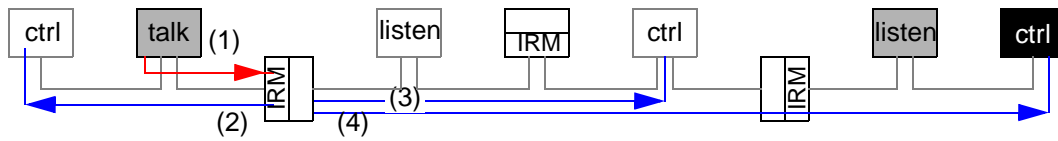


Figure 4—Isochronous-change notification

The distribution of bandwidth change messages on the talker-bus (not illustrated) involves the sending of messages from one portal to another, as described in 1.17.3.

1.13 Clock synchronization

1.13.1 Clock distribution

Although simple in theory, cascading phase lock loops, as illustrated in figure 5, complicates the dynamics of the overall system. This design approach should not be used.

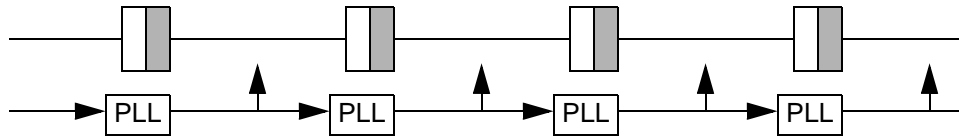


Figure 5—Overly simple clock delivery

Instead, the clock estimate is forwarded through bridges, and PLLs are used to remove jitter from the local timer on each bus, as illustrated in figure 6. This reduces the dependencies of each bus clock from PLL dynamics of the others.

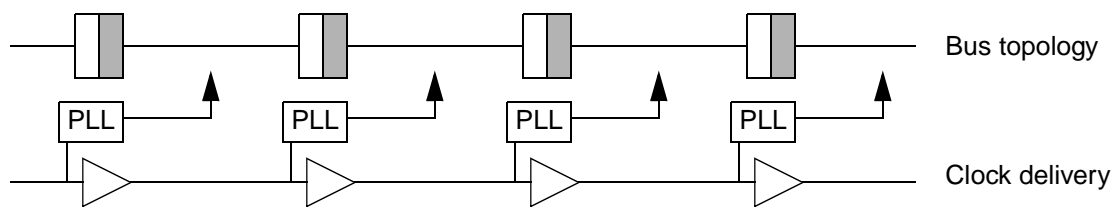


Figure 6—Desired clock distribution model

This design model mandates two time transmissions on the bus. A jitter-reduced signal marks the start of each isochronous cycle; in Serial Bus, the cycle-start packet transports this information. A second PLL source signal is repeated (with additional jitter errors) to the next bus with no PLL jitter filtering; the clockSync packet (see 8.4.2) serves this purpose.

1.13.2 Cycle-master synchronization

On remote buses, the clock-synchronization protocols proceed as follows. In every isochronous cycle a cycle-start packet is transmitted. During each cycle-start packet, all portals latch their current time value. During the following isochronous cycle, the clock-reference node transmits its previously latched value in a clockSync packet.

Other clock-slave nodes derive calculate an observed error as the difference between the clockSync packet and their latched value. That error is added to the free-running clock in the clock-slave nodes, to generate the clock-reference value that is distributed on the adjacent bus.

To minimize special routing requirements, the clockSync packets are “routed” by having alpha portals redistribute this information. The clockSync packet itself is not (strictly speaking) routed, but the observed bus-A time reference is either applied to bus-B or ignored, based on the portals’ source-routing tables.

1.13.3 Clock redistribution

To reduce the phase-lock whiplash effect, bridges are expected to redistribute a delayed clock reference and (when assuming the role of a cycleMaster) a phase-locked time in the cycle-start packet, as illustrated in figure 7. The intent is to redistribute an unfiltered (but delayed) clock reference, while phase-locking the cycle-start packet to reduce clock jitter effects.

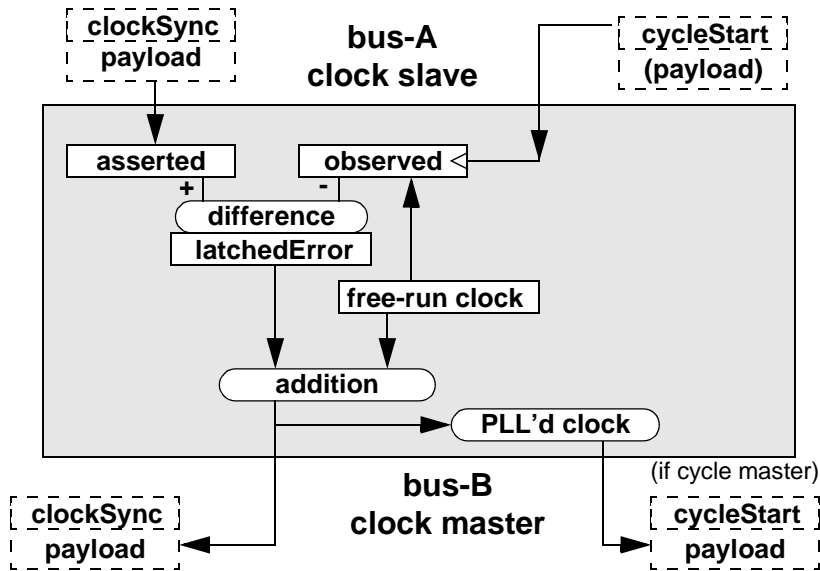


Figure 7—Clock redistribution model

For additional accuracy, the free-running clock value may be rate-adjusted, based on the long-term average of the *latchedError* value. Such rate adjustments shall not the free-run clock frequency to deviate beyond the specifications mandated by the attached bus standards.

1.14 Asynchronous delivery modes

1.14.1 Packet sizes

Bridges shall support a minimum 512-byte transaction payload (which corresponds to an s100 transfer rate) even when faster s200 or s400 transfer rates are supported, and may optionally support larger payloads. Thus, nodeA may be able to perform 1024-byte-payload transfers with nodeB, when only 512-byte-payload transfers with nodeC are possible, as illustrated in figure 8.

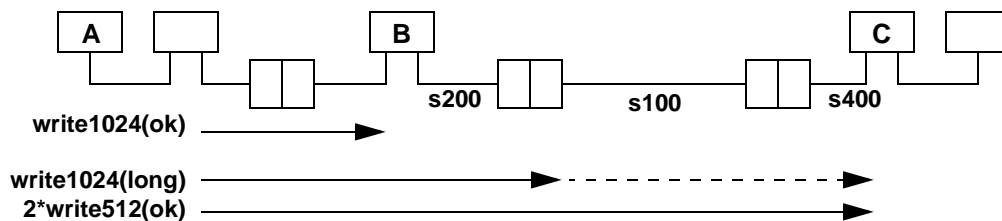


Figure 8—Remote payload sizes

Simple nodes are expected to limit their transaction payloads to 512 bytes, to ensure their delivery. A request with an overly large *data_length* (such that the request is too big or the response would be too big) is rejected by returning an error-reporting response. A response which is too big, based on the observed *data_length* field, is simply discarded.

Sophisticated nodes may attempt to send larger payloads, dropping back to smaller payload sizes if packet-length errors are indicated in the returned response.

1.14.2 Packet transfer speeds

Bridge portals are required to maintain speed-map tables, to identify the speed of local destination nodes S_d as well as the speed of the slowest path to each local-bus portal S_p . Bridge aware nodes can safely send remotely addressed packets at speeds of up to S_p , since the alpha portal is responsible for accepting these packets. Locally addressed packets are normally transferred at speed S_d , the maximum speed allowed between the translating portal and the packet's destination.

Requesters have no need to account for the remote-bus transfer speeds, since the sourcing portal is responsible for final-hop speed conversions. Also, the transfer speeds of a packet sent from nodeA-to-nodeC may change on each of the intermediate buses, as illustrated in figure 9.

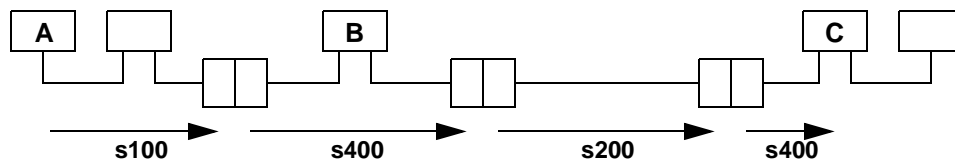


Figure 9—Remote transfer speeds

In figure 9, the packet is originally transferred at s100, then s400, then s200, and finally s400. Thus, a packet may be delivered to the destination with a higher transfer rate than supported by the source.

By default, intermediate portals transmit asynchronous packets at the speed of the slowest portal-to-portal link. In some cases, this may be less than the speed of the connected portals, because the packet may pass through one or more lower speed phys (this is called a speed trap). Terminal portals transmit asynchronous packets at a speed determined by the speed map of the bus.

1.15 Transaction timeout sequences

The requester is expected to maintain active-transaction state between the request-generation and response-returned times. Maintenance of active-transaction state allows returning (possibly out-of-order) responses to be correctly associated with their transaction context and allows transactions to be aborted after transmission failures.

To avoid returning stale packets after the transaction has been aborted, the lifetime of request and response packets (within bridges and responder nodes) is limited. For example, a request packet is rejected if it remains in a bridge for longer than the allowed T_{breq} delay. The bridge rejection response transports error indications ($rcode=resp_data_error$ and $scode=ext_delay_error$), so the requester can understand the cause of the failure, as illustrated by figure 10.

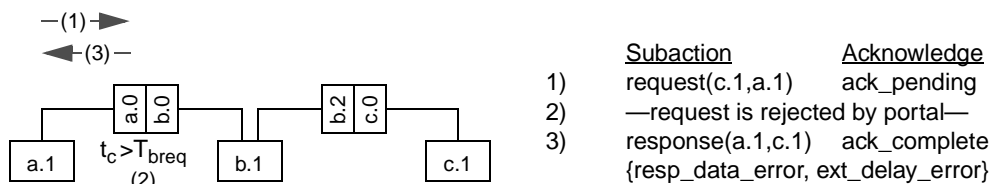


Figure 10—Requesting bridge rejection

In a similar fashion, a request (or response) packet is discarded if it remains in a responder for longer than the allowed T_{res} delay. The requester normally detects such errors by a timeout, as illustrated in figure 11.

