RapidIO™ Interconnect Specification Part 10: Data Streaming Logical Specification

4.1, 6/2017

RapidIO.org

Revision History

Revision	Description	Date
1.3	First release	06/09/2004
1.3.a	No technical changes Converted to ISO-friendly templates	02/23/2005
2.0	Technical changes: errata showing 06-04-00002.004; new features showing 06-01-00000.002	06/14/2007
2.1	Technical changes: errata showing 07-07-00000.004	07/09/2009
2.2	Technical changes: errata showing 10-08-00000.003, 10-08-00001.005, Consolidated Comments on 11-01-00000.000	05/05/2011
3.0	Changed RTA contact information. No technical changes.	10/11/2013
3.1	No technical changes.	09/18/2014
3.2	No technical changes.	01/28/2016
4.0	No technical changes.	06/15/2016
4.1	No technical changes.	06/30/2017

Questions regarding RapidIO.org, specifications, or membership should be forwarded to:

RapidIO.org
8650 Spicewood Springs #145-515
Austin, TX 78759
512-827-7680 Tel.

Table of Contents

Table of Contents

List of Figures

List of Figures

Blank page

List of Tables

List of Tables

Blank page

Chapter 1 Overview

1.1 Introduction

This chapter provides an overview of the *RapidIO Part 10: Data Streaming Logical Specification*. The goal of the specification is to combine the need for efficiency, flexibility, and protocol independence in order to minimize the resources necessary to support a data plane interconnect fabric, and to maintain compatibility and fully inter-operate with the rest of the RapidIO specifications. Implementation of this specification is optional.

The rationale for this optimization is based upon the assumption that platforms are expected to produce many times more revenue than the initial cost of the platform. For example, a platform is expected to produce 10 times the revenue vs. its initial capital costs. If that same platform could cost 10% more but allow 10% more resources for producing revenue rather than doing fabric support, the result would be a significant net gain on the investment. Therefore, enabling more intelligence within the system fabric and relieving the system processing resources to produce revenue, even if that fabric is more expensive, is believed to be a good trade-off.

The features of the data streaming specification define virtual mechanisms in simple forms for building cost sensitive systems and also provides for complex high functioning fabrics for more demanding applications.

It is assumed that the reader has a thorough understanding of the other RapidIO specifications and of data plane equipment and applications in general.

1.2 Overview

Standard encapsulation schemes have been developed for the transmission of datagrams over most popular LANs. A number of different proposals currently exist for the encapsulation of one protocol over another protocol [RFC1226, RFC1234, RFC1701]. The data streaming logical specification defines a mechanism for transporting an arbitrary protocol over a standard RapidIO interface, and addresses interconnection between elements in an end-to-end data communications circuit. The protocol has been carefully designed to provide complete compatibility and inter-operability with existing RapidIO specifications.

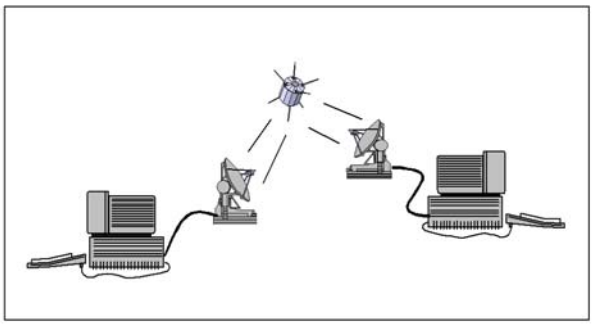

Figure 1-1. End to End Communication Circuit

The defined encapsulation methodology provides for the multiplexing of different network-layer protocols simultaneously over the same link and provides a common solution for easy connection of a wide variety of hosts, bridges and switches. It is envisioned that a RapidIO system will be capable of carrying a wide variety of data types, supporting a diverse set of protocol regimens concurrently.

The Data Streaming Protocol also includes a methodology for end-to-end traffic management. Loosely coupled systems with individual traffic managers admitting traffic to a fabric have to rely on statistical performance and back pressure to try and optimize use of the fabric resources. End-to-end traffic management allows traffic mangers at the ingress endpoints to work in concert with egress endpoints to coordinate traffic flows.

1.3 Features of the Data Streaming Specification

The following are features of the RapidIO data streaming specification designed to satisfy the needs of various applications and systems:

1.3.1 Functional Features

- Protocol encapsulation, independent of the protocol being encapsulated.
- Support for Protocol Data Units (PDUs) of up to 64k bytes through Segmentation and Reassembly (SAR).
- Support for hundreds of traffic classes.
- Support for thousands of data streams between end points.
- Support for concurrent interleaved PDUs between end points.
- Seamless inter-operability with other RapidIO specifications.

1.3.2 Physical Features

- Packet definition is independent of the choice of physical layer interconnection to other devices on the interconnect fabric.

- The protocols and packet formats are independent of the physical interconnect topology. The protocols work whether the physical fabric is a point-to-point ring, a bus, a switched multi-dimensional network, a duplex serial connection, and so forth.
- No dependencies exist on the bandwidth or latency of the physical fabric.
- The protocol requires in-order packet transmission and reception; out-of-order packet delivery is not tolerated.
- Certain devices have bandwidth and latency requirements for proper operation. The data streaming logical layer specification does not preclude an implementation from imposing these constraints within the system.

1.3.3 Performance Features

- Packet headers are small to minimize the control overhead and be organized for fast, efficient assembly and disassembly.
- Multiple transactions are allowed concurrently in the system, otherwise a majority of the potential system throughput is wasted.
- Multiple end point to end point concurrent data streams are supported for high fabric utilization.
- Optional end-to-end traffic management for advanced fabric designs.

1.4 Contents

Following are the contents of the *RapidIO Part 10: Data Streaming Logical Specification:*

- Chapter 1, “Overview,” is an overview of the data streaming logical specification.
- Chapter 2, “Data Streaming Systems,” introduces system issues such as transaction ordering and deadlock prevention.
- Chapter 3, “Operation Descriptions,” describes the set of operations and transactions supported by the RapidIO data streaming protocol.
- Chapter 4, “Packet Format Descriptions,” contains the packet format definitions for the data streaming specification.
- Chapter 5, “Data Streaming Registers,” describes the visible register set that allows an external processing element to determine the data streaming capabilities, configuration, and status of a processing element using this logical specification. Only registers or register bits specific to the data streaming logical specification are explained. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions.
- Annex A, “VSID Usage Examples,” contains a number of examples of how the virtual stream identifier can be used in a system.

1.5 Terminology

The data streaming logical specification introduces some new terms:

Protocol Data Unit - (PDU) A self contained unit of data transfer comprised of data and protocol information that defines the treatment of that data.

Virtual Stream ID - (VSID) an identifier comprised of several fields in the protocol to identify individual data streams.

Virtual input Queue (ViQ), Virtual output Queue (VoQ) - an intermediate point in the system where one or more virtual streams may be concentrated.

Class of service - (cos) a term used to describe different treatment (quality of service) for different data streams. Support for class of service is provided by a class of service field in the data streaming protocol. The class of service field is used in the virtual stream ID and in identifying a virtual queue.

StreamID - a specific field in the data streaming protocol that is combined with the data streams's transaction request flow ID and the source ID or destination ID from the underlying packet transport fabric to form the virtual stream ID.

Segment - A portion of a PDU.

Segmentation - a process by which a PDU is transferred as a series of smaller segments.

Segmentation context - Information that allows a receiver to associate a particular packet with the correct PDU.

Ingress - Ingress is the device or node where traffic enters the system. The ingress node also becomes the source for traffic into the RapidIO fabric. The terms ingress and source may or may not be used interchangeably when considering a single end to end connection.

Egress - Egress is the device or node where traffic exits the system. The egress node also becomes the destination for traffic out of the RapidIO fabric. The terms egress and destination may or may not be used interchangeably when considering a single end to end connection.

Physical Channel ID - Identifies a physical channel using the virtual channel, critical request flow, and priority physical layer fields. The physical channel ID is used to determine how to treat a packet with respect to priority, bandwidth allocation, and transaction ordering.

RapidIO flow - A RapidIO flow is a nexus of the source ID, destination ID and physical channel.

Refer to the Glossary at the back of this document for additional definitions.

1.6 Conventions

||	Concatenation, used to indicate that two fields are physically associated as consecutive bits
ACTIVE_HIGH	Names of active high signals are shown in uppercase text with no overbar. Active-high signals are asserted when high and not asserted when low.
$\overline{\text{ACTIVE\_LOW}}$	Names of active low signals are shown in uppercase text with an overbar. Active low signals are asserted when low and not asserted when high.
italics	Book titles in text are set in italics.
REG[FIELD]	Abbreviations or acronyms for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets.
TRANSACTION	Transaction types are expressed in all caps.
operation	Device operation types are expressed in plain text.
n	A decimal value.
[n-m]	Used to express a numerical range from n to m.
0bnn	A binary value, the number of bits is determined by the number of digits.
0x*nn*	A hexadecimal value, the number of bits is determined by the number of digits or from the surrounding context; for example, 0x*nn* may be a 5, 6, 7, or 8 bit value.
x	This value is a don’t care.

