LCS Protocol Specification Protocol Version 2

Released

Issue 2: April 2001

Revision History

Issue Number	Issue Date	Details Of Change
1	August 2000	Creation of document
2	April 2001	Updated document

1 INTRODUCTION

A key requirement of scalable switching fabrics is to enable the implementation of systems which physically separate the linecards in a switch from the core of the switch. This is done by a LineCard to Switch (LCS) protocol.

This physical separation enables the extensibility and reuse of linecards with upgradable switch cores. Reusable linecards extend the development efforts and installation investment while scalable switch cores ensure continued expansion in capacity. By introducing a scalable and open protocol that separates a router's linecards from its switching core, a number of benefits in switch products are enabled.

1.1 Benefits of the LCS Protocol

The main benefits of this architecture are:

- Provides protection of end-user capital investment by separating investments in linecards from investment in switching cores.
- Enables scalable system aggregate bandwidth from 10's of Gb/s to 10Tb/s and beyond, while maintaining backward compatibility.
- Enables real time upgrades, replacement, and repair of router switching cores in production networks, avoiding network downtime.
- Enables configurations with very high line-speeds and aggregate bandwidth (e.g. hundreds of ports ranging from OC-12 to OC-192 speeds)
- Enables modularity and large port densities — using fiber-optic links to separate linecard racks and switching core across multiple linecard racks, making possible large port densities (e.g. thousands of Fast Ethernet ports or OC-3 ports) and system modularity.
- Enables a redundant switching router core that eliminates network downtime.
- Enables a graceful degradation of switching cores to eliminate network downtime.

1.2 LCS Protocol Goals

The LCS protocol is lightweight, using fixed-sized segments. Forwarding information is contained in each segment. A credit-based flow control mechanism for ingress traffic is also contained in each segment, while it is expected that most of the buffering of the egress traffic is handled by the linecard. The protocol is designed to be flexible enough to support future switches with higher bandwidth, more ports, and/or enhanced quality of service support.

Each LCS segment contains a fixed payload size of either 64 bytes or 76 bytes and eight bytes of prepend. Part of this prepend, the 'Tag Info', is used to identify the segment to the receiving entity. The label, along with some other prepend information, is sufficient to describe to the receiver what action is to be taken on the attached segment.

The use of the credit-based flow control as well as more detailed usage of the labelling mechanism is described on the following pages.

2 LCS DATA LINK LAYER

2.1 LCS Protocol Prepend

LCS segments are 72 or 84 bytes in length. The line to switch format is shown in Table 1, while Table 2 shows the field definitions of this format. Likewise, Tables 3 and 4 show the switch to line format and field definitions, respectively.

Table 1. LCS Format from Linecard to Switch

		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	**Request Field**	Req Valid	Label_1 [13:7]						
Byte 1		Label_1 [6:0]							Type_1 [3]
Byte 2		Type_1 [2:0]			Rsvd				
Byte 3	**Payload Field**	Payload Valid	Seq_Number[9:3]						
Byte 4		Seq_Number[2:0]			Rsvd				
Byte 5	**Hole Field**	Payload_Hole_Request[3:0]				Grant_Hole_Request[3:0]			
Byte 6	**CRC Field**	CRC-16[15:8]							
Byte 7		CRC-16 [7:0]							
Bytes 8-71 or 8-83	**Payload**	Payload Data							

Table 2. Definitions of LCS Fields from Linecard to Switch

Field	Size (bits)	Definition
Req Valid	1	Indicates that request field is valid (Request Label_1).
Label_1	14	Label that is being requested (e.g. unicast-QID/multicast-label/TDM-label).
Type_1	4	Segment type that is being requested (e.g. Control Packet).
Payload Valid	1	Indicates that the Payload Data field contains a valid payload.
Seq_Number	10	A sequence number for synchronizing grants with payloads.
Payload_Hole_ Request	4	Request for a “hole” in the payload stream from switch to linecard. (Priority 3,2,1,0 for bits [3:0], respectively.) (See "2.5 Flow-Control and “Holes”" for explanation).
Grant_Hole_ Request	4	Request for a “hole” in the grant stream from switch to linecard. (Priority 3,2,1,0 for bits [3:0], respectively.) (See "2.5 Flow-Control and “Holes”" for explanation).
CRC-16	16	16-bit CRC calculated over complete 64-bit prepend.[a]
Payload Data	64 or 76 bytes	Payload Data

a. See "2.8 Use of CRC Field" for details of CRC calculation.

Table 3. LCS Format from Switch to Linecard

		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	**Grant Field**	Grnt Valid	Label_1 [13:7]						
Byte 1		Label_1 [6:0]							Type_1 [3]
Byte 2		Type_1 [2:0]			Seq_Num [9:5]				
Byte 3		Seq_Num [4:0]					Payload Valid	Label_2 [13:12]	
Byte 4	**Payload Field**	Label_2 [11:4]							
Byte 5		Label_2 [3:0]				Type_2 [3:0]			
Byte 6	**CRC Field**	CRC-16 [15:8]							
Byte 7		CRC-16 [7:0]							
Bytes 8-71 or 8-83	**Payload**	Payload Data							

Table 4. Definitions of LCS Fields from Switch to Linecard

Field	Size (bits)	Definition
Grnt Valid	1	Indicates that grant field is valid (Grant Label_1 and Sequence Number).
Label_1	14	Label for credit (e.g. QID/mcast-label/TDM-label).
Type_1	4	Segment type for credit (e.g. TDM).
Seq_Num	10	Sequence Number associated with grant.
Payload Valid	1	Indicates whether the egress payload is valid (Payload Label_2 and Payload Data).
Label_2	14	Label for egress payload (e.g. ingress port identifier).
Type_2	4	Segment type for egress payload (e.g. Control Packet).
CRC-16	16	16-bit CRC calculated over complete 64-bit prepend.[a]
Payload Data	64 or 76 bytes	Payload Data

a. See "2.8 Use of CRC Field" for details of CRC calculation.

2.2 Use of Label and Type Fields

This section provides detail about the generic 14-bit labels and 4-bit types defined in Table 5, Table 6, Table 7, and Table 8. In all cases, in multi-bit fields, the MSB is transmitted first. For each segment, the "Type Field" and "Label Field" are encoded as follows:

Table 5. Encoding of 4-bit Type Fields

Type[3:0]	Segment Type	Use
0000	Control Packet	Provides for inband control information between the linecard and the switch core.
0001	Single-Phase Control Packet[a] (Optional)	Provides for inband control information with timing critical requirement.
0010,...,0110	Reserved for future use	Currently not defined for use.
0111	TDM (Optional)	TDM service provides preallocated bandwidth at a regular time interval.
1000,...,1011	Unicast Best-Effort (Priority 0,1,2,3)	Prioritized unicast service where priority 0 is the highest priority.
1100,...,1111	Multicast Best-Effort (Priority 0,1,2,3) (Optional)	Prioritized multicast service where priority 0 is the highest priority.

a. Valid only in the Request Type_1 field; this encoding is reserved in the Grant Type_1 and Payload Type_2 fields.