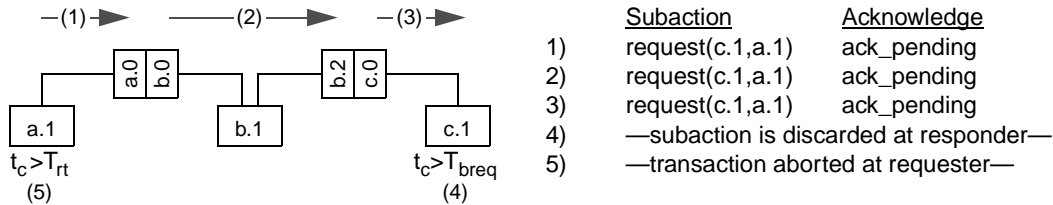


Figure 11—Responder node discard

Finally, a response packet is discarded if it remains in a bridge for longer than the allowed T_{bres} delay. The portal is responsible for logging an error, so the requester can better diagnose the cause of the failure. The requester normally detects such error by a timeout, as illustrated in figure 12.

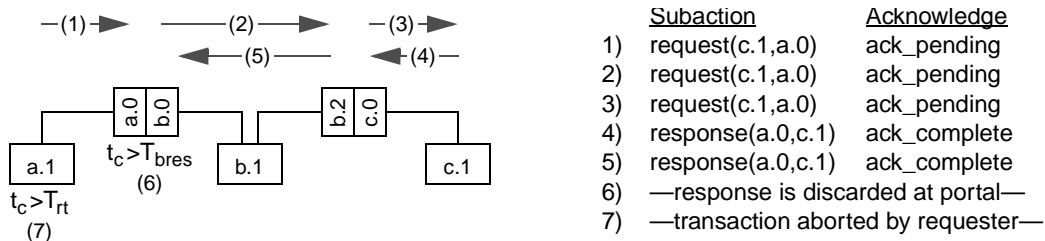


Figure 12—Responding portal discard

To avoid returning stale (and possibly misinterpreted) responses after the transaction has been aborted, the requester's T_{rt} timeout is normally larger than the sum of the residency times.

1.16 Topology changes

1.16.1 Refresh and restart operations

Topology changes can change busID assignments, as new or reset buses receive previously unused addresses and old bus addresses are marked dirty. A cable-topology change starts with a bus reset on the affected bus, which forces that busID address to be reassigned. When the bus reset completes, portal-to-portal communications establish new busID-address assignments, in one of the following ways:

- Net refresh. A net refresh reassigns busID addresses to buses on the affected net. Previously assigned (but now unused) busID addresses are marked DIRTY.
- Net restart. A net restart assigns busID addresses to buses on the affected net. Existing EUI-to-nodeID address translations are purged. Previously assigned and DIRTY busID addresses are eventually marked FREE.
- Net reset. A net reset assigns busID addresses to buses on the affected net. Existing EUI-to-nodeID address translations are purged. Asynchronous queues in bridges are purged. Previously assigned and DIRTY busID addresses are immediately marked FREE.

The nature of the refresh/restart/reset operation depends on the extent of the topology change and the survivor/victim classification of the nodes. The nonreset busID addresses of the survivors remain unchanged and the busID addresses of the victims (when necessary) are changed to avoid address-assignment conflicts. These address reassignment conventions are further detailed in the following subclauses.

The survivor portals are identified by the continued presence of identical prime-alpha and alpha portal identifiers; other portals are victim portals. Nodes managed by survivor portals are called survivor nodes; other nodes are called victim nodes. Survivor portal addresses are more stable, because they have precedence in the net-refresh and net-restart virtual address selection operations.

The prime-alpha and alpha portal identifiers are generated as a side effect of the net refresh and net restart operations (see clause 7). Each bus has one alpha portal and each net (collection of bridge-connected buses) has only one prime-alpha portal. Survivor and victim classifications are based on EUI-64 identifiers, although supplemented EUI-64s are used to select between prime portal candidates.

1.16.2 Change notifications

NOTE—These notifications may be eliminated if their functionality is unnecessary or can be obtained in other ways. An option would be to pass messages through the portals, so they reach the alpha portal on each bus. The alpha portal could then use a local broadcast to disseminate the message.

A net reset is responsible for forcing the invalidation of EUI-to-nodeID translations that may be cached in victim nodes. Rather than relying on an unconfirmed broadcast invalidation signal, portals set quarantine bits to inhibit the use of possibly-stale EUI-to-nodeID translations.

A courtesy broadcast signal informs affiliated nodes of the change-of-topology condition. Since this is only a courtesy notification (no packets are corrupted if this is ignored), the system is not compromised by the less-robust nature of the unconfirmed broadcast indications. Several forms of courtesy notifications are provided, as follows:

- *net_added*. One or more busIDs have been added (when only one is added, that bus is identified).
- *iso_change*. One or more IRMs have been modified by their talker's.

1.16.3 Routing table properties

Each portal is assumed to have access to a 1024-entry routing table. Each routing table entry consists of two bits, which represent one the following states:

- FREE. Forwarding of this busID through portals is disabled.
The busID is not contained in any node's EUI-to-nodeID translation.
- DIRTY. Forwarding of this busID through portals is disabled.
The busID may remain in one or more EUI-to-nodeID translations.
- USED. The busID is not forwarded through this portal, but is forwarded through some others.
- FORW. The busID is forwarded through this portal (and possibly others in the busID-routing path).

The net refresh and restart operations assign busIDs and leave bus-portal-resident routing tables initialized in a consistent fashion, as illustrated in figure 13. In this and following illustrations, the two-digit labels represent busID and phyID portions of virtual addresses; the more relevant portal-resident routing tables are shown below their portal locations.

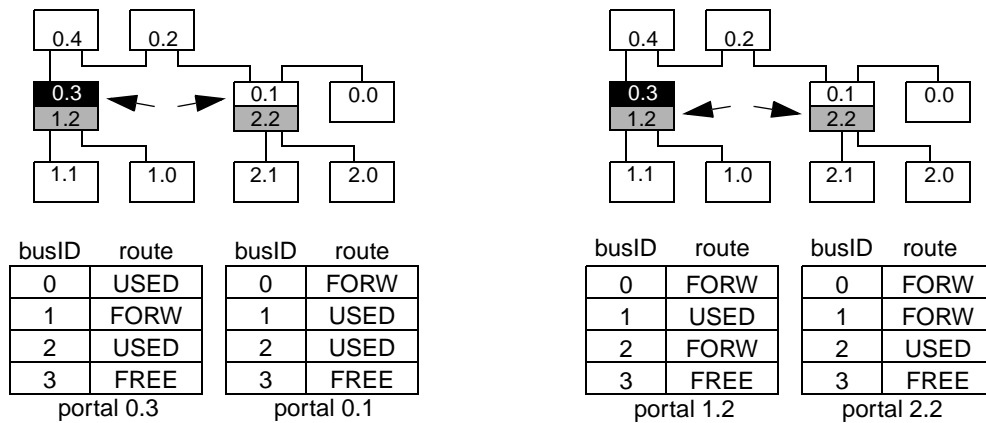


Figure 13—Routing table values (legend in 2.4.1)

The legend for figure 13 is used throughout this document and is described in 2.4. For the convenience of the reader, the legend notation is summarized here. The isolated rectangles and rectangle pairs represent nodes and bus-bridge portals respectively. In portals and nodes, the “2.1” label, ‘2’ and ‘1’ represent the *busID* and *localID* portions of the a virtual nodeID respectively. Black, grey, and white shadings represent prime-alpha, alpha, and non-alpha portals respectively.

1.16.4 Aging of busID assignments

When portal information is merged during the net reset or net refresh operations, the bus-route states are merged on a per-busID basis, using the merging specification algorithm described in table 1.

Table 1—Validity merging algorithm

	Matching StateB			Differing StateB		
StateA	VALID	DIRTY	FREE	VALID	DIRTY	FREE
VALID	VALID	VALID	VALID	<i>DIRTIED</i>	<i>DIRTIED</i>	VALID
DIRTY	VALID	DIRTY	DIRTY	<i>DIRTIED</i>	<i>DIRTIED</i>	DIRTY
FREE	VALID	DIRTY	FREE	VALID	DIRTY	FREE

The left or right validity matrix is used if the contexts match or do not match respectively. For the entries marked *DIRTIED*, the resulting state is DIRTY and (due to potential use conflicts) this value cannot be used.

1.16.5 Conflicting busIDs

A bus reset can leaves responding nodes in an undefined, or defined differently, state that cannot be safely accessed by an uninformed requester. When the requester is on the same bus as the responder, the bus reset is simultaneously observed by both, so the requester is always informed of the reset condition.

When requester and responder are on separate buses, the requester cannot be easily and reliably informed of the bus-reset condition. To prevent unsafe behaviors of the uninformed requester, a new busID is assigned to all nodes after each bus reset, whether nodes are disconnected, connected, or unchanged. This inhibits others access of potentially inconsistent state (CSR locations and function may change during a bus reset). Assignment of new busID addresses also eliminates the possibility of unintentionally reassigning any node to the virtual addresses of another.

A bus reset forces local nodes' busID assignments to their initial $3FF_{16}$ value, as specified by the Serial Bus standard. The alpha portal is responsible for assigning nodes their new busID addresses after each bus reset, so that nodes can respond to virtual as well as physical addresses.

Address assignments involve directed writes: the nonportals' *NODE_IDS* registers are written first and the portals' *NODE_IDS* registers are written last. Portals inhibit transmission of virtually addressed packets until their *NODE_IDS.busID* value changes from its initial $3FF_{16}$ value, so that packets are never sent to not-yet-initialized virtual addresses.

1.16.6 Bus topology changes

The bus is reset when a node is disconnected from the bus. The survivor subbus is identified by persistent prime-alpha and alpha portal identifiers; the other subbus is a victim. After the bus reset, the survivor and victim nodes perform bus refresh and bus restarts respectively. In all cases, the bus reset causes a new bus number to be assigned.

1.16.6.1 Disconnected node

In the survivor subbus portals, the reset bus is assigned a new *busID* (in this example 0 changes to 3) is marked DIRTY and no quarantines are set, as illustrated in figure 14. In this illustration, the tables represent validity maps for locally assigned stableID addresses.