1.7 Useful References

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC791, September 1981

[RFC1226] Kantor, B. "Internet Protocol Encapsulation of AX.25 Frames", RFC1226, University of California, San Diego, May 1991.

[RFC1234] Provan, D. "Tunneling IPX Traffic through IP Networks", RFC 1234, Novell, Inc., June 1991.

[RFC1700] J. Reynolds and J. Postel, "Assigned Numbers", RFC1700, October 1994.

[RFC2460] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6)", RFC2460, December 1998.

[RFC1884] Hinden, R., and S. Deering, Editors, "IP Version 6 Addressing Architecture", RFC1884, Ipsilon Networks, Xerox PARC, December 1995.

[RFC2004] C. Perkins, "Minimal Encapsulation within IP", RFC2004, October 1996.

Chapter 2 Data Streaming Systems

2.1 Introduction

This overview introduces the role of the data streaming logical layer in an overall system. It provides some possible use examples. See Annex A, “VSID Usage Examples”, for more example details.

2.2 System Example

Figure 2-1 shows a block diagram of an example RapidIO-based networking system in which protocol encapsulation is required. A number of typical data path type devices are connected with a variety of proprietary and/or somewhat standard interfaces and the entire system is tied together with a RapidIO switching fabric of some topology.

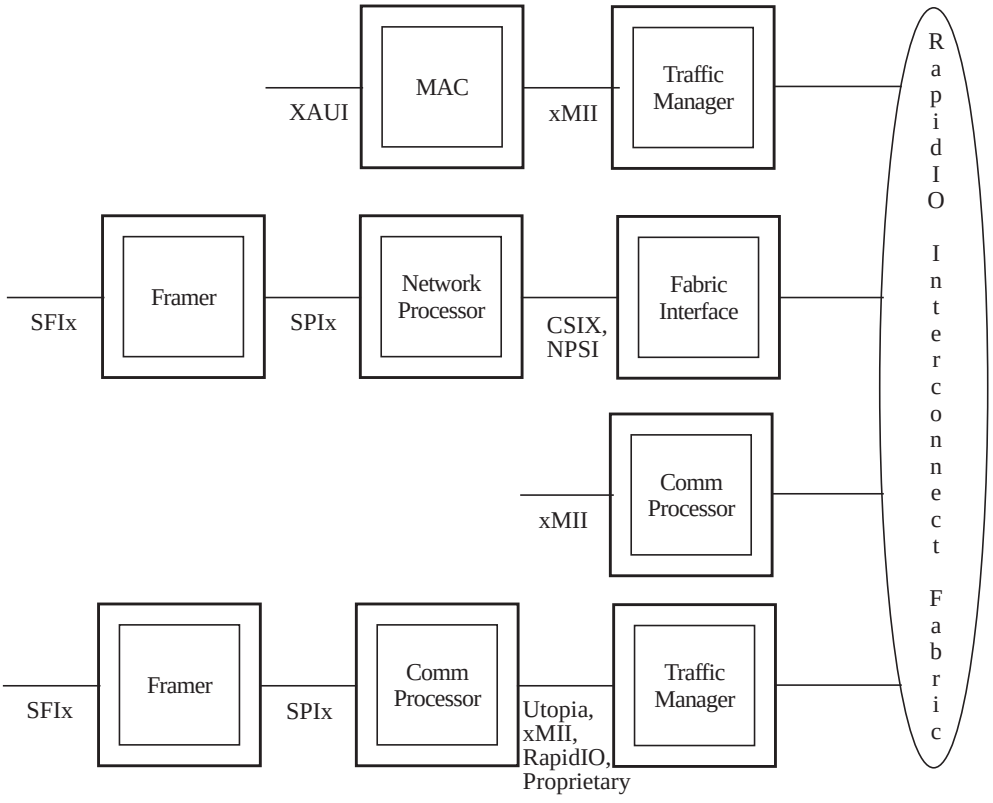

Figure 2-1. Example of a RapidIO-Based Networking System

Data “streams” represent logical connections between an ingress port and an egress port. A connection spans the transfer of multiple PDUs. The transfer of PDUs may be separated by discrete intervals of time, based on the arrival of data at the ingress. Transfer between an ingress process and an egress process is unidirectional. An I/O device may be bi-directional, containing both an ingress process and an egress process. These processes are usually completely independent consisting of separate streams in each direction.

A given ingress may service hundreds, thousands, even millions of streams at any given time depending on how specifically a PDU is classified. Traffic may be lumped into a single stream, or classified by user and application to form millions of data streams.

Data streaming transactions differ from most other RapidIO transactions in two ways: they must accommodate larger variably sized data transfers, and the transactions are not acknowledged with a response packet. The *data streaming logical layer* is intended to support data from a variety of hardware and processing devices. These devices have a variety of different interfaces, protocols, and degrees of sophistication. This specification is intended to enable these kinds of devices to exist on the RapidIO interconnect.

2.3 Traffic Streams

A stream identifier identifies independent streams of traffic between the end producer (for example, a web server) and end consumer (for example, a home personal computer) of the encapsulated data. Stream identifiers vary with protocol and may include multiple fields from the various networking layers included in the protocol. A unit of data that contains a discrete identifier is called a Protocol Data Unit, or PDU. A PDU may or may not have an ordering relationship with another PDU being transmitted between that same producer and consumer, depending upon the higher layer protocol being carried. A traffic stream is a series of PDUs that have an ordering relationship between each other. A PDU has no ordering relationship with a PDU from different producers and consumers pairs.

The data streaming logical layer uses a **virtual stream identifier** (VSID) to allow multiple end to end traffic streams of PDUs to be uniquely identified and managed concurrently within the RapidIO system. Creation of a VSID is done by performing a protocol specific classification process on a PDU. The complexity of the classification process is directly proportional to the sophistication of the system as required by the application. The VSID allows the traffic to be reassociated with an appropriate application at the egress without having to perform a second protocol-specific classification. A VSID is comprised of fields from the data streaming protocol: **source** and **destination ID** from the underlying packet transport fabric, **class of service**, and **streamID**.

2.4 Operation Ordering

A transaction request flow is defined as an ordered sequence of request transactions comprising a specific PDU from a given source ID to a given destination ID. Each packet in a transaction request flow has the same source identifier and the same destination identifier. All traffic streams are mapped onto transaction request flows. These flows may also be shared with other RapidIO logical layers transactions, and therefore the relationship between streams, traffic classes, virtual queues, and all RapidIO transaction request flows are implementation specific.

There may be multiple transaction request flows between a given source ID and destination ID pair. When multiple flows exist between a source ID and destination ID pair, the flows are distinguished by a flow indicator referred to as a "flowID", introduced in the *RapidIO Part 1: Input/Output Logical Specification*. RapidIO allows multiple transaction request flows between any source ID and destination ID pair. Any number of transaction request flows may exist between the two IDs. The flowID represents the lowest level of traffic management in a RapidIO system as that is the construct mapped directly on to the switch fabric itself.

The transaction request flows between each source and destination ID pair may be allocated to different virtual channels in the underlying fabric and may also be prioritized within a channel. The flows are labeled and identified alphabetically as in the other logical layer specifications, and the channels labeled and identified numerically with channel then priority, starting with 0 as first channel or lowest priority, then 1 as second channel or next lowest priority, etc. For example, flowID 0A is channel 0 flow A, flowID 1C is channel 1 flow C, flowID 3E is channel 3 flow E, and so on. This flow information provides class of service information when mapped by the application to the switch fabric.

Allocation of transaction request flows to virtual channels and the relative priority within each channel is application dependent. A special case is a single virtual channel application which must follow the same prioritization of flows and labeling as the other logical layers (flowID A, flowID B, flowID C, etc.). The channel label (0) is dropped. This channel may include traffic from the other logical layers.

At the link level, when multiple transaction request flows within the same virtual channel exist between a given connected source and destination ID pair, transactions of a higher priority flow may pass transactions of a lower priority flow, but transactions of a lower priority flow may not pass transactions of a higher priority flow. There are no ordering rules for flows in different channels. A traffic stream being transmitted between a source and a destination ID pair must utilize the same flowID value so that the ordering of the traffic stream is maintained. As a class of service indicator, the physical channel ID is used by the underlying RapidIO fabric to determine how to treat a packet with respect to other packets with respect to priority and ordering. It is expected that in a mixed control and data plane application that both I/O logical and data streaming transaction request flows will exist in a RapidIO system simultaneously, possibly between the same ID pairs.

To support transaction request flows, all devices that support the RapidIO data streaming logical specification shall comply as applicable with the following Fabric Delivering Ordering and End point Completion Ordering rules. Note that these rules are very similar and complementary to the rules specified in *RapidIO Part 1: Input/Output Logical Specification*.