Table 6. Usage of the Ingress Request Label_1

Segment Type	Type_1	Encoding of Request Label_1[13:0]
Control Packet	0000	Rsvd[a] (13), CP-category (1). CP-category = 0 for LCS Control Packets; CP-category = 1 for CPU Control Packets.
Single-Phase Control Packet	0001	Rsvd[a] (6), SPCP-category (8). SPCP-category = 00000000 for TDM Frame Sel 0; SPCP-category = 00000001 for TDM Frame Sel 1
TDM	0111	MUX (2)[b], Expansion[c] (2), TDM Tag (10)
Unicast Best-Effort (Priority 0,1,2,3)	10xx	MUX (2)[b], Expansion[d] (2), Destination Port (8), Destination subport[e] (2)
Multicast Best-Effort (Priority 0,1,2,3)	11xx	MUX (2)[b], Expansion[f] (2), Multicast Tag (10)

a. Reserved. All bits labeled Rsvd must be set to zero.
b. The MUX[1:0] field is placed at the beginning of the Request Label_1 field in all segments (i.e. both data traffic and control packets). If this field is not used for a MUX function, it can be used as additional bits for expansion. If not used, it must be set to zero.
c. TDM Expansion field used in products if the number of TDM slots is increased. If this field is used, the "TDM Tag" field will increase contiguously. If not used, must be set to zero.
d. Unicast Expansion field used in products if the number of ports is increased. If this field is used, the "Destination Port" field will increase contiguously. If not used, must be set to zero.
e. If the port has no subports (i.e. if it is connected to a single linecard without multiplexing), this field must be set to zero.
f. Multicast Expansion field used in products if the size of the Tag table is increased beyond 1024. If this field is used, the "Multicast Tag" field will increase contiguously. If not used, must be set to zero.

Table 7. Usage of the Egress Grant Label_1

Segment Type	Type_1	Encoding of Grant Label_1[13:0]
Control Packet	0000	Rsvd[a] (13), CP-category (1). CP-category = 0 for LCS Control Packets; CP-category = 1 for CPU Control Packets.
TDM	0111	MUX (2)[b], Expansion[c] (2), TDM Tag (10)
Unicast Best-Effort (Priority 0,1,2,3)	10xx	MUX (2)[b], Expansion[d] (2), Destination Port (8), Destination subport[e] (2)
Multicast Best-Effort (Priority 0,1,2,3)	11xx	MUX (2)[b], Expansion[f] (2), Multicast Tag (10)

a. Reserved. All bits labeled Rsvd must be set to zero.
b. The MUX[1:0] field is placed at the beginning of the Grant Label_1 field in all segments (i.e. both data traffic and control packets). If this field is not used for a MUX function, it can be used as additional bits for expansion. If not used, it must be set to zero.
c. TDM Expansion field used in products if the number of TDM slots is increased. If this field is used, the “TDM Tag” field will increase contiguously. If not used, must be set to zero.
d. Unicast Expansion field used in products if the number of ports is increased. If this field is used, the “Destination Port” field will increase contiguously. If not used, must be set to zero.
e. If the port has no subports (i.e. if it is connected to a single linecard without multiplexing), this field must be set to zero.
f. Multicast Expansion field used in products if the size of the Tag table is increased beyond 1024. If this field is used, the “Multicast Tag” field will increase contiguously. If not used, must be set to zero.

Table 8. Usage of the Egress Payload Label_2

Segment Type	Type_2	Encoding of Payload Label_2[13:0]
Control Packet	0000	Rsvd[a] (13), CP-category (1). CP-category = 0 for LCS Control Packets; CP-category = 1 for CPU Control Packets.
TDM	0111	Expansion[b] (4), Source Port (8), Source subport[c] (2)
Unicast Best-Effort (Priority 0,1,2,3)	10xx	Expansion[d] (4), Source Port (8), Source subport[c] (2)
Multicast Best-Effort (Priority 0,1,2,3)	11xx	Expansion[e] (4), Source Port (8), Source subport[c] (2)

a. Reserved. All bits labeled Rsvd must be set to zero.
b. TDM Expansion field used in products if the number of ports is increased. If this field is used, the “Source Port” field will increase contiguously. If not used, must be set to zero.
c. If the source port has no subports (i.e. if it is connected to a single linecard without multiplexing), this field will be zero.
d. Unicast Expansion field used in products if the number of ports is increased. If this field is used, the “Source Port” field will increase contiguously. If not used, must be set to zero.
e. Multicast Expansion field used in products if the number of ports is increased. If this field is used, the “Source Port” field will increase contiguously. If not used, must be set to zero.

2.2.1 Subport Mode Label and MUX Fields

Label fields are arranged such that port addresses can be subdivided into four multiplexed, subport addresses. Each segment type with a subport field utilizes bits [1:0] for the subport addresses. When subports are not utilized, these bits will be set to zero.

Multiplexing can be done either spatially or time division. Spatial multiplexing uses independent physical connections between a switch port and each of the subport linecards. In this mode, there is no requirement for a MUX field.

Time division multiplexing shares a common physical connection to carry segments to/from each subported linecard to a switch port. The time multiplexing between subports may be done on a quad linecard, or may be done at an intermediate stage. When time-multiplexed, segment transmissions are sequenced between each of the four subports through the use of a MUX field.

When a port of an LCS device uses the time-multiplexed subport mode, two bits in the Request Label_1 field of ingress and Grant Label_1 field of egress segments are used to multiplex/demultiplex each of four subport channels. The 2-bit field is called the MUX[1:0] field. For ingress segments, the MUX[1:0] bits are carried in the Request Label_1 field; for egress segments, the MUX[1:0] bits are carried in the Grant Label_1 (but not the Payload Label_2). Note that for ingress traffic, this field is used to designate the source subport of the ingress segment and is independent of the Request Label_1 subport field. Likewise for egress traffic, the MUX field is used to designate the destination subport of the egress segment and is independent of the Grant Label_1 and Payload Label_2 subport fields.

Segments entering the switch core must have the MUX[1:0] field correctly set. The MUX[1:0] field in arriving segments must follow the sequence 00,01,10,11,00,... in consecutive segments. The MUX[1:0] field in departing segments must also follow the sequence 00,01,10,11,00,... in consecutive segments. When the MUX subport sequence must be interrupted by the need to send an Idle frame, the sequence should continue with the next subport in sequence following the previously sent subport. The MUX[1:0] field is placed at the beginning of the Request Label_1 or Grant Label_1 field in all segments (i.e. both data traffic and control packets).

Segments departing or arriving at the subported line card may have the MUX[1:0] field set to correspond to the linecard’s designated subport. Optionally, the linecard may set the MUX[1:0] field to zero, relying on an intermediary multiplexer device to correctly set the MUX[1:0] field. In such a configuration, the LCS prepend CRC is generated and checked masking the MUX[1:0] field to zero, enabling the intermediary multiplexer device to independently set the MUX[1:0] field without termination and regeneration of the CRC field. Note that without the requirement of setting the MUX[1:0] field, the linecard’s framing function does not have to be subport aware in a system with an intermediary, time-multiplexed data stream.

2.3 How the Flow Works

A sequence of three phases are required to transfer a segment from the ingress linecard to the switch core; these are (1) The Request Phase, (2) The Grant Phase, and (3) The Transmit Phase.

2.3.1 The Request Phase

When the ingress linecard has a segment to send, it sends a request to the switch core. For now, the linecard does not send the segment payload, just the request. The *Request Label_1* field is part of the request to identify the flow to which the requested segment belongs. As we will see, the accompanying 64-byte (or 76-byte) Payload Data field does not correspond to the request field; it contains a payload that has been previously and separately granted to by the switch core.

2.3.2 The Grant Phase

When the switch core receives the request, it stores the request until it is ready to receive the segment payload from the linecard. The details of how and when the request is granted are specific to the implementation of the switch core and are not discussed further here. Assuming opportunities of the same priority of request continue to become available, the switch core will eventually be ready to receive the segment payload and will issue a grant to the linecard. The minimum time for this to occur is shown in Table 9. The grant signals to the linecard to send the segment payload. Because the segment payload still resides on the ingress linecard, the switch core must issue a grant to the linecard asking it to send the segment payload. The grant field sent from the switch core to the ingress linecard contains the same *Grant Label_1* field as the original *Request Label_1* field.