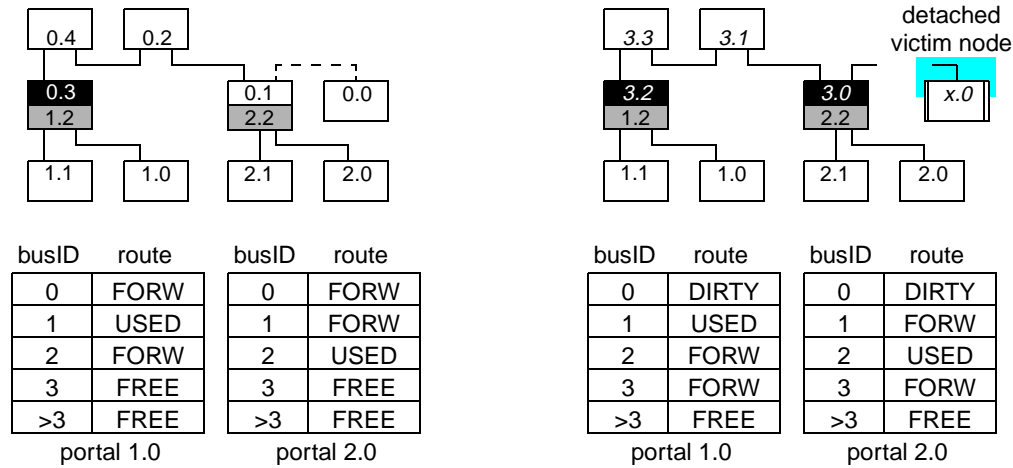


Figure 14—Disconnected node

The victim node is responsible for detecting the absence of portals, which causes it to discard its EUI-to-nodeID translations.

The net refresh activities initially involve the surviving portals, and they can easily confirm the unchanged nature of the net topology. With this knowledge, the local portals can re-enable packet transmissions while the net-refresh related packets propagate through the remaining portions of the interconnect. Due to its quick recovery, the reassignment of a nonconflicting busID is called a *soft net refresh* operation.

1.16.6.2 Reconnected node

When a node is reconnected to a bus, that bus is reset and assigned a new busID address, as illustrated in figure 15. As with the node disconnection example, a mild net refresh allows other portals to communicate with each other while a new nonconflicting busID is being assigned.

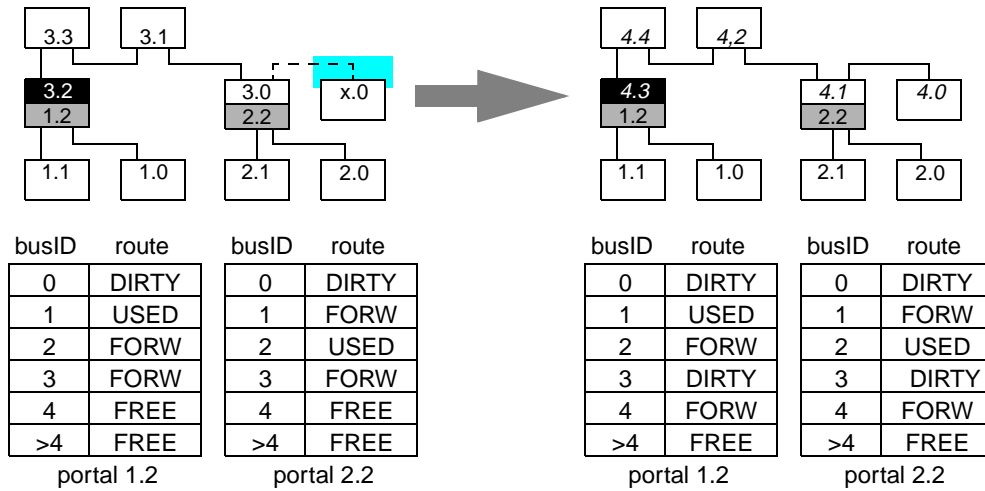


Figure 15—Reconnected node

1.16.6.3 Net topology changes

1.16.6.3.1 Subnet disconnection

When a subnet disappears from a net, net refresh and restart operations are performed on the survivor and victim subnets respectively. Survivor portals observe unchanged prime-portal and alpha-portal identifiers; other portals behave as victim portals. The victim-subnet portals are responsible for quarantining their bus-local nodes, forcing them to discard their EUI-to-nodeID translations, as illustrated in figure 16.

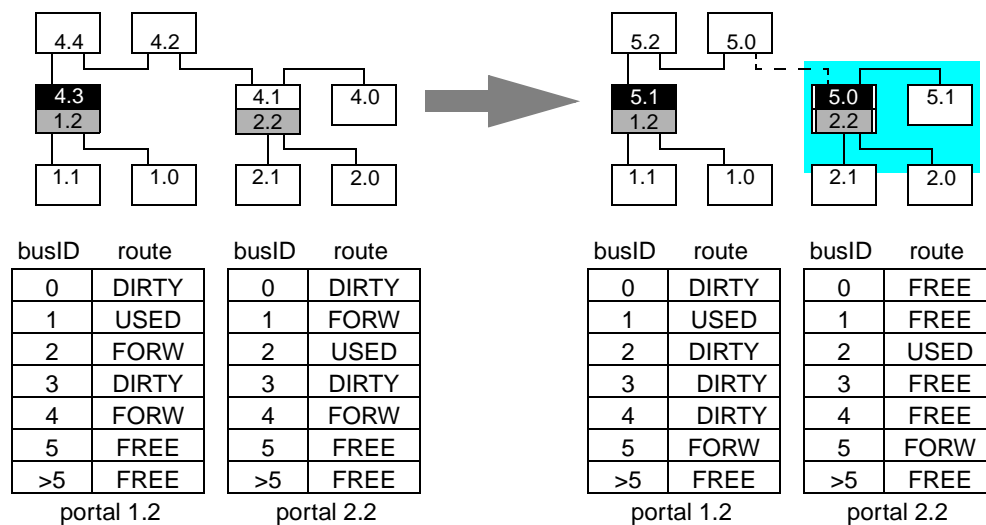


Figure 16—Subnet disconnection

1.16.6.4 Subnet reconnection

When a subnet is reconnected, the unaffected survivor subnet addresses remain unchanged. Previously disconnected subbus nodes and victim buses are assigned nonconflicting busID addresses, as illustrated in figure 17.

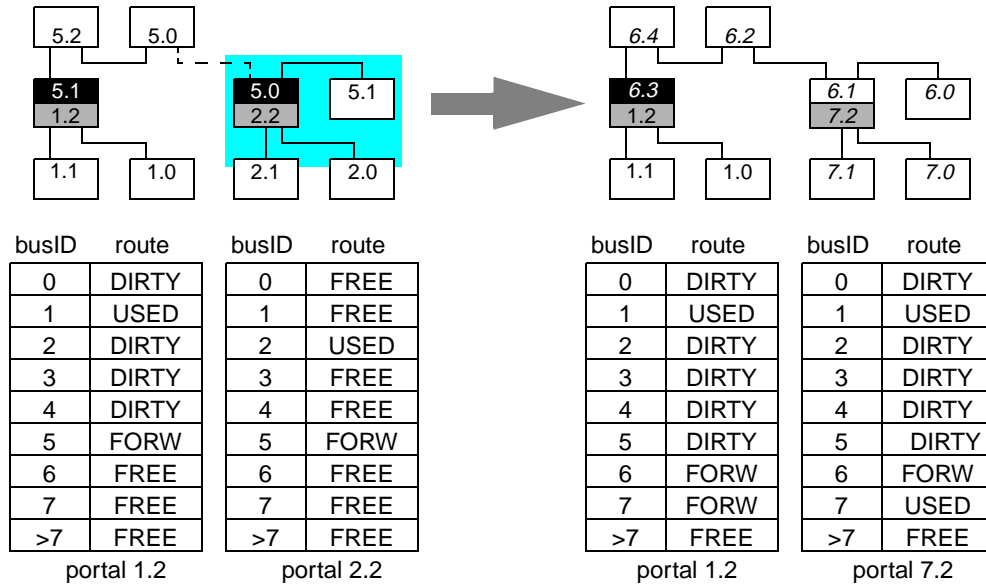


Figure 17—Subnet reconnection

1.16.7 Redundant topology changes

1.16.7.1 Redundant disconnection

A redundant disconnection forces some nodeID address changes, since previously merged subbuses receive distinct busID addresses, as illustrated in figure 18. A survivor subbus observes persistent prime-alpha and alpha portal identifiers; other subbuses are defined to be victims.

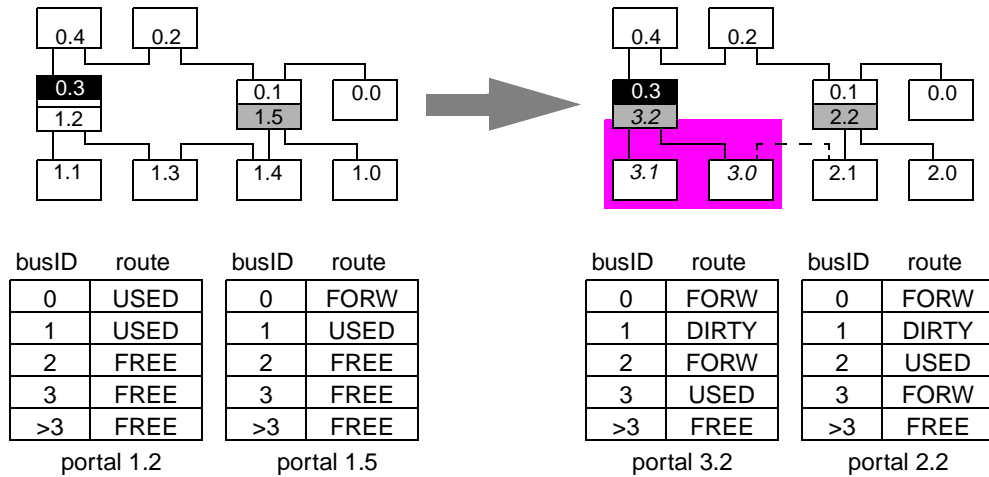


Figure 18—Redundant disconnection

1.16.7.2 Redundant reconnection

A redundant reconnection forces some virtual nodeID changes, since distinct busID addresses are merged, as illustrated in figure 19. A survivor subbus is identified by a persistent prime-portal and alpha-portal identifiers; other subbuses are defined to be victims. The victim subbus inherits the busID of the survivor; victim nodes are assigned new (previously FREE) virtualID addresses.