Fabric Delivery Ordering Rules

1. **Transactions within a transaction request flow (same source identifier, same destination identifier, same flowID, same PDU) shall be delivered to the logical layer of the destination in the same order that they were issued by the logical layer of the source.**
2. **Request transactions that have the same source (same source identifier) and the same destination (same destination identifier) within the same virtual channel but with different flowIDs shall be delivered to the logical layer of the destination as follows.**
   - **– A transaction of a higher priority transaction request flow that was issued by the logical layer of the source before a transaction of a lower priority transaction request flow shall be delivered to the logical layer of the destination before the lower priority transaction.**
   - **– A transaction of a higher priority transaction request flow that was issued by the logical layer of the source after a transaction of a lower priority transaction request flow may be delivered to the logical layer of the destination before the lower priority transaction.**
3. **Request transactions that have different sources (different source identifiers) or different destinations (different destination identifiers) or different virtual channels are unordered with respect to each other.**

End point Completion Ordering Rules

1. **Request transactions in a transaction request flow shall be completed at the logical layer of the destination in the same order that the transactions were delivered to the logical layer of the destination.**

It may be necessary to impose additional rules in order to provide for inter-operability with other interface standards or programming models. However, such additional rules are beyond the scope of this specification.

2.5 Class of Service and Virtual Queues

Data streaming systems may support thousands, even millions of active data streams. These streams are eventually interleaved onto the single underlying packet transport fabric. The process for deciding which streams may share common resources is sometimes referred to as virtual queueing. To facilitate virtual queueing at the ingress and/or egress of the fabric, and to provide for more sophisticated management of traffic streams, the data streaming logical layer provides a class of service (cos) identifier. The cos field exists to provide a common semantic as to how the traffic stream is to be treated. The relationship between the ingress/egress cos and the end to end flowID assigned to the traffic stream is implementation specific.

At the ingress to the fabric, thousands of streams may be combined into fewer virtual output queues (VoQs) using just the **destination ID** and the **class of service** portions of the VSID as shown in Figure 2-2. The cos field defined by this specification is comprised of one byte. The number of bits utilized by a particular device depends upon the number of data buffering structures implemented, but are always from the most significant bit of the cos field to the least significant bit. For example, a device with two buffering structures (or "bins") maps a packet to a bin using bit 0, a device with four bins maps a packet to a bin using bits 0 and 1, and so on.

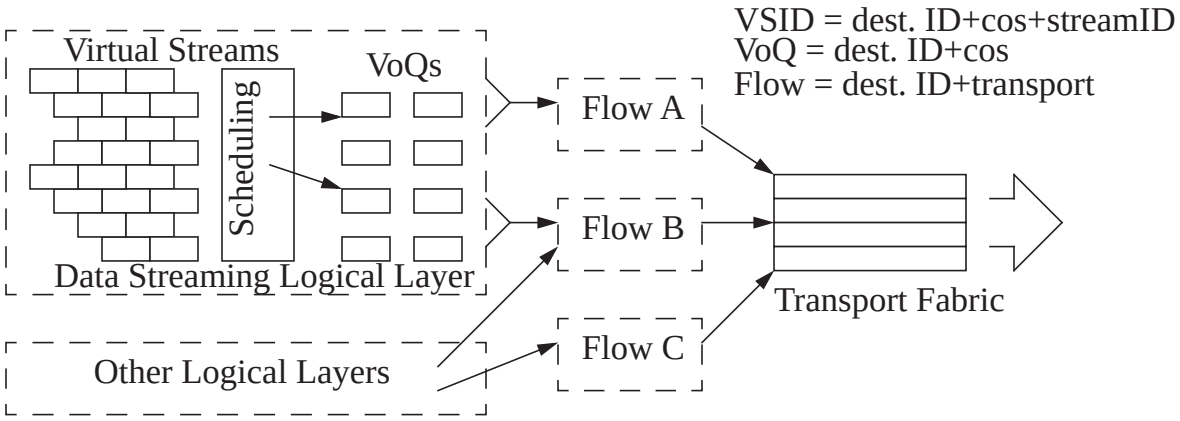

Figure 2-2. Mapping Virtual Streams at the System Ingress

As shown in Figure 2-2, as the virtual output queues are mapped on to the flowIDs and then on to the underlying packet transport fabric, they may be intermingled with other logical layer transactions. The use of the transport fabric must account for the needs of the total environment and is application and implementation specific. End points designed to support a wide variety of applications for data streaming should offer some flexibility in how virtual queues are mapped down on to the transport fabric in the implementation.

A reverse process (virtual input queueing) may or may not occur at the destination. If there is a critical resource needed to process traffic on egress from the fabric, the system designer may choose to fan the traffic back out into virtual queues. This allows the fabric egress processing to re-prioritize utilization of the critical resource.

This is illustrated in Figure 2-3.

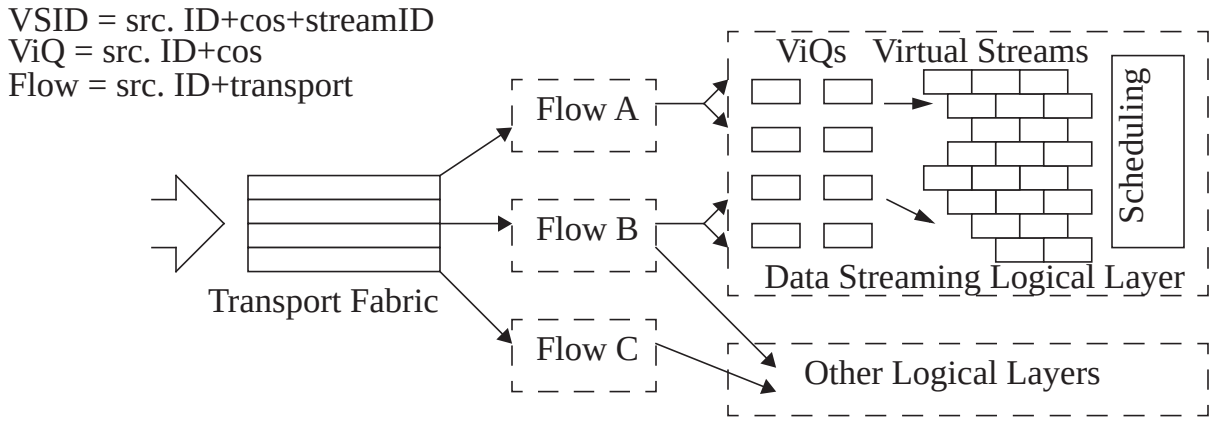

Figure 2-3. Mapping Virtual Streams at the System Egress

A switch device may choose to utilize the information carried in the cos field by acting as a "virtual" end point, removing the traffic streams from the underlying packet transport fabric, reassembling the individual PDUs, and fanning the streams back out into some larger number of queues. It then re-injects the traffic streams back into the underlying transport fabric re-ordering the traffic using the cos. This permits intervening devices to participate in the overall assurance of quality of service in the system.

2.6 End-to-end Traffic Management

Other RapidIO specifications provide for traffic management at the physical layer, and for flows at the logical layer. The traffic management for Data Streaming allows endpoints to coordinate traffic flows between class based queues and stream based queues. The protocol includes a hierarchy of methods and field specifiers to be adaptable to a wide variety of queueing and management designs. The use of Data Streaming's traffic management is optional, and may be inter-operable only between endpoints with similar capabilities. Endpoints that support the same traffic management capabilities will be able to exchange traffic management packets related to those capabilities. The implementation specific nature of the algorithms which determine traffic management in different endpoints may lead to unexpected system behavior.

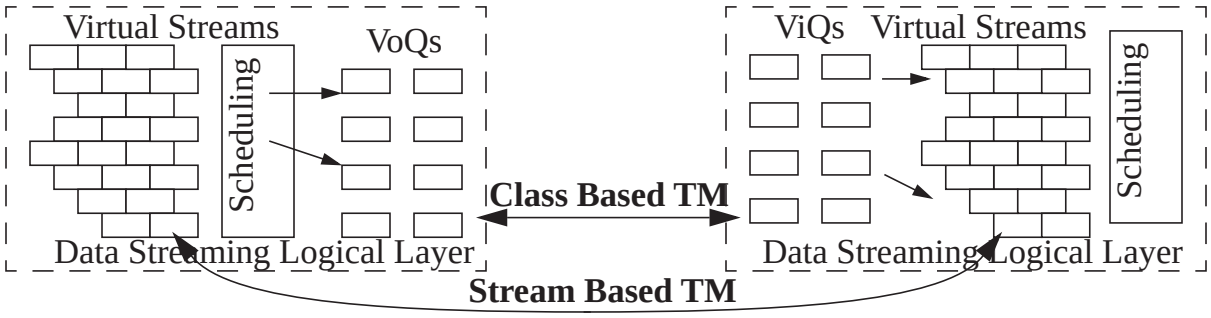

Figure 2-4. Class Based and Stream Base Traffic Management

2.7 Deadlock Considerations

A deadlock can occur if a dependency loop exists. A dependency loop is a situation where a loop of buffering devices is formed, in which forward progress at each device is dependent upon progress at the next device. If no device in the loop can make progress then the system is deadlocked.