2.3.3 The Transmit Phase

When the ingress linecard receives the grant, it transmits the segment payload to the switch core. The order in which the grants are received is the order in which the segment payloads must be transmitted. Grants must be responded to within the minimum and maximum time periods shown in Table 9. The segment payload has now been transmitted from the ingress linecard to the switch core and the transaction is complete.

2.3.4 Flow Summary

More precisely, the ingress linecard can only send a request to the switch core when the switch core has room to accept it. Hence, the linecard must keep track of the number of outstanding requests and not exceed the request capacity of the switch core. This is implemented using credit-based flow-control between the linecard and the switch core. Both the linecard and switch core maintain a “request counter” for each queue, where a queue is maintained for each priority of every output port. Whenever the linecard transmits a request to the switch core it increments the appropriate counter; when the switch core receives a request it also increments its counter. When the switch core transmits a grant to the linecard it decrements the appropriate counter; when the linecard receives a grant and transmits a segment payload, it decrements its counter. Clearly, the linecard may only issue new requests when the appropriate counter does not exceed the maximum number of outstanding requests the switch core can queue. To prevent throughput from becoming degraded, the switch core and line card must be able to hold in excess of one RTT (round-trip-time) of requests for each queue.

The time from when the switch core issues a grant to when the corresponding segment payload arrives from the linecard, will depend on the RTT. Hence, the switch core must synchronize arriving segment payloads with their grant. To aid this process, the switch core attaches a sequence number to each grant. The linecard attaches the same sequence number to the transmitted segment payload.

There is no equivalent three-phase mechanism between the switch core and the egress linecard. In other words, LCS does not support queue specific egress flow-control. Instead, priority specific flow control is provided through the use of grant and payload hole requests discussed in "2.5 Flow-Control and “Holes”". In the absence of a pending hole request, grants or payloads that become available at the output of the switch core are immediately transmitted to the egress linecard.

Based on the protocol listed above, the following latency relationships must be met.

Table 9. Request and Grant Latency Requirements

	Linecard plus media	Relationship	Switch Core
Minimum Request to Grant[a]	a us[b]	>	v us
Minimum Grant to Segment[a]	b us	>	w us
Maximum Grant to Segment[a]	c us	<	x us

a. Latencies are defined at the switch interface and do not include media delays
b. This value represents the min. number of requests the linecard must generate in order to keep the pipeline full.

2.4 Sequence and Error Control

There are a number of different types of error that can occur in the process described above. We need to ensure that when errors occur, the protocol can quickly recover, continue to operate and that segments are not transmitted to the wrong destination.

The first issue is: if a grant or ingress segment payload is lost, how does the switch core reunite arriving payloads with the grants that were issued? For this, LCS uses a grant sequence number (*Sequence_ Number*). The mechanism operates as follows. Every time the switch core issues a grant, it attaches a sequence number. When the linecard receives the grant, it simply attaches the same sequence number to the outgoing payload. If a grant or payload is lost, it will be detected by the switch core. No action is needed by the linecard.

The second issue is: what happens if a request or grant is lost? Won't the linecard and switch core request counters become inconsistent? Indeed they will. In LCS, two different mechanisms are used to correct inconsistencies: one mechanism for unicast, and one for multicast.

2.4.1 Maintaining Consistent Values for Unicast Request Counters

Here, we consider how the unicast request counters evolve over time during normal operation, then see how the counters can be corrected if they become inconsistent.

A linecard and its port on the switch core maintain a separate request counter for each queue. Let's consider one unicast queue, and its associated request counters at time-slot n: $C_L(n)$ counts the number of requests for which the linecard has not yet received a grant from the switch-core, and $C_S(n)$ counts the number of requests for which the switch core has not yet issued a grant to the linecard. $C_L(n)$ is incremented each time the linecard sends a request, and decremented each time it receives a valid grant; likewise, $C_S(n)$ is incremented each time the switch core receives a valid request from the linecard, and decremented each time the switch sends a grant to the linecard. During normal, correct operation, $C_L(n)$ and $C_S(n)$ may have different values; in fact:

$$C_S(n) = C_L(n) - R(n) - G(n), \qquad (1)$$

where: $R(n)$ = the number of requests in-flight (for this queue) between the linecard and the switch-core, and $G(n)$ = the number of grants in-flight (for this queue) between the switch-core and linecard. It follows that during normal operation,

$$C_S(n) \le C_L(n) \tag{2}$$

Note that Equation (2) still holds if requests and grants are lost.
As a precursor to looking at the recovery mechanism, consider what happens if $C_S(n)$ is artificially large; e.g. if

$$C_S(n) = e(n) + C_L(n) - R(n) - G(n), \tag{3}$$

where $e(n)$ represents the number of “extra” requests held by the switch-core. This arises when the switch-core holds requests which have not been made by the linecard, and will lead to grants being sent to the linecard for segment payloads that don’t exist. If the linecard simply ignores grants to empty queues, this situation is unstable and the counters will correctly converge to Equation (1) over time.
Now let’s consider the recovery mechanism. The linecard sends a Request Count Control Packets to the switch core, carrying the value $C_L(n)$. When the switch core receives the control packet (say, at time $n + k$), it simply adopts the new value, and therefore immediately sets

$$C_S(n+k) = C_L(n) = C_L(n+k) - R(n+k) \tag{4}$$

and continues to operate. If no grants are in flight, i.e. $G(n+k) = 0$, then the counters are immediately consistent and the process is complete. If however, $G(n+k) > 0$, then the switch core counter contains $G(n+k)$ too many requests, and

$$C_S(n+k) = e(n+k) + C_L(n+k) - R(n+k) - G(n+k). \tag{5}$$

Fortunately, as we saw above, this situation is unstable and over time the counters will converge to their correct values in Equation (1).

Summary

A request counter is made consistent by the linecard sending a Request Count Control Packet to the switch-core (Refer to "2.6.1 LCS Control Packets" for a description of Control Packets). When the switch core receives a request count from the linecard, it always adopts the new value. i.e. The linecard’s request counter value is considered to be the “master” copy; when a Request Count Control Packet is sent, the switch core will immediately adopt the value given to it by the linecard. Apart from sending the Control Packet, no other corrective action is required by the linecard.

A linecard must either send Request Count Control Packets periodically, or send when CRC-errors have been detected. If periodic updates are sent, the frequency is at the discretion of the linecard and should be based on the expected bit-error rate.[1] Updates should not be sent too often, because an update can cause grants to be generated for empty queues, and hence waste bandwidth.

1. For example, consider a link with a the BER of 10^{-15}. Hence, one segment is expected to be in error approximately every 10^{12}. If, for example, the linecard and switch core maintain as many as 1000 request counters, then sending an update Control Packet every 100,000 segment times will be more than adequate. This represents a loss of just 0.001% of the ingress data rate.

2.4.2 Maintaining Consistent Values for Multicast Request Counters

Maintaining consistent values for multicast counters is more difficult. This is because the counter at the switch core for a multicast queue really consists of a list of requested multicast labels. Because of this, it is not valid to send a non-zero value for a Request Count Control Packet for a multicast queue.

Consider what happens if a multicast segment, C, is waiting at the head of a multicast queue on the linecard. Under normal operation, the next valid grant corresponding to C's queue will contain the same label as C. However, if C's request or grant was lost in transit, the next grant correctly received by the linecard will (probably) not match C's label. In fact, the grant for C will never arrive, and so C must be dropped.

We need to make sure that the linecard can re-synchronize with the arriving stream of multicast grants.