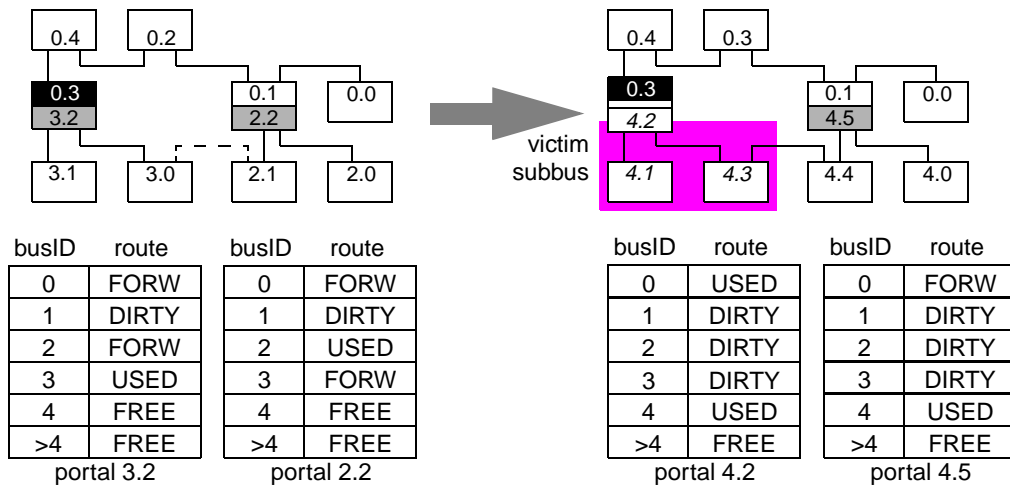


Figure 19—Redundant reconnection

1.17 Bus initialization sequences

1.17.1 Bus reset information

The net refresh protocols assume that topological information is available at the completion of each bus reset. This information is used to detect changes in the attached nodes, and also identifies its portal's "neighbor", allowing messages can be sent from one neighbor to the next. After a bus reset completes, each bridge portal is aware of the following information:

- 1) Phy lists. A cable-topology dependent ordered list of local-bus phys is known. Nodes are classified into categories: portal, legacy, and bridge-aware.
- 2) Routes. Each route state is specified for each bus: FORW, THRU, DIRT, or FREE.
- 3) Context. A context (portalIDs) is associated with the route states.
- 4) Bus address. A previously used busID address is known.

1.17.2 Next-neighbor ordering

NOTE—The following protocols are physical layer dependent and may be different on other Surreal interconnects.

The net refresh protocols are designed to yield a predictable logical topology for a give physical cable topology, regardless of which nodes are selected to be root. The bus refresh protocols involve communications between portals and their neighbors. To ensure stability, the "neighbor" portal is defined by the topology, independent of phyID assignments, as described in this subclause.

Neighbors are defined by relative ringID values, where each node derives ringID values from the observed selfID packets. The assignment of ringID values is based on a conceptual routing of signals, illustrated in figure 20. In figure 20, the labels 'a', 'b', 'c' identify phy ports '0', '1', and '2' respectively, and these ports have an implied internal ordering, listed below:

```
port[a].in=>port[b].out  
port[b].in=>port[c].out  
port[c].in=>counter=>port[a].out
```

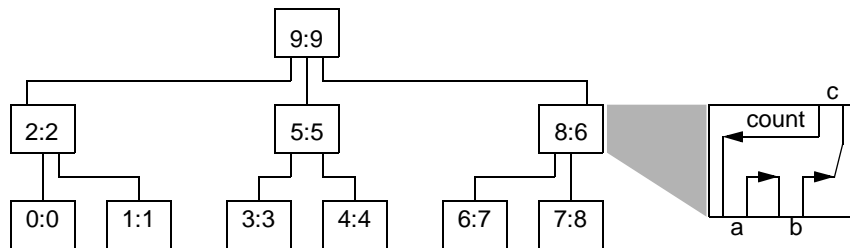


Figure 20—Persistent ringID ordering

Each node assigns conceptual ringID values to the others, starting with ringID=0 on its own port[a] output. The ringID values are assigned by tracing a path through other phys, incrementing the ringID when passing through the port[a] output. This assignment strategy always yields the same next-neighbor selections, despite changes in the selected-root assignment.

1.17.3 Bus refresh messages

The communication protocols for bus refresh involve the sending of messages from each portal to its neighbor, in a daisy chained fashion. These responseless-write messages are idempotent, so missing-ack failures can be simply and safely retried.

A bus refresh starts with a bus reset. After the bus reset completes, each portal sends messages to the next portal, allowing messages to flow in a circular direction, as illustrated in figure 21. In this context, “next” means the portal with the next larger ringID assignment.

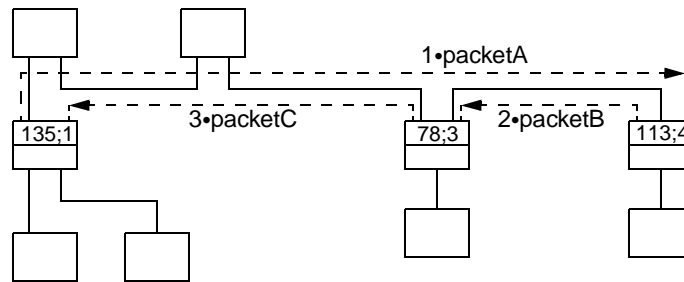


Figure 21—Portal-to-portal message paths

This form of sequential/circular writes is the basis for reliable net refresh communications. Writes that incur errors or busy indications are simply (and immediately) retried until successful. Confirmations are provided by allowing these writes to circulate through other portals until they return to their origin.

The constant sending of responseless writes ensures their successful completion, without mandating special fault-retry protocols. The circular nature of the communication allows the originator of these writes to verify their completion. Such communications are sufficient for reliable broadcasts, but are more flexible because write payloads can be modified as they pass through connected portals.

1.18 Net refresh/reset messages

The protocols used for net refresh were designed with the following assumptions in mind:

- 1) Precedence. A 2-bit precedence allows software to specify prime-portal preferences. This value is prepended to the portal's EUI to generate a stable refreshID identifier. The local-bus portal with the largest refreshID identifier is selected to become the net's prime portal.
- 2) Messages. Messages are distributed to other bus-local portals in the following fashion:
 - a) Ordered. The topology-dependent root-independent portal ordering is available after bus reset. This allows each node to circulate messages by sending them in a next-neighbor ordering.
 - b) Idempotent. Messages are idempotent responseless writes, so failures can be simply retried.
- 3) Selected victims. Normally only a few (victim node) transactions are terminate prematurely.
- 4) Constructive reconnect. A net is minimally disrupted by transient disconnect/reconnect sequences.

1.19 Net refresh

1.19.1 Bus reset effects

A bus reset occurs when a new node (*Ac*) is attached, as illustrated in figure 22. The bus reset has the effect of invalidating *NODE_IDS.busID* addresses in local portals, which effectively isolates them from remote. The intent of a net refresh is to acquire a new bus number for the reset bus, without affecting the *busID* addresses or routes of other portals.

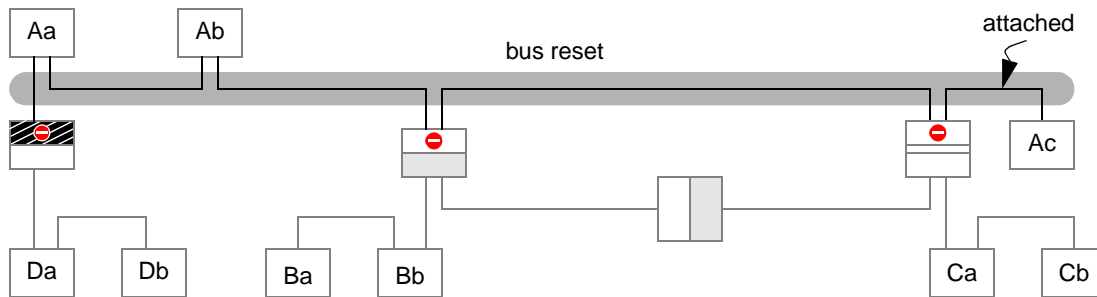


Figure 22—Node attachment effects

To coordinate activities when resets occur on multiple buses, a prime portal (cross-hashed black rectangle) is selected to coordinate the net refresh activities. For stability, the node with the largest refreshID is selected to become the prime portal; the refreshID is a concatenation of a 2-bit preference number and the portal's 64-bit EUI. The prime portal's EUI also serves as the context identifier for bridge routing tables.

1.19.2 Bus acquisition

A net refresh starts with messages sent between local bus bridge portals. The intent is to select one of these portals (the prime-portal candidate) to coordinate the net refresh operation. Each node sends acquisition messages to its neighbor and these messages initially contain *refreshID* (a prime-portal selection identifier). Each candidate monitors incoming refreshID values and changes to a server when a larger *refreshID* is observed. The largest (most preferred) refreshID value eventually circulates to all local portals, as illustrated by figure 23.

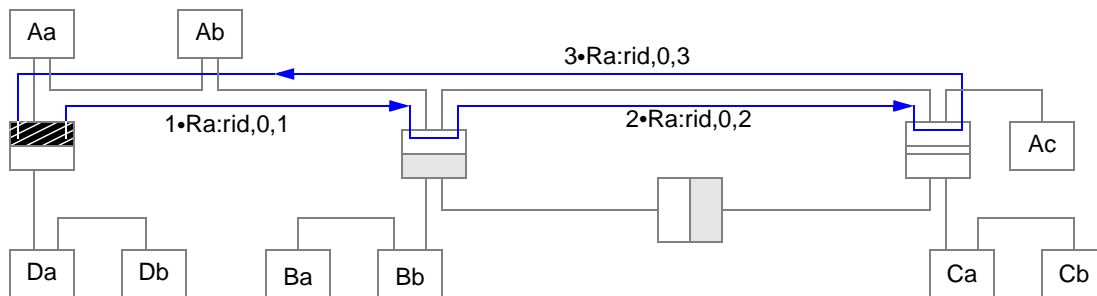


Figure 23—Local bus acquisition

In this illustration, the leading strings identify the message orderings: “1•” messages appeared first, “2•” messages appeared next, and “3•” messages appeared last. The acquisition message supplies a refresh identifier *rid*, as well as *busCount* and *portalCount* fields (0 and 1-to-3 in this example).

For robustness, the *portalCount* value in acquisition messages is incremented when passing through portals. This ensures the eventual demise of rogue resets, by allowing them to be aged until dead. Reset messages are sent periodically, once each arbitration interval, until the net refresh completes. In the absence of continuous messages, portals timeout and attempt to become prime portals.

1.20 Net acquisitions

The net refresh eventually forms a spanning tree by circumscribing the paths through bus bridge portals, as illustrated in figure 24. Each portal communicates with its adjacent neighbor by writing messages into a standardized CSR location. During the final state of a net refresh, the portal-to-portal messages flow in the direction of the solid arrows.