The data streaming logical specification does not have any dependency loops since the defined operations do not require responses. However, a real RapidIO system is required to support the I/O logical maintenance operation, and will very likely require the use of other logical operations for control functions. Support for these other logical operations may have significant deadlock considerations for processing element and system designs.

Blank page

Chapter 3 Operation Descriptions

3.1 Introduction

This chapter describes the RapidIO data streaming protocol. The field encodings and packet formats are described in Chapter 4, “Packet Format Descriptions.”

Data path data movement through a machine has requirements that are significantly different than those for control path and traditional DMA functions. Many times this data is encapsulated data, which also many times contains further encapsulated data. For example, the data moving through the system may be encapsulated Ethernet packets, which may in turn be encapsulating TCP/IP packets.

This style of data movement is typically not address-based as with DMA type I/O, and consequently follows a queue based message passing paradigm. Data path data movement also has much more complex requirements in the area of class (or quality) of service than control path communications, and generally requires managing a number of queues at the egress of the system. There is also a need to be able to identify and manage many thousands of data traffic streams that pass through a RapidIO based data path system. The data being passed through the RapidIO system may not be directly generated or consumed by the device connected to the RapidIO portion of the machine, but instead by a distant end user, such as a personal computer attached to a LAN. This necessitates the addition of a new protocol to the RapidIO logical layers, the data streaming protocol.

The RapidIO data streaming protocol uses request transactions through the interconnect fabric as with other RapidIO operation protocols. Since many data movement protocols guarantee data delivery in an upper layer protocol, the generation of responses indicating completion are not needed. Such upper layer protocols may also allow data to be discarded if necessary, for example, under error or fabric congestion conditions.

3.2 Data Streaming Protocol

This section describes the RapidIO data streaming protocol.

3.2.1 Data Streaming Operation

A data stream represents a logical connection between a source and a destination pair. A stream may consist of multiple transactions and requires the allocation of

resources at both the source and the destination. This may be done in advance of any data transfer, or in response to receiving a new transaction. Since streams are virtual constructs between source and destination pairs, they may be reused for different data transfers at any time as long as the source and destination pair are both synchronized as to the stream usage.

A data streaming operation consists of individual data streaming transactions, as shown in Figure 3-1. A series of transactions is used to send PDUs between two end points. The data streaming protocol is completely independent of the PDU's native protocol.

Data streaming transactions do not receive responses, so there is no notification to the sender when the transaction has completed at the destination.

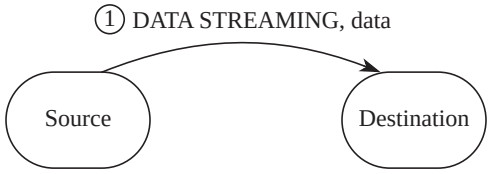

Figure 3-1. Data Streaming Operation

3.2.2 Virtual Streams

A stream is represented by a unique virtual stream identifier, or VSID. This identifier represents the handling of all PDUs within the stream for the duration of a PDU's transit of the RapidIO fabric. The identifier is created by performing some form of protocol specific classification of the PDU. The classification can be as complex or as simple as the application warrants. The VSID allows this protocol specific classification to take place one time at the ingress to the fabric. After that, the handling of the PDU is protocol independent.

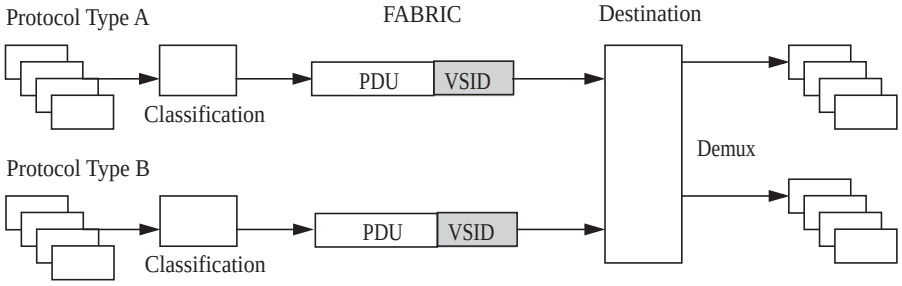

Figure 3-2. Virtual Streams

The VSID is used at the destination to "reclassify" the PDU. This sorts the data back into contexts that can now be protocol specific again. This virtual addressing model eliminates the need for the source and the destination to align the use of buffers and other resources. Therefore, the VSID can be used to carry a wide variety of information about a stream through the system, such as the protocol being encapsulated, de-multiplexing exit port IDs instructions, very fine grained buffer

management, etc., as required for a specific application.

The VSID is a “key” comprised of multiple fields. These fields are the source/destination ID, cos, and streamID.
From the source’s viewpoint: **destination ID+cos+streamID** represents a unique stream.
From the destination’s viewpoint: **source ID+cos+streamID** represents a unique stream.

By using the complete key, each source and destination pair is free to allocate the use of these fields independently. Some examples of how the VSID may be applied in a system are described in Appendix A, “VSID Usage Examples,” on page 55.

NOTE:VSIDs

Destinations are permitted to define their use of Virtual Stream IDs to pre-associate certain kinds of traffic with certain end processes. Sources shall be able to label a stream with any VSID necessary to inter-operate with the largest number of possible destination implementations.

3.2.3 PDU Sequences Within Streams

As described earlier, a traffic stream may consist of a sequence of related PDUs that have ordering requirements between each other. A stream of PDUs is transmitted one PDU at a time to preserve the required ordering. PDUs that do not have an ordering relationship may be separated into different streams or may be interleaved in common streams. A stream is identified by the interconnect fabric by the combination of the destination ID and either the cos field or the flowID, depending upon the complexity of the fabric, as described in “Section 2.5, Class of Service and Virtual Queues” on page 19.

Only one PDU from any given stream will be transmitted at a time at the source, but fabric conditions may result in multiple PDUs in transit. The fabric must guarantee that delivery of PDUs (and segments of PDUs as described below) remain in order. A fabric may load balance traffic through multiple paths on a stream by stream basis.

3.2.4 Segments within a PDU

The basic mechanism of segmentation defines a general methodology to provide for larger PDUs than are accommodated by the standard 256 byte limit on a RapidIO data payload. The standard industry term for this function is “Segmentation and Reassembly”, or SAR. A PDU that is to be transmitted from the initial producer to the final consumer is broken up (segmented) into a series of blocks of data. The consumer “reassembles” that data back into the original PDU. The maximum size of a PDU that a particular destination can accept is specified in a CAR (see Chapter 5, “Data Streaming Registers”). The system must be configured in accordance with

these limitations.

The block size used for the segmentation process is specified by the Maximum Transmission Unit, or MTU, parameter. The MTU is defined in Chapter 5, “Data Streaming Registers”. The MTU is a system-wide parameter agreed to by all processing elements participating in the SAR process. By managing the MTU size for the system, the variability in latency for the system can be controlled.

A data streaming transaction is also referred to as a segment. The transmission of a PDU for any given stream may result in one or more transactions (segments). A typical sequence is made up of three types of transactions, a start segment, some number of continuation segments, and an end segment. Start segments and continuation segments are always filled to the MTU size. End segments are variable in size containing the remainder of the PDU. If a PDU is equal to or less than the MTU size, it is carried in a single segment. A single segment may also be variable in size, matching the PDU payload. Since flowIDs and the cos are assigned on a PDU basis, all segments of a PDU must also have that same flowID and cos assignments.

A start segment contains the necessary fields to identify the VSID and “open” a *segmentation context*. The segmentation context for a stream is defined as the combination of the source ID and the flowID, and is used by a receiver to reassociate the segments of a particular PDU. Using **source ID+physical channel ID** allows each source and destination pair to have one PDU for each physical channel ID that is explicitly supported by the system interleaved in the fabric at any one point in time. Note that for a destination device that can be a multicast target and/or supports multiple destination device IDs, the **destination ID** for a PDU must also be included as part of the segmentation context in order to prevent possible PDU corruption at the destination device. The VSID is used when opening a segmentation process at the destination to associate the PDU with its stream since the continuation and end segments do not carry that information. After the receipt of the end segment, the segmentation context is “closed” (the sending processing element has an analogous definition for open and closed). The stream and PDU associated with a segmentation context is not permitted to change during the time that the context is open.

Since there may be a large number of PDU sources and concurrent contexts per source, the amount of context state that a destination may have to handle can potentially get very large. The number of contexts that can be supported by a particular destination end point is specified in a CAR (see Chapter 5, “Data Streaming Registers”). These segmentation contexts must be allocated to sources by system software.

For efficiency, information as to which block of the PDU is contained in a specific packet is not included in the header. This requires that the transmitter issue the sequence starting with the first block of the PDU and proceeding sequentially through the PDU, and requires the underlying transport fabric to deliver the sequence to the data streaming logical layer in the issued order.