Method 1

One simple, but drastic, way for the linecard to re-synchronize is as follows. When an arriving multicast grant does not match the head of line segment payload, reset the multicast queue at the linecard and send a zero Request Count Control Packet to the switch core. This will clear all but the multicast payloads that have already been granted and whose grants are in flight back to the linecard. All of the multicast payloads in the queue will be dropped at both ends of the link. The linecard should not send any new requests for at least one maximum RTT to allow the in-flight grants to arrive, and in the meantime drop any newly arriving grants. The multicast queues at both ends are now empty and the linecard can resume normal operation.

Method 2

A less drastic approach when a grant does not match the head-of-line segment payload is to drop the first *two* payloads in the multicast queue. If only one request (or grant) was lost, then the next grant will match correctly with the head-of-line payload. If more than one request (or grant) was lost, then the linecard will eventually re-synchronize. Note that this requires additional memory operations, potentially increasing the memory bandwidth requirements of the linecard.

Method 3

Alternatively, if the linecard can perform two label comparisons per segment time, it can simply compare the arriving grant with the head of line segment payload, *and* with the payload immediately behind it. If the grant matches the head-of-line payload, send it. If it matches the second payload, send it and drop the head-of-line payload. If it matches neither payload, then drop both.

Methods 2 and 3 do not catch one pathological error condition — what if a bit-error on the line caused a legal multicast request or grant to be fabricated (i.e one with a legal CRC)? Although this is extremely unlikely, it is possible. In this case, Methods 2 and 3 will cause segment payloads to be dropped from the queue indefinitely. A simple way to detect this condition is to place a threshold on the number of consecutive payloads that can be dropped. If the threshold is exceeded, invoke Method 1, or reset the port of the switch core.

2.5 Flow-Control and “Holes”

LCS does not support per-input credit-based flow-control. In other words, segment payloads and grants available at the output of the switch core are immediately transmitted to the linecard.

However, LCS does support a simplified flow-control mechanism using “holes” and operates as follows. A “hole” is said to occur if, in a single time slot, the switch core does not send a segment payload or grant to the linecard. The linecard may explicitly ask for “holes” in both the payload stream and grant stream from the switch core to the linecard. For each *Payload_Hole_Request* that the linecard sends with bits set corresponding to one or more priority values, the switch core will ensure that at least one segment payload “hole” occurs at a later time for that priority, including both unicast and multicast best-effort service. Note that Control Packet (CP) and TDM services are not included in the payload hole considerations. In a similar fashion, for each *Grant_Hole_Request* that the ingress linecard sends with bits set corresponding to one or more priority values, the switch core will ensure that at least one grant “hole” occurs at a later time for that priority, also including both unicast and multicast best-effort service and excluding CP and TDM services. The segment stream may be stopped indefinitely by sending a continuous stream of hole requests with all four priority bits set.

Support of hole requests is required for LCS compliance, while support of hole requests on a per priority basis is optional. For LCS compliance, a linecard that does not support per priority hole requests will set all four hole request bits to request a “hole”. A switch that does not support per priority hole requests will logically “OR” all four hole request bits to generate a “hole”.

The time between the request and the “hole” occurring must be specified for each LCS device. The relationship between the linecard and switch core is defined in Table 10 below.

Table 10. Hole Request Latency Requirements

	Linecard plus media	Relationship	Switch Core
Maximum Grant_Hole_Request[a]	d us	>	y us
Maximum Payload_Hole_Request[a]	e us	>	z us

a. Latencies are defined at the switch interface and do not include media delays

Even though the linecard intends to send a continuous stream of hole requests, that stream might be interrupted by the need to send an Idle segment. An LCS compliant fabric will consider an Idle segment to have the same values for the hole request bits as were received in the previous non-Idle segment. In effect, the hole request bits will be “sticky” for the Idle segment. This is to prevent buffer overrun at the linecard.

2.6 Control Packets

The LCS link is controlled by Control Packets (CPs) that are generated by the linecard or the switch core. When the switch core sends a CP to the linecard, it marks the correct Payload Label_2 and Type_2 fields as a Control Packet and sends the control information in the payload data. When the linecard wishes to send a CP to the switch core, it goes through the normal three phase request/grant/transmit process. The exception to this process is the optional Single Phase Control Packets (SPCPs) described in "2.6.3 Single-phase Control Packets (Optional)". Control Packets have a higher priority over other traffic; therefore, grants will be returned in response to CP request prior to other outstanding requests.

There are three categories of CPs: (1) LCS Control Packets that are used to perform low-level functions such as link-synchronization, (2) CPU Control Packets which are used for communication between the linecard and the switch core CPU, and (3) Single Phase Control Packets which are used as a special form of LCS Control packets for timing sensitive information.

2.6.1 LCS Control Packets

LCS CPs are specified by setting the Label_1/Label_2 Field to 14’h0000 (CP-category for LCS Control Packets) and associated Type Field to 4’h0 (Type for Control Packets). Table 11 describes the contents of the first 10-bytes of the LCS segment payload for an LCS CP format.

Table 11. LCS Control Packet Format

Field	Size (bits)	Definition
CP_Type	8	Indicates the type of LCS Control Packet (e.g. TDM)
CP_Type Extension	56	Indicates the information carried with this type of control packet. Unused bits must be set to zero.
CRC-16	16	16-bit CRC calculated over 10-byte control packet payload.[a]

a. See "2.8 Use of CRC Field" for details of CRC calculation.

Because there is no flow control mechanism for LCS CPs sent to the switch fabric, and a CP request/grant does not distinguish the CP_Type, care must be taken so as not to overflow the receiving CP queues. At most, only one request for each CP_Type should be outstanding.

In what follows, the three CP_Types of currently defined LCS CPs are described.

TDM Control Packets (Optional)

The format of TDM CPs is described in Table 12.

Table 12. TDM Control Packet Format

Field	Size (bits)	Definition
CP_Type	8	00000000
TDM Frame Sel	1	Indicates which TDM frame to use.
Unused	55	Set to zero.
CRC-16	16	16-bit CRC calculated over 10-byte control packet payload.

Note that when a TDM CP is sent from the linecard to the switch core, no reply is generated by the switch core.

Request Count Control Packets

Request Count CPs are only sent from the linecard to the switch, and are used to keep request counter values consistent between the linecard and the switch core. When the switch core receives this CP, it updates the corresponding request counter (specified by Label_1 and Type_1) with the count field. This control packet can be sent periodically or when the linecard suspects that the linecard and switch core have different values (e.g. following a CRC error).

Request Count CPs are required for unicast traffic and also required for multicast and TDM requested traffic when supported. For unicast traffic, the Label_1 field is set as described in Table 6. For multicast traffic, the Label_1 field is undefined and should be set to zero. The format of the Request Count CP is described in Table 13

Table 13. Request Count Control Packet Format

Field	Size (bits)	Definition
CP_Type	8	00000001
Label_1	14	Label for request counter.
Type_1	4	10xx unicast 11xx multicast
Count	14	Update counter value, always set to zero for multicast.
Unused	24	Set to zero.
CRC-16	16	16-bit CRC calculated over 10-byte control packet payload.

Start/Stop Control Packets

Start/Stop Control Packets are used to toggle the flow of data traffic over the link (here, data traffic is defined to be TDM, best-effort unicast and multicast segment payloads and associated requests and grants. i.e. all non-Control Packet segments). Start/Stop Control Packets can be sent over an LCS link in either direction.

If a linecard sends a Stop Control Packet to the switch core, the switch core will stop sending grants to the linecard for ingress data traffic, and will stop sending egress data traffic to the linecard. When a Start Control Packet is sent, grants and egress data traffic resume.