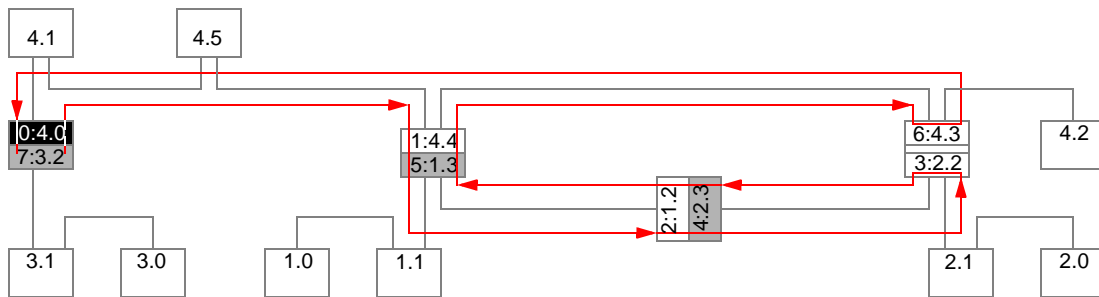


Figure 24—Circumscribed net topology

At the conclusion of the net refresh, each node has a net-unique *nodeID* consisting of *busID* and *localID* components; a node's *localID* equals its *phyID*. In addition, each portal has a distinctive portal identifier, that can be used by others to better navigate through the topology.

2. References and definitions

NOTE—The following lists identify additional material for the current draft and therefore not intended to be complete.

2.1 References

The following standards contain provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below.

ANSI/ISO 9899-1990, Programming Language—C.^{1,2}

ISO/IEC 13213: 1994 [ANSI/IEEE Std 1212, 1994 Edition], Information Technology—Microprocessor systems—Control and Status Register (CSR) Architecture for Microcomputer Buses.²

IEEE Std 1394-1995, Standard for a High Performance Serial Bus³

2.2 Conformance glossary

2.2.1 reserved codes. A set of values for a defined field that are not used within this standard, but are reserved for future revisions of this standard. A reserved code shall not be generated or, upon development of a future standard, may be used as specified by such a standard. The recipient of a *reserved* code shall check its value and shall reject code values that were not defined at the time of its inception.

2.2.2 reserved fields. A set of bits not defined by this standard, but reserved for future revisions of this standard. A reserved field shall be zeroed or, upon development of a future standard, set to a value specified by such a standard. The recipient of a *reserved* field shall not check its value.

2.3 Technical glossary

The following terms are used in this standard:

2.3.1 alpha portal: A single portal within each bus, that is selected based on its route from the prime-alpha portal.

2.3.2 consuming portal: The bridge portal that eavesdrops and accepts asynchronous request/response packets addressed to itself or remote buses routed through this bridge.

2.3.3 CIP: An acronym for “common isochronous packet”.

2.3.4 controller-path portal: Defined relative to a given node and isochronous connection. If the node and the controller are on separate buses, the first portal on the path between the node and its isochronous controller. Otherwise, the alpha portal on the node’s bus.

2.3.5 common isochronous packet: An isochronous packet that has a standard format defined by 61883.

¹Replaces ANSI X3.159-1989.

²ISO/IEC [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 or from International Electrotechnical Commission, 3 rue de Varembe, Case Postale 131, CH 1211, Genève 20, Switzerland/Suisse.

³IEEE 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

- 2.3.6 controller proxy:** A bridge sequencer that is responsible for maintaining an isochronous connection.
- 2.3.7 displaced victim:** A node whose virtualID was changed (during a net refresh) to avoid duplicate busID assignments for nodes on the same bus.
- 2.3.8 GASP:** An acronym for “global asynchronous stream packet”.
- 2.3.9 global asynchronous stream packet:** A standard asynchronous stream packet format, which supplies the source_ID of the requester and a 48-bit packet format identifier.
- 2.3.10 listener-path portal:** The talker-bus portal on the path between the talker and the listener.
- 2.3.11 listener tag:** A controller-proxy resource that identifies an isochronous listener.
- 2.3.12 producing portal:** The bridge portal that retransmits or rejects packets delivered by the adjacent consuming portal.
- 2.3.13 physical address:** A 16-bit node address, which has $physicalID.busID=3FF_{16}$ and $physicalID.localID=phyID$ components.
- 2.3.14 PLL:** An acronym for phase locked loop.
- 2.3.15 prime portal:** An abbreviation for prime alpha portal.
- 2.3.16 prime alpha portal:** A single portal within each net, selected based on its largest *refreshID* value.
- 2.3.17 Surreal interconnect:** An interconnect abstraction that defines the set of bus services mandated for buses conforming to this bridge architecture specification.
- 2.3.18 talker tag:** A controller-proxy resource that identifies an isochronous talker.
- 2.3.19 talker-path portal:** The listener-bus portal on the path between the listener and the talker.
- 2.3.20 victim:** An abbreviation for a displaced victim or sacrificed victim node.
- 2.3.21 victim node:** A node managed by a victim portal.
- 2.3.22 victim portal:** A portal on a victim subnet or subbus.
- 2.3.23 victim subbus:** A previously disjoint bus that (when joined) inherits a new alpha-portal location.
- 2.3.24 virtualID:** A 16-bit node identifier with a globally unique busID less than $3FF_{16}$.

2.4 Bridge topology illustrations

2.4.1 Attach/detach illustrations

Illustrations are often used to describe the effects of a net refresh. These illustrations show before and after topologies and identifier assignments, as top-left and top-right figures, as illustrated in figure 25. For clarity, a dotted line in the top-left figure identifies the cable that is being attached or detached. For brevity, this legend is referenced (rather than replicated) in other subclauses.

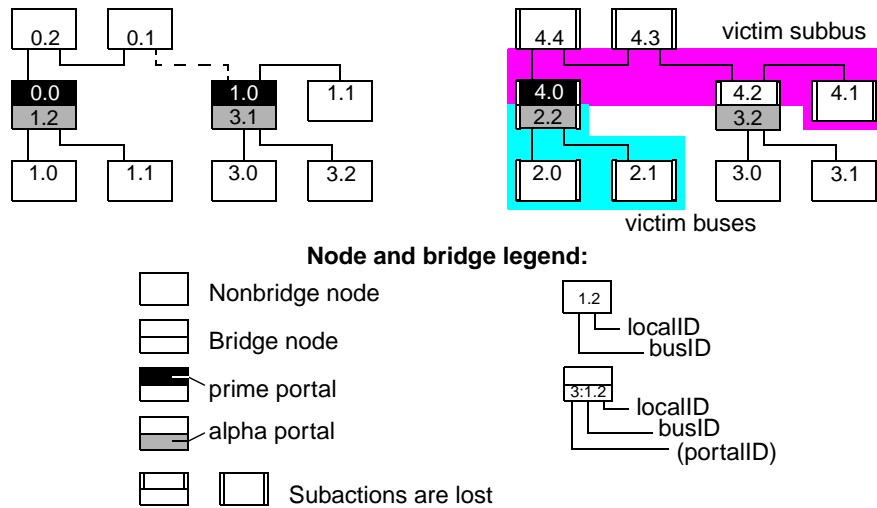


Figure 25—Disruptive attachment (illustration)

A bridge is illustrated as two adjacent rectangles, where each rectangle represents one of its portals. The prime portal (one per net) is shaded black and the alpha portal (one per bus) is shaded grey.

Rectangles on the sides of portals or nodes indicate when subactions may be affected. Black rectangles imply queues are purged, which may effect more than this node’s transactions. White rectangles imply virtual-addresses, whose effects are limited to this node’s concurrently active transactions.

Each portal has a two digit label, such as “1.0”; the “1” is the assigned busID address, and “0” is the assigned virtualID. Each nonportal node has a two digit label, such as “2.1”; the “2” and “1” are the *busID* and *localID* portions of its virtualID respectively.

2.4.2 Isochronous connect/disconnect illustrations

The alpha portal is responsible for allocation of isochronous resources and (when applicable) management of talker-node oPCRs. Portal messages and multiple register accesses are sufficient to form the base connection, as illustrated in figure 26.

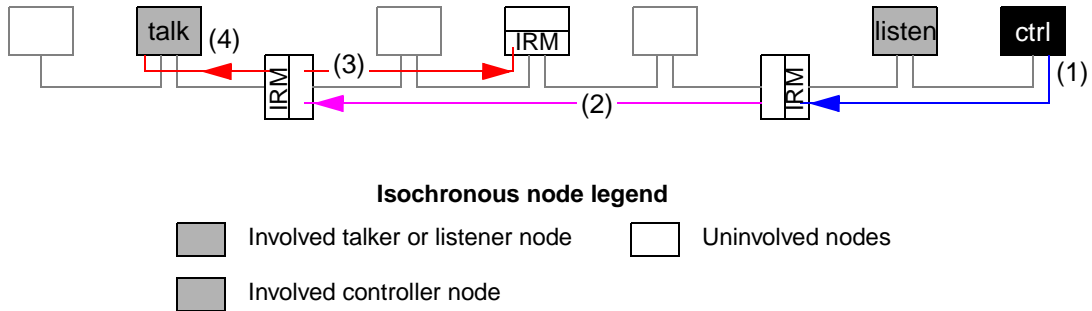


Figure 26—Isochronous connection illustrations

6. Isochronous connection management

6.1 Isochronous overview

6.1.1 Isochronous stream identifiers

An objective of the isochronous connection management is to maintain isochronous communications in the presence of semistable *nodeID* and *channel* number assignments. For this reason, managed connections are identified by the *EUI* and *plugID* of the talker. The combination of these two values is called the *streamID*.

For unmanaged connections, the talker has no *plugID*, so the pseudo-plug identifier is provided by the controller.

6.1.2 Overlaid connections

On talker may be connected to multiple listeners and portions of the route to these buses may be the same. To improve efficiency, only one isochronous channel is used on the shared portion of the connection and a use count is maintained. When the use count is larger than one, this is called an overlaid connection.

When multiple logical connections follow the same hops, these connections are “overlaid” to reduce isochronous resource requirements. An overlaid connection requires an additional up-stream counter, to avoid premature disconnections, but uses no additional isochronous channel or bandwidth.

The overlaid counter identifies the number of portals that are currently listening to the channel, and therefore is always less than 64. In a sense, the overlaid connection can “branch” on each bus and the connection count reflects the number of branches. Since each branch of an overlaid connection may itself have more branches, the total number of overlaid connections can be much larger than 64.

6.2 Isochronous proxies

To maintain the time-sensitive integrity of isochronous connections, bridge portals provide isochronous proxies that act on behalf of the connection controller. For each isochronous connection, controller proxies are located on intermediate portals between talker and listener buses. Each type of proxy has a distinct set of responsibilities, listed below:

- 1) Listener. The listener’s talker-path portal has an additional listener-tag, for each of the local-bus listeners. The proxy is responsible for listener-tag responsibilities, after bus resets, as follows:
 - a) Verify. Local nodes are checked to verify the continued presence of the listener node.
 - b) Resume. The talking channel number is copied into the listener portal’s iPCR.
- 2) Talker. The talker’s listener-path portal has an additional talker-flag. The proxy is responsible for talker-flag responsibilities, after bus resets, as follows:
 - a) Verify. Local nodes are checked to verify the continued presence of the talker.
 - b) Reacquire. Isochronous resources are reacquired in a timely fashion.
 - c) Resume. The talker’s oPCR is updated with the current isochronous channel.
- 3) General. A controller proxy is located on each of the listener-to-talker portals. This proxy has certain responsibilities after a listener-side bus reset, as follows:
 - a) Reacquire. Isochronous resources are reacquired in a timely fashion.
 - b) Reattach. Listener and and pilot proxies are informed of the current isochronous channel.