Figure 3-3 shows a 24 byte PDU that is to be segmented for transmission, with an eight byte MTU (note that an eight byte MTU is not permitted in this specification; it is used to simplify the illustration). Since the PDU is divisible by the size specified as the MTU, all data payloads are exactly that size and no padding is necessary. The sender takes byte 0 (the first byte of the PDU) through byte 7 as the data payload to transmit in the start segment. The second data payload consists of bytes 8 through 15, which is transmitted in a continuation segment. The last data payload consists of bytes 16 through 23, which is transmitted in the end segment. Since the data payloads are required to be delivered to the receiver's data management hardware in order of transmission, the receiver can correctly reassemble the original PDU when all three packets have arrived.

To guarantee the packet ordering, all packets making up an individual PDU and all PDUs in a stream must be in the same transaction request flow, as described in "Section 2.4, Operation Ordering" on page 17.

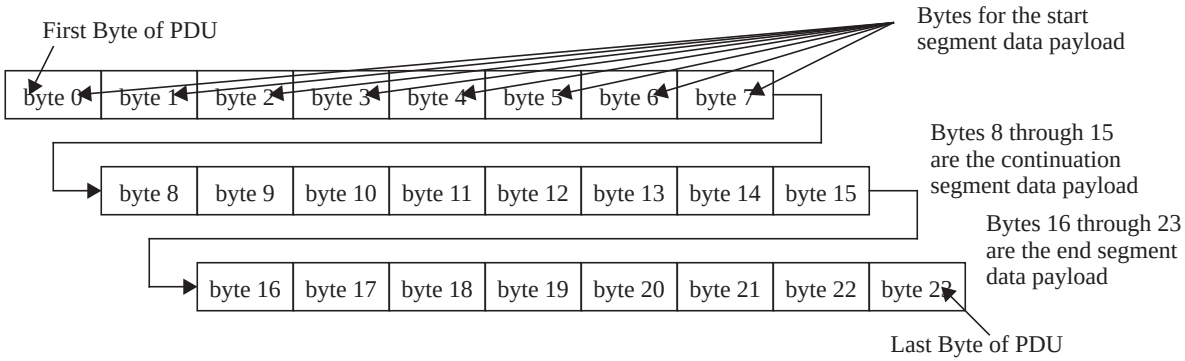

Figure 3-3. PDU Segmentation and Reassembly Example 1

Figure 3-4 shows an example of a similar situation, except that this time the PDU is 21 bytes. In this case, the end segment has a data payload that is less than the specified MTU, and also has a pad byte to round out the data payload to be a multiple of half-words. A bit in the end segment (the "P" bit) indicates the presence of the pad byte. An additional bit (the "O" bit) indicates that the data payload has an odd number of half-words and is therefore oddly aligned. The number of half-words in the data payload as well as the presence of a pad byte can be determined from a PDU length field contained in the end segment header.

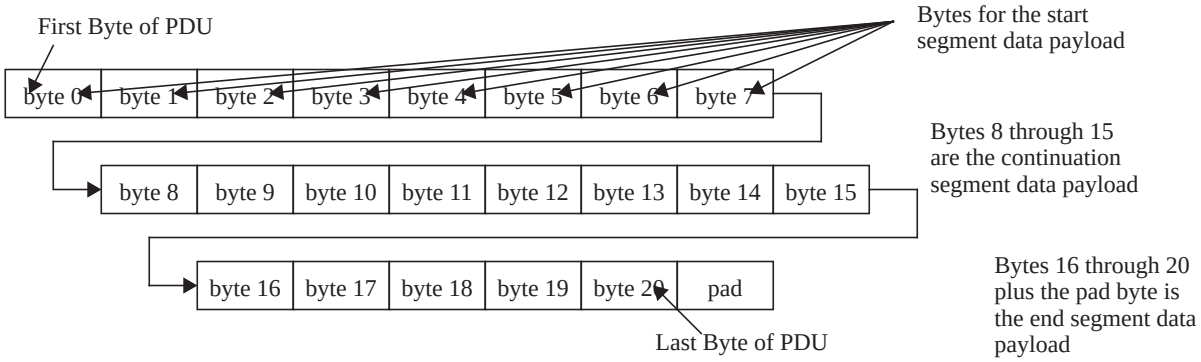

Figure 3-4. PDU Segmentation and Reassembly Example 2

3.2.5 Rules for Segmentation and Reassembly

Segmentation (source)

1. In order to limit implementation complexity due to possible PDU ordering issues, the following conditions must be met:
   – No more than one PDU from a given stream shall be segmented at a time
   – No more than one PDU from a given RapidIO flow shall be segmented at a time
   – PDUs associated with different RapidIO flows may be segmented concurrently
2. Segments are filled with bytes from the PDU in order as shown in Figure 3-3 and Figure 3-4.
3. The first segment is marked as start segment (see section 4).
4. The start segment is filled to the end of the PDU data or to the MTU size.
5. If the end of the PDU data is encountered, the start segment then re-marked as a single segment.
6. If the start segment reaches MTU size (and there is remaining PDU data), the start segment is encapsulated, and a continuation segment is opened.
7. Continuation segments are filled to MTU size from the PDU data, in order.
8. When the end of PDU data is encountered, the segment is marked as the end segment. The end segment data payload size may be less than or equal to the MTU size.
9. If the source wishes to abort a PDU transmission, it sends an end segment with no data payload and with the length field set to zero.

Reassembly (destination)

1. Upon receiving a segment with a start bit, the reassembly unit opens a "context" containing the virtual stream ID and associates it with the segmentation context consisting of the source ID, the destination ID, and the physical channel ID. (The destination ID is only required for devices supporting multicast and/or multiple destination IDs.)
2. The reassembly process transfers the entire payload into the reassembly buffer in order. The amount of data transferred is counted for comparison to the length field.
3. If the packet is a single segment, the amount of payload data must be equal to or less than the MTU size or the PDU is defective.
4. If the packet is a start segment and the payload data does not match the MTU size the PDU is defective.
5. Reassembly continues with continuation packets. All continuation packets must match the MTU size or the PDU is defective. All data transferred to the reassembly buffer is counted.
6. An end segment terminates the reassembly process. An end segment may be received immediately after a start segment. The data payload size must be less than or equal to the MTU size or the PDU is defective. The data from the end segment is transferred according to the data payload size and counted.
7. Once all the data has been reassembled, the length (provided by the end segment packet header) is checked against the received data count. A mismatch indicates a lost continuation segment and the PDU is defective.
8. Receiving a continuation or end segment on a closed context indicates a lost start segment and the PDU is defective.
9. Receiving a start or single segment on an open context indicates a lost end segment and the PDU is defective. The existing context is closed, and the new context is opened.

In all cases, a defective PDU results in discarding the entire PDU. The method used for reporting the discard event is beyond the scope of this specification. It may be desirable for a destination to have a timeout as part of the lost packet detection mechanism, but the definition and time interval are also outside of the scope of this specification.

3.3 Class of Service and Traffic Streams

A virtual stream ID is partitioned into three pieces as previously discussed: port (identified by the source/destination ID), class (the cos field), and the stream identifier (the streamID field). These fields form a specific hierarchy for transitioning packets from highly individualized streams to coarser groupings of traffic. At the fabric ingress, egress, and potentially at interim points (where competition for resources may occur) the traffic may be resegregated and queued by class. In the packet transport fabric, switching is done by destination ID and the

mapped flowID, as described in Section 2.4. The full class of service identifier (CoS ID) is a subset of the VSID. It consists of the source/destination ID (or ingress/egress port) plus the cos field.
Ingress queueing should be based on: **destination ID+cos**
Egress queueing should be based on: **source ID+cos**
as shown in Figure 3-5.

Including the source or destination ID in the CoS ID allows the class of service to be specific to the source and destination pairing.

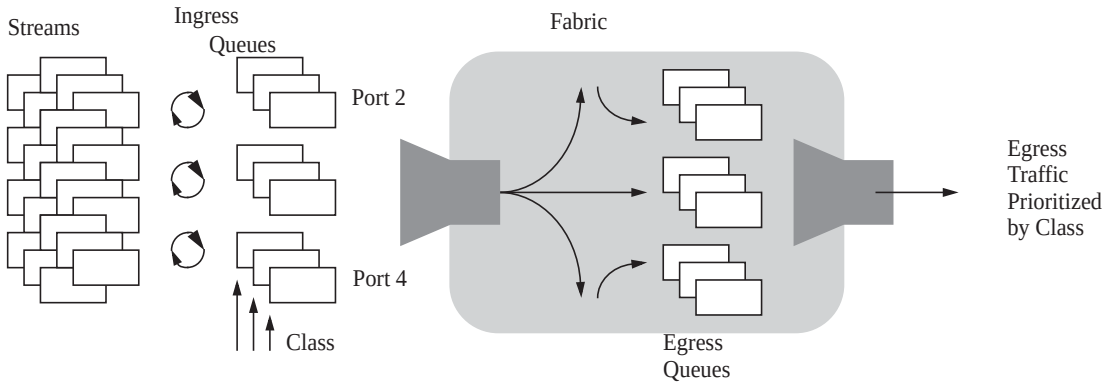

Figure 3-5. Traffic Sorting Based on CoS ID

The cos field shall be used beginning with the MSB (bit 0) using the necessary number of bits for the number of classes supported.