If the switch core sends a Stop Control Packet to a linecard, the linecard must stop sending new data traffic requests into the switch core until a Start Control Packet is sent.

Note that when either end has received a Stop Control Packet, it can continue to request, grant, send and receive all categories of Control Packets.

When an LCS link powers-up, it will be in Stop mode: i.e. it will act as if a Stop Control Packet has been sent in both directions.

Table 14. Start/Stop Control Packet Format

Field	Size (bits)	Definition
CP_Type	8	00000010
Start/Stop	1	1=Start, 0=Stop.
Unused	55	Set to zero.
CRC-16	16	16-bit CRC calculated over 10-byte control packet payload.

2.6.2 CPU Control Packets (Optional)

CPU Control Packets are specified by setting the Label_1/Label_2 Field to 14'h0001 (CP-category for CPU Control Packets) and associated Type Field to 4'h0 (Type for Control Packets).

When the linecard sends a CPU CP to the switch core CPU, the complete payload data is passed to the switch core CPU. Likewise, when the switch core sends a CPU CP to the linecard, the complete payload data is passed to the linecard.

No flow control mechanism is explicitly implemented for CPU CPs, and will be implementation dependent.

2.6.3 Single-phase Control Packets (Optional)

Single-Phase CPs (SPCPs) are a form of LCS Control Packets. SPCPs can communicate a subset of the information that LCS Control Packets can send. Single-phase Control Packets are embedded entirely within a Request Label_1, eliminating the need for an LCS request/grant exchange. They are useful for sending very timing-sensitive information.

The specific Request Label_1 field values used for different categories of SPCPs are also described below. Not all LCS-compliant products support this feature.

Single-Phase Control Packets use the Request Label_1 and Type_1 fields as shown in Table 15. These have the same effect as the corresponding LCS Control Packet. They are provided as a convenient way of transmitting time-sensitive information without requiring a full request/grant exchange.

Table 15. Single-Phase Control Packet Request Label_1 and Type_1 Usage

Field	Size (bits)	Definition
Request Label_1 Reserved Bits	6	000000 (Set to zero.)
Request Label_1 SPCP-category	8	00000000 -- TDM Frame Sel 0. 00000001 -- TDM Frame Sel 1.
Type_1	4	0001 (Type SPCP)

2.7 Request Counters in the Linecard

The linecard must implement a request counter for each of the request queues in the switch core to which it will issue requests. Clearly the linecard does not need to support a different request counter for each of the 2^{14} possible label values, and indeed this is not necessary. Linecard design should take into account the maximum number of ports they will need to support for the range of switch fabrics they will be used with. The number of request counters required is directly related to the number of ports, priorities, and service classes they will support.

The size of each Request Counter is based on the number of segments in a Round Trip Time (RTT) for the switch fabrics that linecard will operate with. Therefore, it is dependent on the Minimum Request to Grant latency that a switch fabric operates, plus the additional latency required by the transmission medium delay and linecard processing time, divided by a segment transfer period. For example, if the Minimum Request to Grant Latency is 4 us, the medium delay is 1 us, the linecard processing time is 1 us, and the segment transfer period is 40 ns, the Request Counters must support at least 150 segments. The maximum supported LCS count is 2^{14}.

2.8 Use of CRC Field

Control Packets and LCS protocol segment prepends use the CRC-16 polynomial: $x^{16} + x^{12} + x^5 + 1$. This is the same polynomial used in X.25.

1. Prior to starting the calculation, the remainder is set to 0xffff.
2. In all cases, the CRC-16 is calculated over all bits (80-bits of the CP or 64-bits of the prepend) with the CRC field set to all zeroes.
3. Transmission of the CRC-16 is done with the most significant bit sent first.

The following are some sample CPs and prepends, showing the CRC-16 generated for each. The CRC-16 field is always the last 16 bits.

LCS Start CP:

0x0280000000000000C566

Request Count CP:

0x01037280240000006FDB

LCS Prepend from Switch to Linecard:

0x8719362C09DCC9C7 [1] (subport = 0)

0xE719362C09DCC9C7 [1] (subport = 3, MUX masked to 0 for CRC)

0xE719362C09DC96DF (subport = 3, MUX included in CRC)

1. Note the MUX[1:0] field is assumed equal to “00” in these CRC-16 calculations.

LCS Prepend from Linecard to Switch:

0x81B9409D00106039 [1] (subport = 0)

0xE1B9409D00106039 [1] (subport = 3, MUX masked to 0 for CRC)

0xE1B9409D00103F21 (subport = 3, MUX included in CRC)

3 LCS PHY LAYER

The LCS PHY provides a high-speed parallel transfer of data in the form of framed, fixed-sized segments that have been channelized across multiple media connections. Higher segment rates are also supported through trunking options, while lower segment rates are supported through multiplexing options.

The channelization, framing, and transmission functions of the LCS PHY have been specified to allow use of a variety of available Serdes (serialization/deserialization) and parallel fiber optics components. Portions of this specification will refer to examples of media types. The LCS PHY specification does not preclude other compatible options.

Channelization requirements include the mapping of segments to multiple channelized frames, channel skew specifications, and interchannel (segment) alignment operation.

Frame alignment, encapsulation, retiming and the required code-points for a parallel, full-duplex interface are included in the framing function.

Transmission requirements include the clocking requirements and electrical characteristics of the parallel interface.

The parallel interface used by the LCS PHY is based on widely available Serdes implementations that include their own endec (encoder/decoder) functionality. These components also fit well with the available and emerging parallel fiber optics components. Optionally, an LCS device can include the Serdes and 8B/10B Endec functions, but these are not strictly required by the LCS specification. In fact, LCS implementations that forego the use of optical transceivers or Serdes and endecs altogether for single board design are also compliant with the LCS specification.

3.1 Link Speed Options

The LCS PHY supports two options in link speeds, 1.5 Gbaud and 2.5 Gbaud. These link speed options have various requirements that affect channelization and framing functions, discussed in their respective sections. Serdes and optical transceiver components are available to support the 1.5 Gbaud links, while 2.5 Gbaud link component are currently under development.

3.2 Segment Payload Options

The LCS Protocol uses an eight byte prepend and supports two segment payload sizes, a 64-byte payload with an eight byte LCS prepend (8+64 byte segment) and a 76-byte payload with an eight byte LCS prepend (8+76 byte segment). Support of the various sized payloads is handled differently for each link speed. Discussion of these options is also covered in the channelization and framing sections.

3.3 Segment Channelization

The LCS Physical Layer includes the external interface for receiving and transmitting LCS segments from and to the linecard. The interface consists of 1 or more parallel “ganged” interfaces that each carry a serialized portion of a segment called a frame. An LCS linecard or switch should therefore state the number of channels in a gang and the bytes per channel utilized for its particular implementation.

In the receive direction each channel receives a serial stream of 8B/10B coded data which can be decoded into a parallel byte stream plus two additional bits - one to indicate a control code-point and one for an errored code-point. In the transmit direction, an eight bit byte of parallel data plus one bit to indicate control word is sent through each channel as serialized 8B/10B encoded data.

The optional parallel interface has separate ingress and egress 10-bit parallel buses. The busses operate in source synchronous mode and so carry a clock with them to latch the data. A source synchronous transmit clock is driven from the LCS device interface to the Serdes devices. For data arriving from the remote LCS device, each Serdes device recovers the receive clock from the bit stream and provides it to the local LCS device’s parallel bus receiver interface.

Table 16, Table 17 and Table 18 show the different external interface configurations supported. When utilizing the parallel interface, a Serdes device is used to convert between the byte-stream and the 8B/10B encoded serial stream. This Serdes interface can either be a Single Data Rate (SDR) for the 1.5 Gbit/s interface or Double Data Rate (DDR) for the 2.5 Gbit/s interface.