The steps associated with connect, disconnect, bus reset, and *bus_changed* activities are illustrated in the following subclauses.

6.3 Isochronous management sequences

6.3.1 Nonoverlaid isochronous connections

6.3.1.1 Nonoverlaid connection

Establishment of an isochronous connections begins with knowledge of the isochronous payload size. This normally involves a read of the talker node's oPCR (not illustrated) or ROM. Connection management does requires this information, but its location and format are beyond the scope of this standard.

The connection formation starts with a message sent from the controller to the listener's talker-path portal, as illustrated in figure 27. Connection steps involve portal-to-portal messages and portal-to-IRM transactions, as listed below. Internal transfers, which never appear on a bus, are identified by a '*' marker.

- 1) Dispatch. The controller directs the message to the listener's talker-path portal.
 *Allocate. The controller proxy and listener tag resources are allocated.
 *Acquire. The controller proxy acquires bus-local isochronous resources.
- 2) Handoff. The message is redirected to the next talker-path portal.
 *Allocate. A controller proxy resource is allocated.
- 3) Acquire. The controller proxy acquires bus-local isochronous resources.
 *Acquire. The talker's listener-path proxy acquires bus-local isochronous resources.
- 4) Attach. The write is directed to a talker's oPCR, attaching the talker.

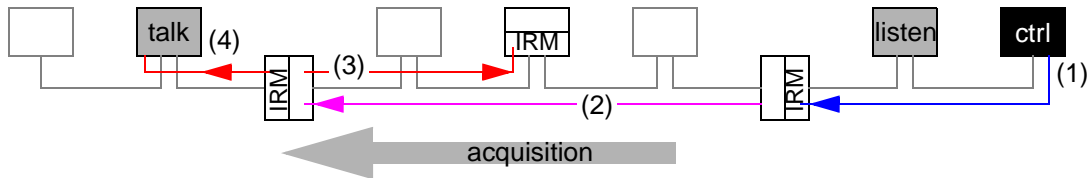


Figure 27—Nonoverlaid connection acquisition

Connection confirmation flows in the opposite direction, as illustrated in figure 28. Confirmation messages commit speculatively allocated resources and communicate channel numbers, as listed below.

- 5) *Redirect. The talker's listener-path portal passes the message to the next listener-path portal.
 The commit message contains the channel number to be used by the listener-path portal.
- 6) Attach. The listener's proxy writes the channel number to the listener's iPCR.
- 7) Reply. The listener's proxy directs a connection-complete reply to the controller.

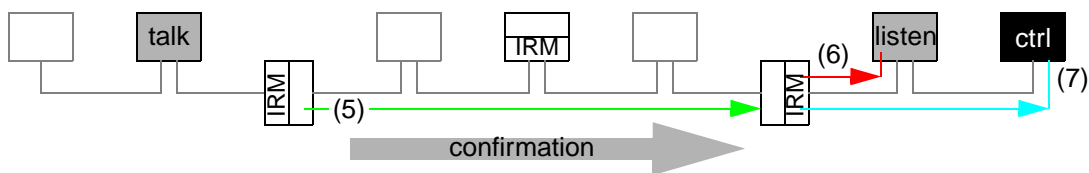


Figure 28—Nonoverlaid connection confirmation

6.3.1.2 Nonoverlaid connection state

The isochronous connection steps establish controller proxies in the listener's talker-path portal, intermediate bridges (not illustrated), and the talker's listener-path portal, as illustrated in figure 29

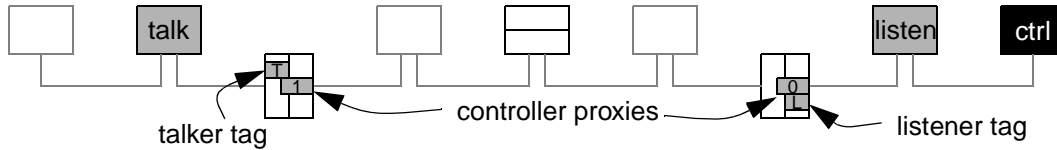


Figure 29—Nonoverlaid connection state

The leftmost controller proxy has a *listener_count* value of 1, indicating one proxy is listening. The rightmost controller proxy has a *listener_count* value of 0, because no proxies are listening. The *listener_count* value does not include listener tags; their presence is inferred by their tag attachment.

6.3.1.3 Nonoverlaid disconnection

Isochronous disconnection also starts with the listener and involves a sequence of portal-to-portal messages and portal-to-IRM transactions, as illustrated in figure 30. Note that the disconnect reply is returned from the talker's listener-path portal, rather than the listener's talker-path portal.

- 1) Dispatch. The controller directs a disconnect message to the listener's talker-path portal.
- 2) Detach. This proxy detaches the listener, by writing its iPCR.
- *Release. The controller proxy releases locally acquired isochronous resources.
- *Release. The controller proxy resources are released.
- 3) Handoff. The message is forwarded to the next talker-path portal.
- 4) Release. The controller proxy releases locally allocated isochronous resources.
- *Release. The controller proxy resource is released.
- 5) Detach. The controller proxy detaches the talker, by writing to its oPCR.
- *Release. The proxy releases previously acquired isochronous resources.
- *Release. The proxy resource is released.
- 6) Reply. The talker's listener-path portal directs a reply message to the controller.

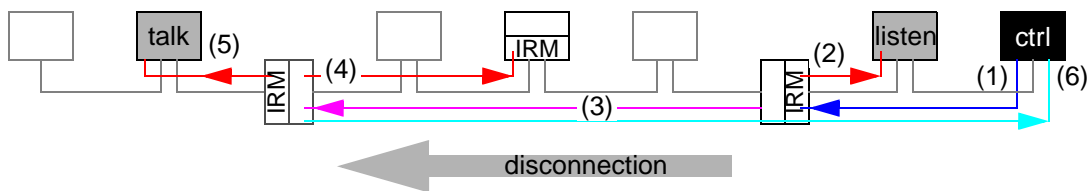


Figure 30—Nonoverlaid disconnection

6.3.2 Overlaid connections

6.3.2.1 Overlaid connection

A fully overlaid connection completes quickly, sharing the previously acquired isochronous resources. For example, a central listener can be overlaid on a previously established connection (figure 29), as illustrated in figure 31 and listed below.

- 1) Dispatch. The controller dispatches a disconnect message to the listener's controller-path portal.
- 2) Redirect. The message is redirected to the listener's talker-path portal.
- *Allocate. A listener tag resource is allocated and affiliated with the pilot proxy.
- 3) Acquire. The listener is attached, by writing the channel number to its iPCR.
- 4) Reply. The proxy generates the connection-complete reply for the controller.

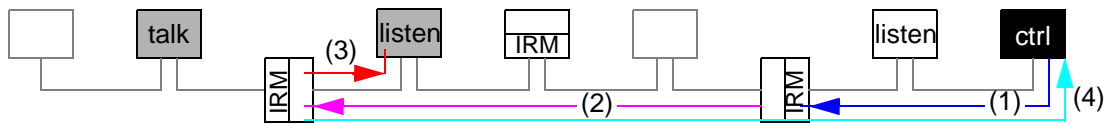


Figure 31—Overlaid connection/disconnection

6.3.2.2 Overlaid connection state

The overlaid connection creates a listener tag on a pre-existing controller proxy, as illustrated in figure 32.

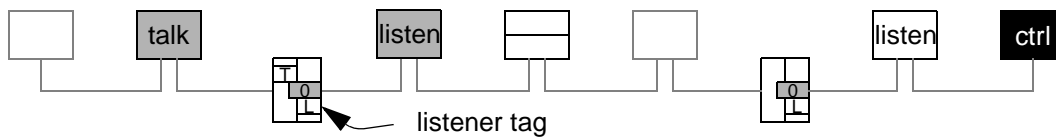


Figure 32—Overlaid connection state

6.3.2.3 Overlaid disconnection

An overlaid isochronous disconnection involve a sequence of portal-to-portal messages and portal-to-IRM transactions, illustrated in figure 31 and listed below:

- 1) Dispatch. The controller dispatches a disconnect message to the listener's controller-path portal.
- 2) Redirect. The message is redirected to the listener's talker-path portal.
- 3) Detach. The listener proxy detaches the listener, by writing to its iPCR.
- *Deallocate. The listener tag resource is released.
- 4) Reply. The controller proxy directs disconnection-complete reply to the controller.

6.3.3 Partial-overlay connections

6.3.3.1 Partial-overlay disconnection

A partial-overlay isochronous disconnection involves a sequence of portal-to-portal messages and portal-to-IRM transactions. For example, an isolated listener can be removed on a previously established connection (figure 32), as illustrated in figure 33 and described in the steps below:

- 1) Dispatch. The controller directs a disconnect message to the listener's talker-path portal.
- 2) Detach. This proxy detaches the listener by writing to its iPCR.
 - *Release. The proxy releases local-bus isochronous resources, by updating the IRM.
 - *Deallocate. The listener-tag and controller-proxy resources are released.
- 3) Handoff. The next talker-path is informed of its downstream proxy detachment.
- 4) Reply. The current portal directs a disconnect-complete reply to the controller.

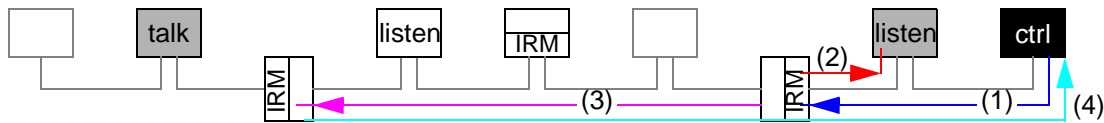


Figure 33—Partial-overlay disconnection/reconnection

6.3.3.2 Partial-overlay disconnection state

The partial-overlay disconnection releases resources associated with an isolated listener, while retaining the state associated with central listeners, as illustrated in figure 34.