Bit 0 - 2 Classes of Service

Bits 0, 1 - 4 Classes of Service

Bits 0, 1, 2 - 8 Classes of Service supported

etc.

3.4 Traffic Management

Data Streaming Traffic Management (TM) supports end-to-end flow control through multiple mechanisms. The protocol includes On/Off, Rate based and Credit based schemes. There is also room in the protocol for user defined operations. Traffic Management uses the extended packet format for Type 9 (see Chapter 4 for packet definitions). The traffic management format makes up a message that contains:

<VSID><Wildcard+Mask> <Message XType><Parameter 1><Parameter 2>

The VSID (from the standard type 9 header) plus the wildcard and mask fields in the extended packet header forms the operand (the queue or queues) for the traffic management message. The message type and type specific parameters form the operation.

3.4.1 Traffic Management Operand

All Data Streaming traffic management protocols use a common mechanism for queue designation. The message overloads the data streaming VSID fields to define the stream to which to apply the message. In addition, the extended header format includes modifiers, in the form of a wild card and a mask to expand the scope of the message beyond a single stream. The message can apply to:

- A single stream, identified by <Dest><cos><streamID>
- A group of streams in a class identified by <Dest><cos>
- A group of classes, identified by <Dest><cos><mask>
- All classes for a given port, identified by <Dest>

Note that the operand is always based on a VSID that is the target (intended destination) of the data. The egress is not responsible for knowing the mapping of queues taking place at the ingress.

Example: Endpoint 21 sends <cos 3> XOFF to endpoint 6
Endpoint 6 would stop all streams to VSID: DestID 21, class 3.

3.4.2 On/Off Traffic Management

The messages supported are:

Egress to Ingress:
<Q> XOFF (where <Q> is any operand described in 3.4.1)
<Q> XON

Ingress to Egress:
<Q> Q_STATUS <Level> (see Section 4.3.2 for message and parameter formats)

Basic On/Off traffic management consists of egress to ingress messages that direct the operand <Q> be stopped (XOFF) or started (XON). The ingress may signal the need for service with the Q_STATUS message, with *level* indicating the relative fullness of the queue.

Ingress devices may admit traffic based on any ingress specific scheduling algorithm. This message does not modify the algorithm except to suspend/resume traffic flow.

When traffic management is enabled and set to *TM Type 0* (basic), the messages shall be supported by the egress and honored by the ingress according to the rules in Section 3.2.5.

3.4.3 Rate Base Traffic Management

The messages supported are:

From Egress to Ingress

<Q> XON (where <Q> is any operand described in Section 3.4.1)
<Q> XOFF
<Q> INCREASE <Amount>
<Q> DECREASE <Amount>

From Ingress to Egress
<Q> Q_STATUS <Level> (see Section 4.3.3 for message and parameter formats)

XON and XOFF messages are as defined in Section 3.4.2.

Rate based flow control is a *relative* control protocol. <Amount> is a ratio relative to the current rate of traffic flow (see Chapter 4 for field definitions). The ingress is pre-configured with a specific traffic scheduling algorithm. The egress uses the INCREASE / DECREASE mechanism to modify the ingress' scheduling process, usually based on the egress' ability to move the traffic to its next destination.

DECREASE<0> is a special message meaning MAINTAIN current rate, and INCREASE<max> is a special message to DOUBLE the current rate.

The ingress may use the Q_STATUS message to indicate the status of its queues, with *level* indicating the relative fullness of the queue, allowing a closed loop decision process. For example, if the ingress sends successive messages indicating rising queue levels, the egress may choose to increase the rate at which the ingress has permission to admit traffic from that queue.

When the traffic management mode is set to *TM Type 1* (rate), the XON, XOFF messages are included to carry forward the basic operation (the type 0 messages no longer need to be used).

3.4.4 Credit Based Traffic Management

Credit based traffic management permits the egress to control traffic on a PDU by PDU basis. In this mode PDUs are only transmitted when an ingress is given an allocation of credits for a specific queue (identified by the operand <Q>). Traffic flow stops when the allocation of credits reaches zero.

The basic message formats are:

Egress to Ingress:

<Q> XON (where <Q> is any operand described in Section 3.4.1)
<Q> XOFF
<Q> ALLOCATE <AU> <Credits>

Ingress to Egress:
<Q> CREDIT STATUS <AU> <Credits>
<Q> Q_STATUS <Level> (see Section 4.3.4 for message and parameter formats)

Allocate is used by an egress endpoint to tell an ingress endpoint that it has <some number of> credits available for use. The credits are assigned to an *allocation unit*

<AU>. The allocation unit allows blocks of resources to be grouped, permitting coarser management, and requiring less precision in the synchronization between an ingress and the egress.

When the credits in an allocation unit are consumed, the ingress begins to use credits in the next allocation unit. The egress assigns credits in allocation units on a rotating basis. Allocation units have local scope. The allocation unit value has local scope to the <Q> designation.

The protocol allows or pipelining up to 16 allocation units. Endpoints do not have to use 16 allocation units. The number used is a function of how far in advance credits need to be issued. The minimum implementation is 2 to ping-pong buffer groups and keep some credit available at the ingress. The egress may issue any number of allocation units (starting with 0) and rolling over at whatever limit it supports. It is up to the egress to be sure an AU is unused before reusing that number.

<Q> ALLOCATE <AU> <0> is used to retire an allocation. It may be used by an egress endpoint to free a block of buffering for a number of reasons. It can be used to pause a transmission by forcing the only credits to zero. It can be used to clean up memory allocation by forcing an ingress onto the next allocation unit.

Credit Status: is sent by the ingress to delimit the use of allocation units, and to indicate status to trigger new allocations. <Q> CREDIT STATUS <AU> <nn>, where nn is the number of initial credits for an AU, is sent ahead of the first packet to delimit the beginning of the use of that AU.

<Q> CREDIT STATUS <AU> <00> is always sent after the last packet resulting in the number of credits going to 0 for that AU. The egress can close out that chunk of buffering, even if its notion of the number of PDUs received does not agree with the ingress (a PDU has been lost somewhere). <Q> CREDIT STATUS <AU> <00> is also sent in response to an <Q> ALLOCATE <AU> <00> acknowledging that the ingress will send no more PDUs for that allocation, delimiting any PDUs in the pipeline.

<Q> CREDIT STATUS <AU> <xx>, where xx is some number of remaining credits, may be sent by an ingress to indicate a low level of credit allocation as a trigger to request more credits. The egress may also track the number of incoming PDUs from an ingress keeping a local credit balance. Either or both mechanisms can be used to sequence allocations. Note that the egress is not required to keep a specific credit balance, it can allocate and retire allocation units using the delimiting messages.

Queue Status: provides a means to indicate the overall status of a specific queue. The source can use this message to get attention for a queue that has become active, needing credits (transitioning from empty). It can be used to indicate a queue that has gone empty, allowing the destination to deallocate the remaining credits and retire the AU. It can also be used to indicate that the current rate of credits being issued is not keeping up with incoming traffic.

XOFF and **XON** are used to pause / resume transmission without changing the state of credit allocation.

In the example shown in Figure 3-6, the ingress initiates activity with queue status, indicating traffic available. The egress responds with an allocation of 16 credits.

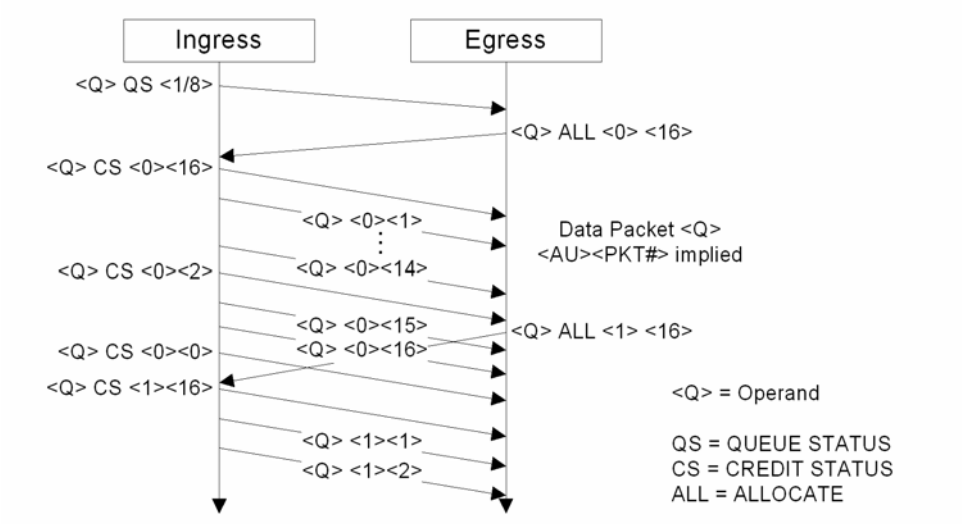

Figure 3-6. Typical Credit Based Flow Control Example

At a level of 2 credits remaining, the ingress sends a credit status of <2> indicating a low level of credits. It proceeds with sending the last two PDUs.