Table 16. Backplane 1.5 Gbit/s per Channel Interface Configurations

Gang of Channels	Data Frame Size	Segment Size	Segment Time	Segments per second	Channel Data Rate	Channel Line Rate
1	72	72	480 ns	2.08 M	1.50 Gbits/s	1.5 SDR
2	36	72	240 ns	4.17 M	1.50 Gbits/s	1.5 SDR
4	18	72	120 ns	8.33 M	1.50 Gbits/s	1.5 SDR
12	6	72	40 ns	25 M	1.50 Gbits/s	1.5 SDR
15	5	72	33.3 ns	30 M	1.50 Gbits/s	1.5 SDR

Table 17. Fiber 1.5 Gbit/s per Channel Interface Configurations

Gang of Channels	Data Frame Size	Segment Size	Segment Time	Segments per second	Channel Data Rate	Channel Line Rate
3	24	72	160 ns	6.25 M	1.50 Gbits/s	1.5 SDR
12	6	72	40 ns	25 M	1.50 Gbits/s	1.5 SDR
14	6	84	40 ns	25 M	1.50 Gbits/s	1.5 SDR

Table 18. Fiber 2.5 Gbit/s per Channel Interface Configurations

Gang of Channels	Data Frame Size	Segment Size	Segment Time	Segments per second	Channel Data Rate	Channel Line Rate
3	32	84	160 ns	6.25 M	2.00 Gbits/s	2.5 DDR
3	32	84	144 ns	6.94 M	2.22 Gbits/s	2.5 DDR
12	8	84	40 ns	25 M	2.00 Gbits/s	2.5 DDR
12	8	84	36 ns	27.78 M	2.22 Gbits/s	2.5 DDR

Table 19. Mapping Segments to 1.5 Gbit/s Channels

Gang of Chnls	Data Frame Size	Channel 1	Channel 2	Channel 3	Channel 4	Channel 8	Channel 9	Channel 12	Channel 14	Channel 15
15	5	Prepend 0-4	Prepend 5-7, Pyld 0-1	Payload 2-6	Payload 7-11	Payload 27-31	Payload 32-36	Payload 47-51	Payload 57-61	Payload 62-63, Pad 3[a]
14	6	Prepend 0-5	Prepend 6-7, Pyld 0-3	Payload 4-9	Payload 10-15	Payload 34-39	Payload 40-45	Payload 58-63	Payload 70-75	
12	6	Prepend 0-5	Prepend 6-7, Pyld 0-3	Payload 4-9	Payload 10-15	Payload 34-39	Payload 40-45	Payload 58-63		
4	18	Prepend 0-7, Pyld 0-9	Payload 10-27	Payload 28-45	Payload 46-63					
3	24	Prepend 0-7, Pyld 0-15	Payload 16-39	Payload 40-63						
2	36	Prepend 0-7, Pyld 0-27	Payload 28-63							
1	72	Prepend 0-7, Pyld 0-63								

a. Pad n is the number of additional bytes that fill the remaining frames. All pad bytes are set to zero.

Table 20. Mapping Segments to 2.5 Gbit/s Channels

Gang of Chnls	Data Frame Size	Channel 1	Channel 2	Channel 3	Channel 11	Channel 12
12	8	Prepend 0-7	Payload 0-7	Payload 8-15	Payload 72-75, Pad 4[a]	Pad 8[a]
3	32	Prepend 0-7, Payload 0-23	Payload 24-55	Payload 56-75, Pad 12[a]		

a. Pad n is the number of additional bytes that fill the remaining frames. All pad bytes are set to zero.

An LCS segment consists of an eight byte prepend and either a 64- or 76-byte payload. The start of the prepend is sent on the first channel, any remainder of the prepend is sent on the next channel, followed by the start of the payload. The remaining channels carry the subsequent bytes of the payload. The exact mapping of bytes to channels is shown Table 19 and Table 20.

An 8+64 byte segment is not explicitly supported when using the 2.5 Gbaud links. However, any smaller segment payload than 76 bytes can be supported by padding the fixed length segments to the full 76-byte payload.

3.4 Alignment and Framing Functions

Typical LCS devices will accommodate the interchannel skew with FIFOs and an alignment procedure based on the framing structures. Segment alignment is distinguished from frame alignment and symbol alignment with the following definitions and alignment functions:

- A segment consists of a 72-byte or 84-byte LCS traffic unit that is spread between multiple, parallel channels for transmission between LCS interfaces.

  Segment alignment is the process used to realign frames from separate channels as one complete segment for further processing.

- A frame consists of a portion (slice) of a segment transmitted through a single channel between LCS interfaces.

  Frame alignment is the process used to identify the boundaries of a received frame so that it can be written into a receiving FIFO as a complete frame.

- A code-point is the non-encoded data that is transferred between the LCS device and an appropriate Serdes device, when the LCS device does not include the encoding or serializing functions. A symbol is the 8B/10B encoded 10-bit sequence of the code-point.

  Symbol alignment is the process by which symbol boundaries are distinguished from a received serial 10-bit sequence. Usually this process is performed by a Serdes function that is either part of an LCS device or a standalone device connected to an LCS device by a parallel interface. The symbol is then decoded and presented to the remaining receiving functions as a code-point.

3.4.1 Symbol Encoding

Code-points are translated into ten bit transmission symbols by an encoding function. The typical block encoder used for LCS is the 8B/10B code also used for Fiber Channel and Gigabit Ethernet. Specific code-points used for LCS are listed in Table 21.

Data code-points consist of eight signals that represent a data byte and a ninth signal denoting the code-point as a data and not control.

Control code-points use the ninth signal to designate a control code-point, with the other eight signals identifying the individual control code-point. The three different control code-points used by LCS that appear in the Idle frame sequence are:

- COMMA: this control code-point is always sent as the first code-point of the idle sequence. The encoded COMMA symbol is used by the Serdes in order to obtain symbol alignment, while the decoded COMMA code-point is used by the Receive State Machine to find the frame boundaries.
- IDLE_NR (idle not ready): this control code-point is sent in the subsequent code-points of the Idle frame when the local receiver is not ready.
- IDLE_R (idle ready): this control code-point is sent in the subsequent code-points of the Idle frame when the local receiver is ready.

The COMMA code-point is chosen to match the same code-point used by many Serdes devices for symbol alignment, and the initialization protocols used by the LCS PHY ensure its use for this purpose. The choice of control code-points for IDLE_NR and IDLE_R complies with the commonly used 8B/10B transmission rules.

Table 21. Code-Points

Control Codes	8B/10B Equivalent	Code-Point	Usage
Idle			
COMMA	K28.5	[bit 9] = ERR, [bit 8] = 1, [bits 7:0] = 10111100	Denotes start of Idle frame followed by five idle codes for 1.5 Gbaud link or one idle code for 2.5 Gbaud link
IDLE_NR	K27.7	[bit 9] = ERR, [bit 8] = 1, [bits 7:0] = 11111011	Denotes receiver not ready
IDLE_R	K29.7	[bit 9] = ERR, [bit 8] = 1, [bits 7:0] = 11111101	Denotes receiver ready
Data Codes			
0-256	Dxx.x	[bit 9] = ERR, [bit 8] = 0, [bits 7:0] = 8-bit data value	Data frame

3.4.2 Symbol Alignment

The COMMA code-point is translated to the 8B/10B block coded symbol for serial transmission. This is often referred to as K28.5 by common Serdes devices. When used by the Serdes, the K28.5 symbol provides a unique bit sequence that can be used to identify the symbol boundary of a bit sequence. This symbol is then translated back into the eight-bit control code-point used by the LCS PHY interface.