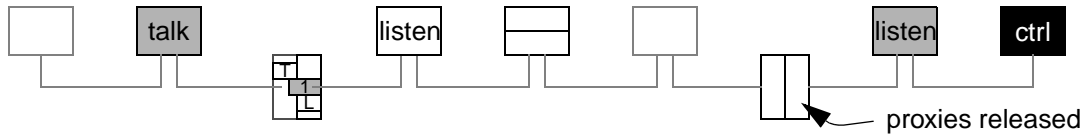


Figure 34—Partial-overlay disconnection state

6.3.3.3 Partial-overlay reconnection

A partially overlaid connection involves resource allocations on the nonoverlaid segments, as illustrated in figure 33 using the steps listed below:

- 1) Dispatch. The controller sends a connect message to the listener's talker-path portal.
 - *Allocate. The controller-proxy and listener-tag resources are allocated.
 - *Acquire. The controller proxy acquires the necessary locally managed isochronous resources.
- 2) Attach. The controller proxy attaches the listener, while writing a channel number to its iPCR.
- 3) Attach. The message is redirected to the next talker-path portal, announcing the proxy's presence.
- 4) Reply. This proxy generates a disconnect-complete reply for the controller.

6.3.4 Unmanaged isochronous connections

6.3.4.1 Unmanaged connection

An unmanaged isochronous connection assumes the controller allocates isochronous resources on the listener and talker buses. The connection formation with a message sent from the controller to the listener-bus controller-path portal, as illustrated in figure 35. Connection steps involve portal-to-portal messages and portal-to-IRM transactions, as listed below.

- 1) Dispatch. The controller directs the message to the listener's talker-path portal.
 *Allocate. A controller proxy resources is allocated.
- 2) Handoff. The message is redirected to the next talker-path portal.
 *Allocate. A controller proxy resource is allocated.
- 3) Acquire. The controller proxy acquires bus-local isochronous resources.
 *Handoff. The message is redirected to the talker's listener-path portal.
 *Allocate. A controller proxy resource is allocated.

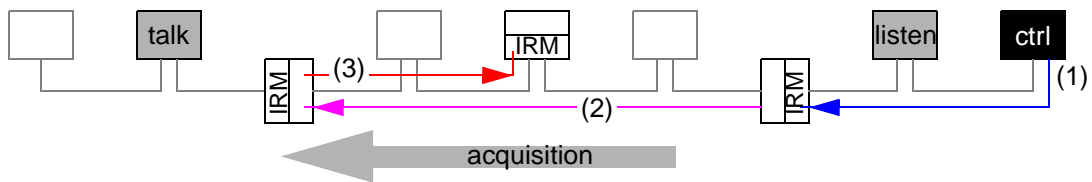


Figure 35—Unmanaged connection acquisition

Connection confirmation flows in the opposite direction, as illustrated in figure 36. Confirmation messages commit speculatively allocated resources and communicate channel numbers, as listed below.

- 4) *Redirect. The talker's listener-path portal passes the message to the next listener-path portal.
 The commit message contains the channel number to be used by the listener-path portal.
- 5) Reply. The listener's proxy directs a connection-complete reply to the controller.

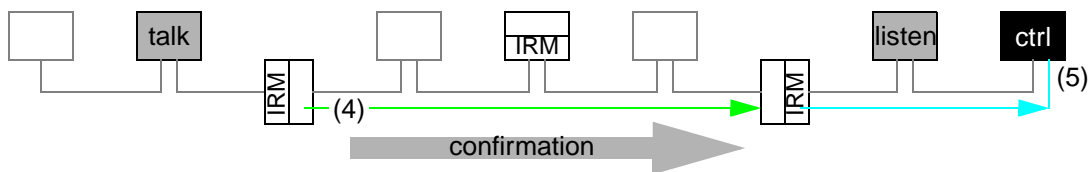


Figure 36—Unmanaged connection confirmation

6.3.4.2 Nonoverlaid connection state

The unmanaged connection steps establish a controller proxy in the listener's talker-path portal, intermediate bridges (not illustrated), and the talker's listener-path portal, as illustrated in figure 29

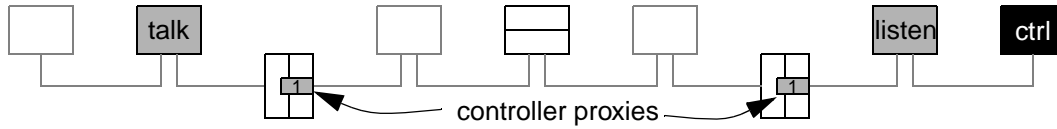


Figure 37—Nonoverlaid connection state

The leftmost pilot proxy has a *listener_count* value of 1, indicating one controller proxy is listening. The rightmost controller proxy has a *listener_count* value of 1, because the controller is managing the listener. The *listener_count* value does not include listener proxies; their presence is inferred by the proxy's listener-tag attachments.

6.3.4.3 Nonoverlaid disconnection

An unmanaged disconnection also starts with the listener and involves a sequence of portal-to-portal messages and portal-to-IRM transactions, as illustrated in figure 30. Note that the disconnect reply is returned from the talker's listener-path portal, rather than the listener's talker-path portal.

- 1) Dispatch. The controller directs a disconnect message to the listener's talker-path portal.
*Release. The controller proxy releases locally acquired isochronous resources.
*Release. The controller proxy resource is released.
- 2) Handoff. The talker-path portal forwards the message to the next talker-path portal.
- 3) Release. The controller proxy releases locally allocated isochronous resources.
*Release. The controller proxy resource is released.
- 4) Release. The proxy releases previously acquired isochronous resources.
*Release. The proxy resource is released.
- 5) Reply. The talker's listener-path portal directs a reply message to the controller.

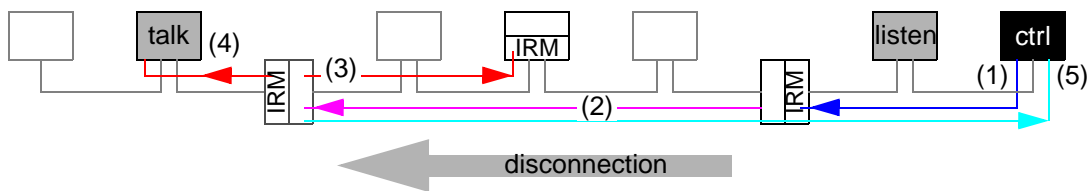


Figure 38—Nonoverlaid disconnection

6.3.5 Bandwidth allocation

6.3.5.1 Isochronous bandwidth parameters

Isochronous connection management uses *maximumBandwidth*, *averageBandwidth*, and *creditLimit* parameters to specify traffic parameters of an isochronous connection. Only the *maximumBandwidth* parameter is used for Serial Bus allocations; the *averageBandwidth*, and *creditLimit* parameters are provided for the benefit of other bandwidth-constrained Surreal interconnects. The rationale for providing additional parameters is twofold:

- 1) Bursty traffic. The type of multimedia traffic expected to pass through bridges may be bursty and the bridge may be able to average the effects of a burst over multiple cycles.
- 2) Limited bandwidth. Some Surreal interconnects may not be able to operate at the full Serial Bus rate, due to physical media or other limitations. Possible examples of limited-bandwidth Surreal interconnects include twisted-pair phone lines as well as wireless RF and IR transmissions.

The *maximumBandwidth* parameter specifies the maximum number of bytes transferred in an isochronous cycle. Specifying the number of bytes (as opposed to quadlets) allows other Surreal interconnects to specify alternate packet-padding schemes.

The *averageBandwidth* parameter specifies the average number of bytes transferred in multiple isochronous cycles. The averaging period is effectively defined by the *bandwidthCredits* parameter, as described in the remainder of this subclause.

The *creditLimit* parameter specifies the extent (in bytes) to which isochronous traffic can exceed the specified average value. Isochronous traffic may have peaks above and valleys below the specified average value, as illustrated in figure 39.

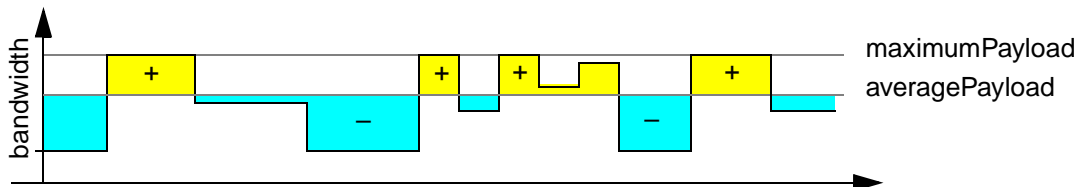


Figure 39—Nonoverlaid connection acquisition

A *runningCredits* value is (in concept) increased or decreased in each cycle, based on the difference between the actual and average bandwidths. The *creditLimit* value specifies a maximum *runningCredits* value that is never exceeded. This is more accurately defined by equation 1, computations that could (in theory) be performed in each isochronous cycle.

$$\begin{aligned} \text{runningCredits} &= \text{Minimum}(0, \text{runningCredits} - (\text{payload} - \text{averagePayload})); \\ \text{assert}(\text{payload} < \text{maximumPayload} \ \&\& \ \text{runningCredits} < \text{creditLimit}); \end{aligned} \quad (1)$$

Not all implementations are not expected to compute a *runningCredits* value, but many are expected to allocate isochronous bandwidth based on knowledge of their averaging buffer size and the traffic pattern's *creditLimit* parameter.

With these parameters, the isochronous bandwidth allocation on some Surreal interconnects may depend on the number of delays introduced by the bridge. For this reason, connection protocols allow the application to

specify acceptable delay values and device the excess delays among the bridges, as described in the following subclauses.

6.3.5.2 Delay sampling

The connection is established by sending messages in the listener-to-talker direction, as illustrated in figure 40. At each step, pilot proxies allocate bandwidth based on the shortest-possible delay, in steps (2) and (5).

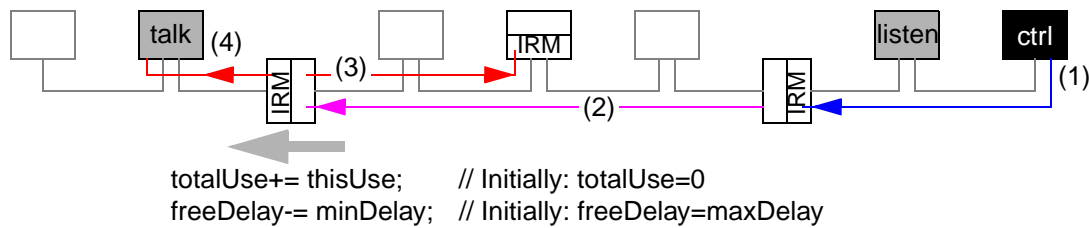


Figure 40—Accumulation of delay preferences

In addition to allocating their local local isochronous resources, bridges increase the message's *totalUse* value by their delay-use. The delay-use value equals the number of internal bandwidth units saved if an additional delay were to be provided. The intent is to allocate residual delays in a weighted fashion, where a bridge's preference is proportional to the benefits derived from additional delay allocations.