The egress responds to the request for more credits with an allocation of 16 additional PDUs. The ingress indicates the last PDU has been sent on the first allocation, and that new PDUs are being sent on the new allocation with the two sequential credit status messages. This is a “typical” scenario.

In the scenario shown in Figure 3-7, the egress moves the traffic from AU 0 to AU 1 by first allocating more credits with AU 1, then terminating the credits for AU 0.

The ingress responds by sending credit status of 0 for AU 0, and credit status to indicate the beginning of AU 1.

There are many ways to use this protocol. An ingress may use the credit status <0> to asynchronously relinquish credits for a stream if an activity timer indicates that there’s been no new traffic for a period of time (or the egress may do the same).

The size of the allocations can be varied using the Queue Status message, allocating larger blocks for fuller queues. An ingress endpoint might have prescribed thresholds for sending queue status message.

The ingress might also have adjustable thresholds for sending Credit Status messages to adjust for pipeline delays, or algorithmically move that threshold up or down based on gaps in allocation.

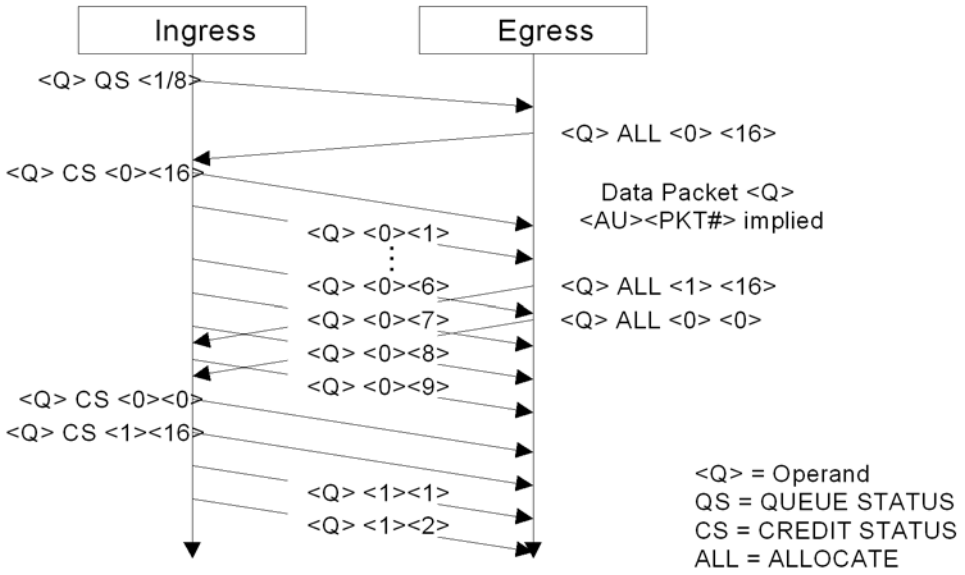

Figure 3-7. Credit Based Protocol Example

3.4.5 Rules for Traffic Management

Supported Functionality

1. Supporting traffic management is entirely optional.
2. Any endpoint advertising support for traffic management shall support the basic mode of operation.
3. An endpoint may or may not support rate or credit based modes of operation.
4. Any supported modes shall be supported fully, incorporating all defined message formats (use of user defined fields is optional).

Therefor the valid combinations for supporting traffic management are:

- Basic Only
- Basic + Rate
- Basic + Credit
- Basic + Rate + Credit

Protocol Rules

1. All TM transactions use the VSID in the type 9 header, so by definition, all TM transactions occur in the same flows as the data. When wild cards are used, the don’t care fields shall be set to values that put the message in the same flow as the affected data.

2. Any rules used by the ingress to associate traffic with a specific VSID shall be used in reverse to associate the VSID in the message from the egress with the at least one queue. Beyond that, any internal hierarchies of queues and relationships to different messages are up to the implementation.
3. Endpoints may implement <Amount>, or <Level> with less precision than described. Receiving endpoints shall support the full range of values by rounding to the desired precision.
4. An ingress shall not overrule a TM directive from the egress. The ingress may discard traffic should the egress not adequately permit that traffic onto the fabric. Any discard algorithm is implementation specific.

Error Handling

5. There are no requirements for timers. An ingress may "insist" by sending repeated Q_STATUS messages.
6. Lost messages are recovered by sending duplicates. Endpoints shall recognize duplicates as such and not behave inappropriately.

   An XOFF to a <Q> already in the off state is ignored.

   An XON to a <Q> already in the on state is ignored.

   A lost Rate DECREASE message results in the egress sending a second DECREASE with a larger requested decrease amount.

   A lost Rate INCREASE message results in the ingress issuing more urgent Q_STATUS messages and an the egress issuing additional INCREASE messages.
7. The exception to rule 6 is the sequencing of allocation units with the CREDIT STATUS message:

   If the <Q> CREDIT STATUS <AU> <0> message is lost, the allocation unit will be assumed closed when the <Q> CREDIT STATUS <AU+1> <N> is received opening operation on the next allocation unit.

   If the <Q> CREDIT STATUS <AU+1> <N> message is lost, the next allocation unit is assumed to be opened when the next PDU for that stream is received.

   Also, as an exception to rule 6, ALLOCATE messages are never duplicated. If an allocation unit message is lost, the egress may recover it with a <Q> ALLOCATE <AU><0> message, insuring it is de-allocated before reusing it.

Chapter 4 Packet Format Descriptions

4.1 Introduction

This chapter contains the definition of the data streaming packet format.

4.2 Type 9 Packet Format (Data-Streaming Class)

The type 9 packet format is the DATA STREAMING transaction format. Type 9 packets always have a data payload, unless terminating the PDU. Unlike other RapidIO logical specifications, the data payload length is defined as a multiple of half-words rather than double-words. A pad bit allows a sender to transmit an odd number of bytes in a packet. An odd bit indicates that the data payload has an odd number of half-words. This bit makes it possible for the destination to determine the end of a data payload if packet padding is done by the underlying transport. An extended header bit allows future expansion of the functionality of the type 9 packet format.

Definitions and encodings of fields specific to type 9 packets are provided in Table 4-1.

Table 4-1. Specific Field Definitions and Encodings for Type 9 Packets

Field	Definition
cos	class of service - This field defines the class of service to be applied by the destination end point (and possibly intervening switch processing elements) to the specified traffic stream.
S	Start - If set, this packet is the first segment of a new PDU that is being transmitted. The new PDU is identified by the combination of the source of the packet and the flowID.
E	End - If set, this packet is the last segment of a PDU that is being transmitted. Both S and E set indicates that the PDU is fully contained in a single packet.
rsrv	Reserved - Assigned to logic 0s by the sender, ignored by the receiver
xh	Extended header - There is an extended header on this packet. The extended header is used for traffic management.
O	Odd - If set, the data payload has an odd number of half-words
P	Pad - If set, a pad byte was used to pad to a half-word boundary

Table 4-1. Specific Field Definitions and Encodings for Type 9 Packets (Continued)

Field	Definition
streamID	traffic stream identifier - This is an end to end (producer to consumer) traffic stream identifier.
length	PDU length - This is the length in bytes of the segmented PDU. 0x0000 - 64kbytes 0x0001 - 1 byte 0x0002 - 2 bytes 0x0003 - 3 bytes ... 0xFFFF - 64kbytes - 1

Table 4-1 details the O and P bit combinations.

Table 4-2. Specific Field Definitions and Encodings for Type 9 Packets

O bit	P bit	Definition
0b0	0b0	Even number of half-words and no pad byte
0b0	0b1	Even number of half-words and a pad byte
0b1	0b0	Odd number of half-words and no pad byte
0b1	0b1	Odd number of half-words and a pad byte

There are three type 9 packet headers, determined by the value of the Start and End bits, which determine if the header is a Start/Single header, a Continuation header, or an End header. The following set of figures shows examples of type 9 packets. Field sizes are specified in bits.

Figure 4-1 is an example of a Single Segment type 9 packet with all of its fields. The data payload size may or may not match the MTU size, so *n* and *m* are determined by the size of the PDU itself. In this example, the data payload is un-padded and there are an even number of half-words. The value 0b1001 in Figure 4-1 specifies that the packet format is of type 9. This is the only type 9 packet that has the xh field.