IDLE_NR and IDLE_R are control code-points used to distinguish the frame boundary and to signal the LCS interface's current receive state to the far end transmit state machine. Receive state status is used in the initialization process, and can be monitored during operation.

3.4.3 Frame Alignment

The LCS interface uses a COMMA to designate the first code-point in a control frame called an Idle frame. It is filled out by IDLE_NR or IDLE_R control code-points to complete the interface's fundamental frame size. More precisely, an Idle-Rdy control frame consists of a COMMA code-point followed by N-1 IDLE_R code-points, where N equals the frame size. An Idle-Not-Rdy control frame consists of a COMMA code-point followed by N-1 IDLE_NR code-points, where N equals the frame size. Idle frame formats are shown in Figure 1. Together, these control code-points define a frame boundary. Should a frame ever be received with a mix of code-points, e.g. a frame that contains both data and control code-points, the receiver should indicate an error and maintain alignment. A more complete discussion of this error condition is covered in 3.4.6 "Frame Processing".

Figure 1. Idle Frames

COMMA	IDLE_NR	IDLE_NR	...	IDLE_NR	IDLE_NR

COMMA	IDLE_R	IDLE_R	...	IDLE_R	IDLE_R

When Idle frames are followed by data code-points, the Data frame boundaries are defined simply by successive sequences of data code-points equaling the fixed frame length. In this way, frames can be sent back-to-back utilizing the entire bandwidth of the link. Data frame formats are shown in Figure 2, where n equals the number of byte for the chosen frame length.

Figure 2. Data Frame

DATA0	DATA1	DATA2	...	DATA n-1	DATA n

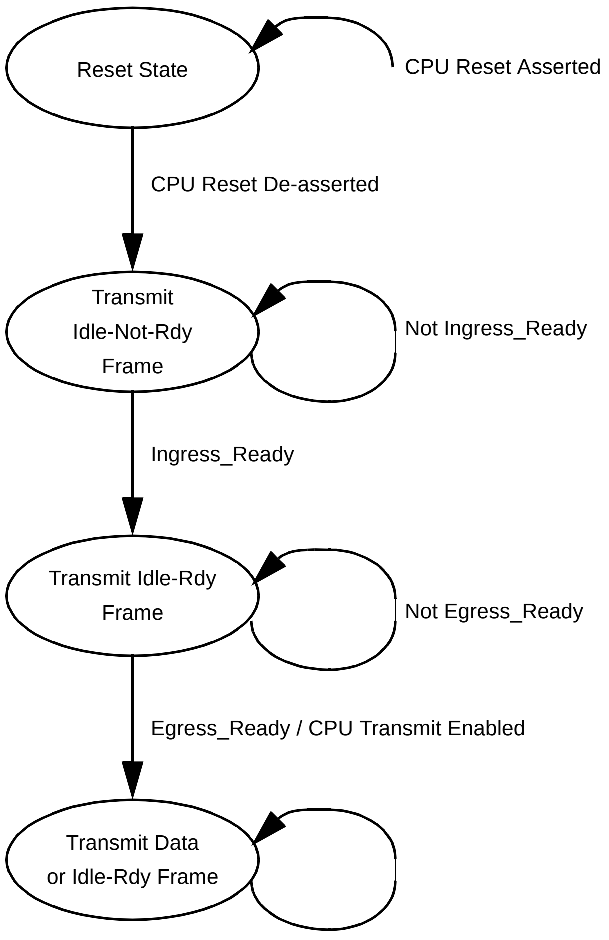
3.4.4 Data Transmission

The Transmit State machine is in Figure 3. To bring the link up, an external interface such as a CPU first disables Data frame transmission. This causes the transmit state machine to send continuous Idle frames.

At first the local receiver link will not be aligned to Idle frame boundaries, so the transmit state machine will send Idle-Not-Rdy frames. Once the local receiver is aligned (receive state machine enters Ingress_Ready state), the transmitter will either automatically or with the aid of a CPU start sending Idle-Rdy frames. When the remote receiver is aligned, it starts sending Idle-Rdy frames that the local receiver detects noting that the local transmitter's Idle stream is aligned at the far-end (local receive state machine enters Egress_ Ready state).

The CPU then enables Data frame transmission, and the transmit state machine can send either Idle frames or Data frames. Idle frames should be sent for ppm reasons (see 3.5.1 "Frequency Compensation"), or if the transmit logic does not have an entire segment to send. For the later, it is also valid to send Data frames with all segment fields marked invalid (effectively all null fields).

Figure 3. Transmit State Machine



3.4.5 Data Frame Reception

There are three levels of synchronization required before segments can be received successfully. Firstly, the serial stream is turned into code-points, known as symbol alignment. Secondly, frame alignment, that is the process by which the frame boundaries are found, is performed. Finally, a complete segment is assembled by alignment of related frames across all the utilized channels, called segment alignment.

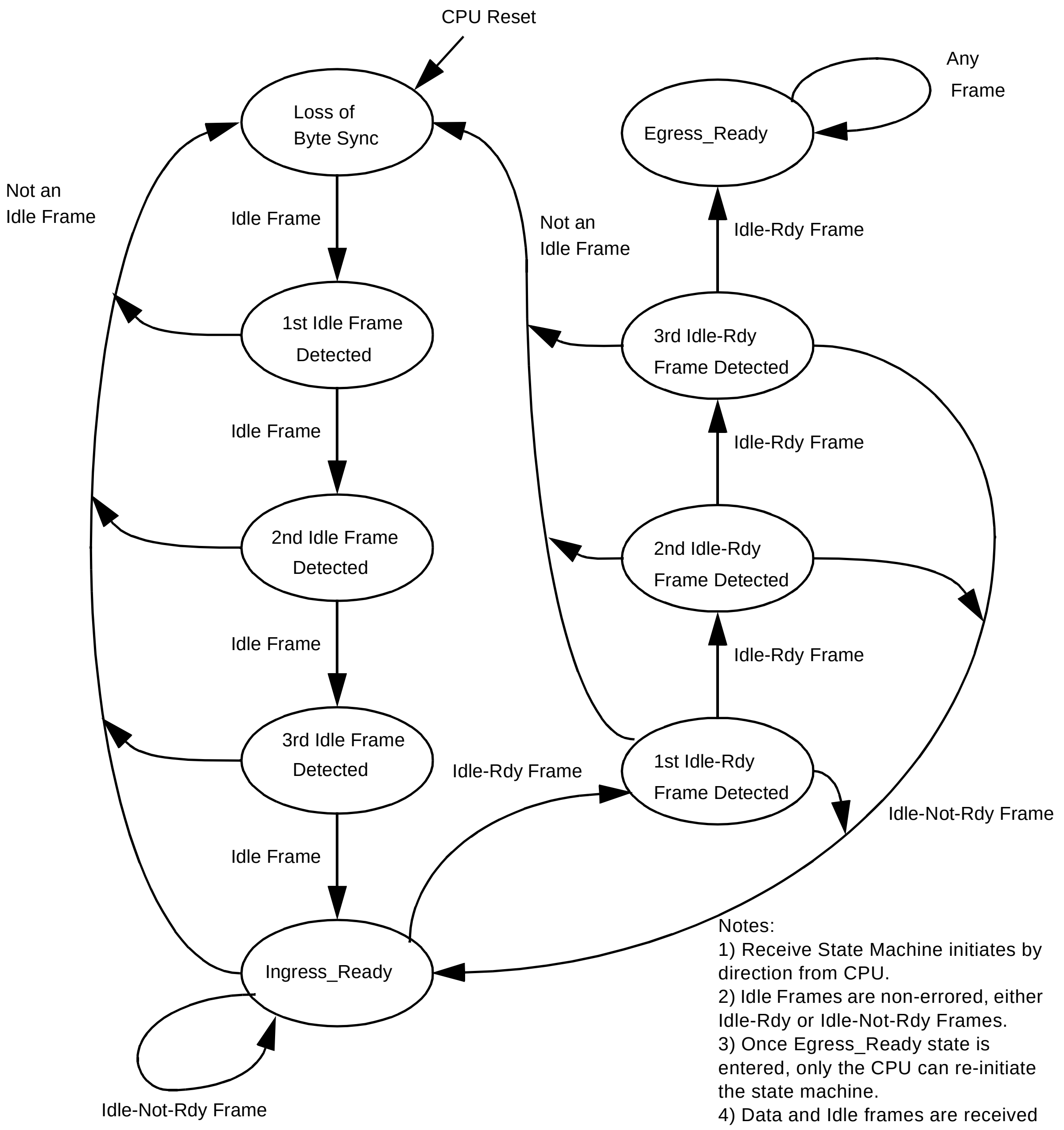
These terms are defined earlier, while their implementation requirements are explained as part of the Data frame reception process here.