When the connection message reaches the talker-proxy, the message has accumulated the *totalUse* values from intermediate bridges. The connection confirmation flows in the talker-to-listener direction, as illustrated in figure 41.

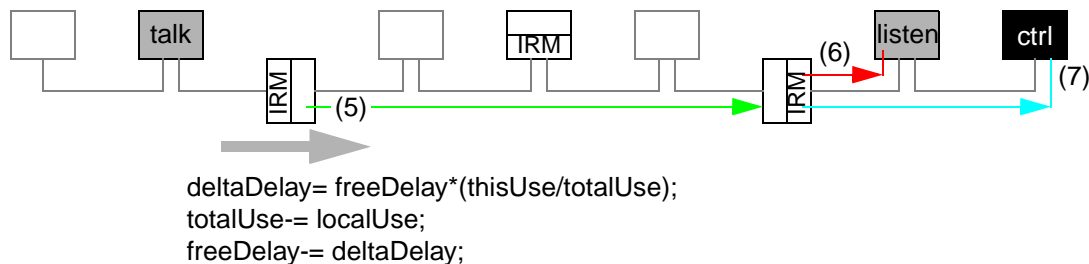


Figure 41—Delay allowance adjustments

Confirmation messages distribute the excess delay allowances, which can enable releases of internal isochronous resources in intermediate bridges (not illustrated). The additional delay allocation, *deltaDelay*, is allocated based on the bridges delay-use parameter. Before the message is forwarded, its *totalUse* and *freeDelay* parameters are adjusted to be reflect this bridge's contribution.

6.4 Bus-reset recovery

One of the reasons for invoking a bus reset is to resolve inconsistent or ambiguous bus-local isochronous resource allocations. The talker-path portal (or alpha portal on the talker's bus) assumes this obligation, allowing resources to be reclaimed in a timely fashion, and (when necessary) informs other pilot/listener proxies of isochronous channel changes.

6.4.1 Talker bus-reset recovery

After a bus reset, the proxies (in listener-path portals) are responsible for timely reallocation of isochronous resources and update of the talker's oPCR. These steps are listed below and illustrated in figure 42.

- 1) *Reclaim. The talker-bus proxies reclaim previously acquired isochronous resources.
 Reattach. The talker proxy directs a write to the talker oPCR, to reattach this connection.
 (the talker's channel number may be changed, depending on the reclamation results (setp 1).

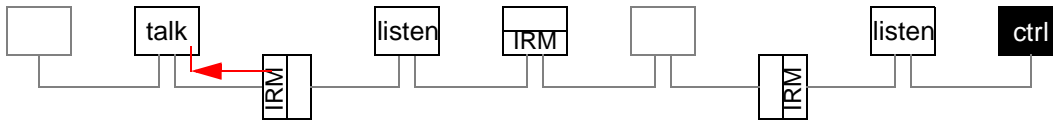


Figure 42—Talker bus-reset recovery

6.4.2 Intermediate bus-reset recovery

Similar sets of bus-reset recovery operations are performed when resets occur on nontalker buses, as illustrated in figure 43.

- 1) Inform. The local-bus proxies inform the talker-path pilot proxy of their presence.
- 2) Reclaim. The talker-path proxy portal reclaims previously acquired isochronous resources.
- 3) Reattach. The talker proxy directs writes to listener's iPCRs, to reattach these connections.
 This reattach write also transports a new channel number, which can be changed by step (2).
- 4) Restore. The talker proxy directs messages to listener-path proxies, to reattach them.
 This reattach message also transports a new channel number, which can be changed by step (2).

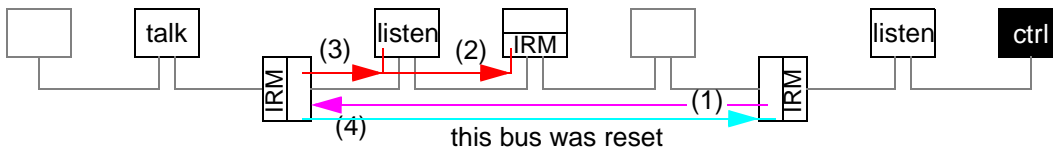


Figure 43—Intermediate bus-reset recovery

6.4.3 Listener bus-reset recovery

Similar sets of bus-reset recovery operations are performed when resets occur on listener buses, as illustrated in figure 44.

- 1) *Reclaim. The talker-path proxy reclaims previously acquired isochronous resources.
 Reattach. The talker-path proxy directs writes to listener's iPCRs, to reattach this connection.
 This reattach write also transports a new channel number, which can be changed by step (2).

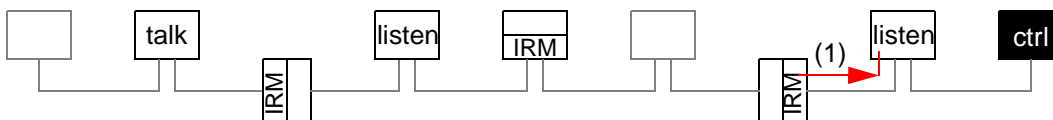


Figure 44—Listener bus-reset recovery (glossary see 2.4.2)

6.5 Net-change recovery

One of the reasons for invoking a net refresh is to resolve inconsistent or ambiguous nonlocal isochronous resource allocations. The listener and talker proxies assume this obligation, allowing resources to be reclaimed (or lost) in a timely fashion.

6.5.1 Talker-bus net-refresh recovery

After receiving a *net_changed* event, controller's are responsible for quickly reclaiming established connections. The reclamation starts with the controller, as illustrated in figure 45 and described below.

- 1) Dispatch. The controller sends a message to the listener's controller-path portal.
*Redirect. The controller-path portal redirects the message to the listener's talker-path portal.
- 2) Handcuff. The listener's proxy reclaims resources and directs the message to its talker-path portal.
*Handcuff. The current proxy reclaims resources and directs the message to the talker's alpha portal.
- 3) Reply. Having validated the connection, a reply message is returned to the controller.

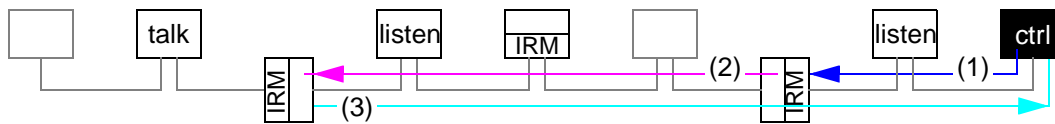


Figure 45—Net-refresh recovery

6.5.2 Bandwidth adjustments

A bandwidth adjustment starts with the talker's adjustment of IRM-resident accounts. This triggers the broadcast of a change indication to attached controllers, as illustrated in figure 4 and described below.

- 1) Change. A lock transaction causes the IRM to increase or decrease its isochronous allocations.
- 2a-c) Broadcast. An *irm_changed* event is broadcast to all nodes, so that the controllers can adjust their connection bandwidths accordingly.
Controllers are expected to initiate a standard reconnection procedure, with revised bandwidth parameters.

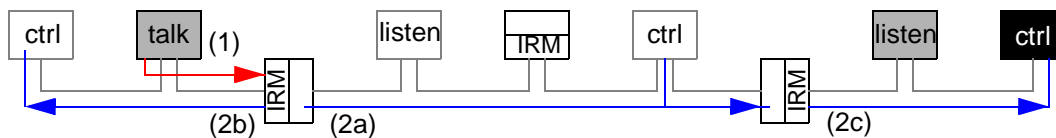


Figure 46—Isochronous-change notification

6.5.2.1 Connection message formats

A managed connection starts with acquisition messages sent to the listener-bus controller-path portals and forwarded towards the talker. These messages have a common format, as illustrated in the left portion of figure 47. The connection confirmation returns additional parameters, as illustrated in the right portion of figure 47.

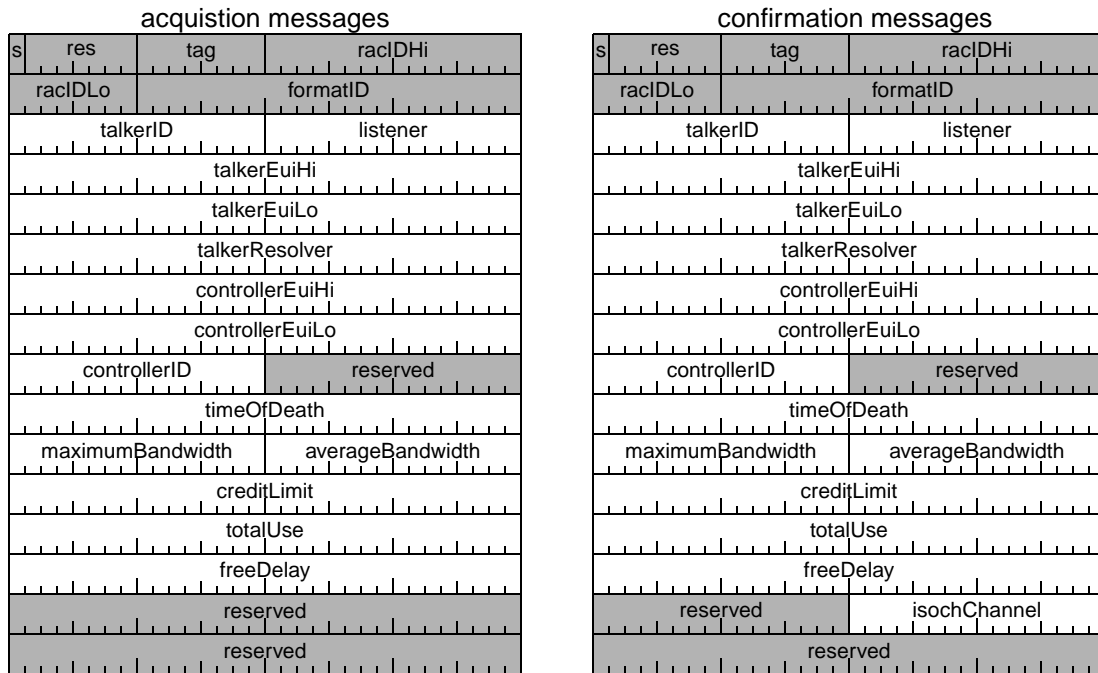


Figure 47—Acquisition and confirmation message formats

6.5.3 Net refresh

TBD—Decide how the connection is broken in the absence of a controller, to recovery from missing controllers or unrecoverable isochronous-connection message errors:

- a) Timeout. Portal-managed isochronous resources are released, if not re-established within T_{nf} after event.
- b) Heartbeat. The isochronous resources are released if not periodically “touched”.