The receiver will only be able to initialize if Idle frames are continuously sent to it. Thus the transmitter at the other end of this link must be programmed to send Idle frames until the receiver is synchronized.

The Serdes devices are responsible for restoring the parallel data, or code-points, from the received serial bit stream. They do this by recognizing the COMMA symbol of positive disparity (which are sent as part of the Idle frame) and then adjusting their receive clock to align the parallel code-points correctly.

The LCS receive interface is responsible for establishing frame synchronization. The receiver does this by looking for Idle frames (either Idle-Not-Rdy or Idle-Rdy) using the state machine shown in Figure 4. The receiver is considered synchronized when it receives 4 consecutive Idle-Not-Rdy or Idle-Rdy frames, and goes into the Ingress_Ready state. Once this happens the receiver automatically instructs the local transmitter to send Idle-Rdy frames. Once in the Ingress_Ready state the receiver logic must receive 4 consecutive Idle-Rdy frames before it considers that the remote receiver is ready and goes into the Egress_Ready state. Note that the receiver can only get out of the Egress_Ready state by being reset by the CPU.

Figure 4. Receive State Machine.



Notes:
1) Receive State Machine initiates by direction from CPU.
2) Idle Frames are non-errored, either Idle-Rdy or Idle-Not-Rdy Frames.
3) Once Egress_Ready state is entered, only the CPU can re-initiate the state machine.
4) Data and Idle frames are received in Egress_Ready state.

3.4.6 Frame Processing

Skew between separate channels in a gang can occur. Once in the Egress_Ready state, segment alignment between ganged channels must accommodate the maximum interchannel skew by adjusting for skew differences. In order to implement a channel skew compensation procedure, skew limits must be identified.

The maximum allowable interchannel skew (i.e. between any pair of channels) is limited to a maximum of 40 ns. This interchannel skew budget can be allocated to skew in the cable plant and skew from all of the transmission components. Cable plant skew can come from skew between fibers in the ribbon fiber cable and skew in any of the connectors used between the parallel optics. Component skew may be found in the parallel transceiver devices, the Serdes components, and the LCS devices.

- The channel skew compensation procedure can then be implemented in several different ways. One method requires a frame alignment process after Egress_Ready state has been reached, and the passing of Data (and not Idle) frames into a receive FIFO for each individual channel.

  To accommodate cases where a frame may be corrupted, the following criteria is used: If three or more code-points in the first five code-points in a frame are control code-points, then the frame is dropped before entering the receive FIFO. All other cases, the frame is passed through the receive FIFO. This process ensures that it is very unlikely that a Data frame will turn into an Idle frame or vice versa. This property is important to prevent an isolated corruption from causing the channels to become misaligned, requiring the entire link to be restarted to reset the frame alignment.

  This method enables the use of Idle frames to avoid receive FIFO overflow when the receiver’s frequency reference may run slightly slower (ppm) than the transmitter.

- An alternative method would write all frames (Data and Idle) into receive FIFOs for each channel. These FIFOs can be emptied at a greater rate than they can be filled, and are emptied across all channels once they all become non-empty. This method requires that the receive FIFOs do not accept frames until the first Data frames appear. After that, all Data and Idle frames are passed through the receive FIFOs.

The required initialization and segment alignment functions are covered in the next section.

3.4.7 Segment Alignment

Sometime after all the channels in both directions are in the Egress_Ready state, the remote transmitter will start sending Data frames. These must be aligned across the receiving channels so that all the frames that make up a segment are processed together, adjusting for the receive clocks on each channel that might be out-of-phase due to skew between channels. This is achieved during initialization by placing Data frames, but not Idle frames, into the receive FIFOs as part of the frame processing step mentioned previously. Once all the channel FIFOs on a given link are non-empty, a complete segment can be read from the FIFOs. In effect, this means that the first (data) segment received after a series of Idle frames is used to align all subsequent segments. This mechanism works because the number of Data frames sent on each channel is always the same.

In the unlikely event a frame processing error causes the channels on the link to be misaligned, all subsequent segments will be misaligned as well. If the misalignment is between channels within the LCS prepend, a series of LCS prepend CRC failures will occur. At this point the CPU should bring the link down and back up. If the misalignment occurs between any of the other channels carrying only payload portions of the segment, then the segment payload will become corrupted. This payload corruption must be detected by the egress linecards which can then notify the switch CPU to bring the suspected links down and re-initialize.

3.5 Transmission

3.5.1 Frequency Compensation

The switch core and the linecard may be clocked from independent frequency references that operate at the same nominal rate, with slight ppm differences in frequency. In such a system, one transmitter will run at a slightly greater speed than its far-end receiver, leading to a possible overflow in the receiver's FIFO. To avoid overflows in the case where the transmitter operates at a faster frequency than the receiver, the transmitter must send Idle segments (that is, an Idle frame on each channel) at regular periods. The Idle frames must be inserted following the previous complete segment on all channels. The period is determined by a ratio of the maximum difference in frequency between the transmitter and receiver. Assuming +/-200 ppm oscillators, and allowing for storage of at least one extra segment in the receive FIFO, then 1 in 2,500 segments must be an Idle segment to avoid overrun.

3.5.2 Signal Interface

The LCS signal interface specifies a serial electrical interface for each channel that can be used alone or with optical transceivers.

An optional parallel interface is also specified that is compatible with common external Serdes components. This interface consists of an eight-bit parallel data bus TXD[7:0] and the associated transmit clock TXC, with an extra bit TXD[8] to identify the control code-points from the data code-points. The basic receive signal interface required by the LCS PHY also consists of an eight-bit parallel data bus RXD[7:0] and the associated receive clock RXC, with two extra bits: TXD[8] to identify the control code-points from the data code-points, and TXD[9] to identify receive error codes, including invalid code-points and running disparity errors.

For the 1.5 Gbit/s links, the TXC operates at 1/10 the link rate, or 150 MHz, as does the RXC derived from the incoming data stream. Setup and hold timing requirements, as well as signal levels, will be based upon available Serdes components.

Likewise, the 2.5 Gbit/s links use a TXC and RXC that operate at a frequency derived from the link rate, but these higher speed links use both the rising and falling edges of the clocks to transfer data across this interface. This mode of operation is called a DDR (Double Data Rate) mode. The clocks used on the 2.5 Gbit/s links therefore operate at 125 MHz. Setup and hold timing requirements, as well as signal levels, will be based upon available 2.5 Gbit/s Serdes components.