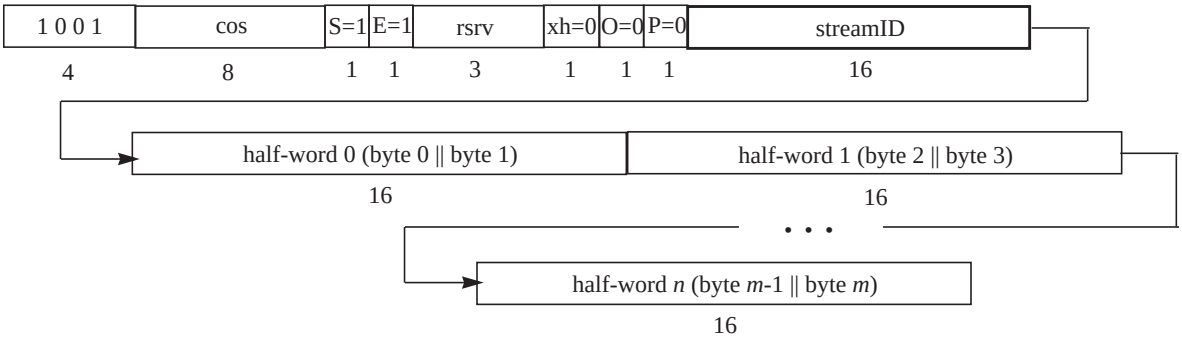

Figure 4-1. Single Segment Type 9 Packet Bit Stream Format Example

Figure 4-2 is an example of a Start Segment type 9 packet with all of its fields. The data payload that matches the MTU, so *n* and *m* are determined by the MTU size. The value 0b1001 in Figure 4-2 specifies that the packet format is of type 9.

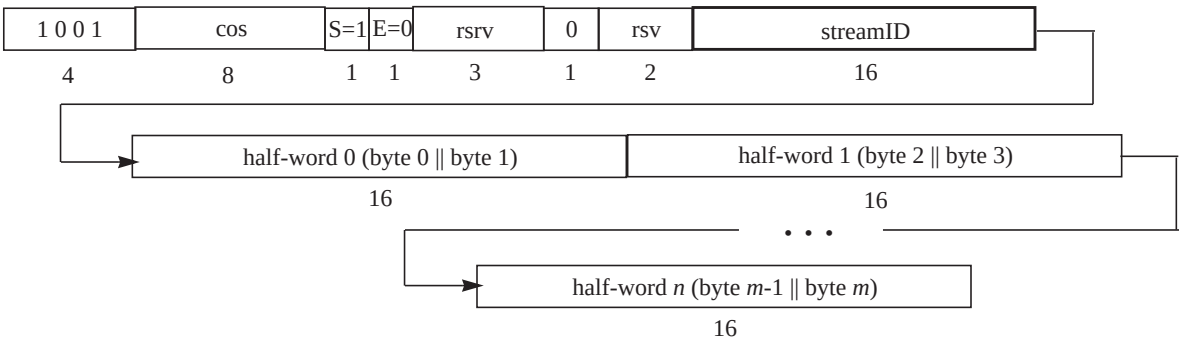

Figure 4-2. Start Segment Type 9 Packet Bit Stream Format Example

Figure 4-3 is an example of a Continuation Segment type 9 packet with all of its fields. The size of the data payload must match the MTU size. The half-words (and correspondingly, bytes) are contiguous in the manner shown in the preceding examples. The value 0b1001 in Figure 4-3 specifies that the packet format is of type 9.

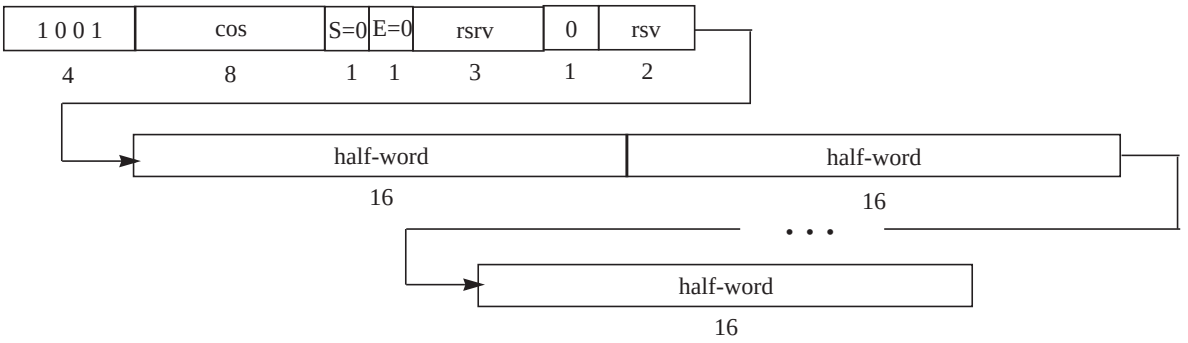

Figure 4-3. Continuation Segment Type 9 Packet Bit Stream Format Example

Figure 4-4 is an example of an End Segment type 9 packet with all of its fields. The size of the data payload is determined by the remainder of the size of the PDU (the length field) divided by the size of the MTU. For convenience at the destination, the O and P bits are used as they are for a single segment. In this example, the data payload size does not match the PDU size, has a pad byte, and is an odd number of half-words. The half-words (and correspondingly, bytes) are contiguous in the manner shown in the preceding examples. A length value of 0 and no data payload can be used to force the PDU to be discarded. The value 0b1001 in Figure 4-4 specifies that the packet format is of type 9.

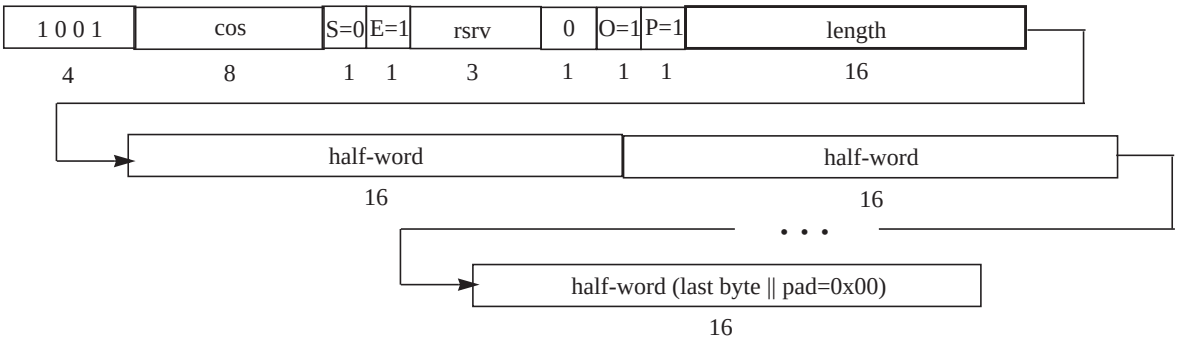

Figure 4-4. End Segment Type 9 Packet Bit Stream Format

4.3 Type 9 Extended Packet Format

The type 9 extended packet format for traffic management between two data streaming endpoints is shown below.

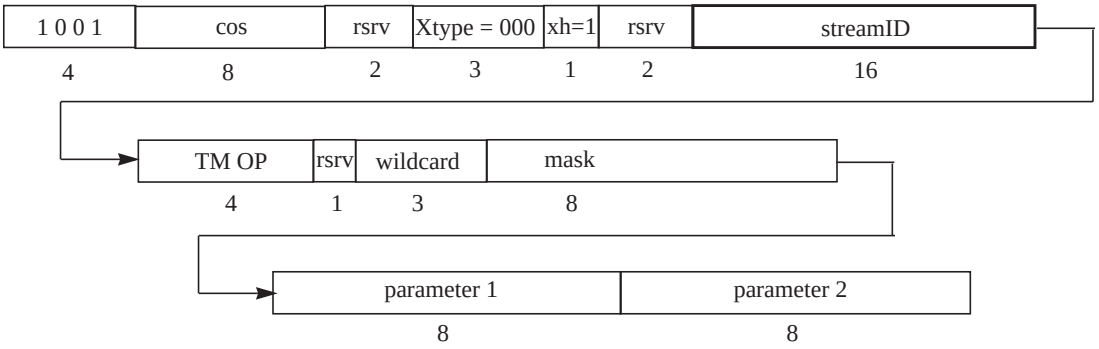

Figure 4-5. Traffic Management Bit Stream Format

The extended type 9 packet is identified first by the XH bit equal to 1. The Class of Service field and the StreamID are included from the data packet format. The segmentation bits, Start and End are not used, or are the Odd and Pad fields. Those are left 0s. The 3 bit reserved field, is defined for the extended packet format as defining the type of extended packet. Type 0 is a traffic management packet.

The TM packet adds a fixed 4 bytes to the packet as shown above and defined below:

Table 4-3. Extended Header Fields

Type 9 Fields	Encoding	Definition
cos	—	Class Of Service: This field defines the class of service assigned to the stream that is being managed with this message. See Section 4.3.1.
rsrv	—	Reserved
xtype	0b000	Traffic Management Packet
	0b001 - 0b111	Reserved
xh	0b1	Extended Header: There is an extended header on this packet. Currently the extended header is only used for stream management messages. It is always assigned to 0b1 for type 9 extended packets.
streamID	—	Traffic Stream Identifier: This is an end to end (producer to consumer) traffic stream identifier that is being managed with this message. See Section 4.3.1.
TM OP	0b0000	Basic Stream Management Message: message specifying base level XON/XOFF functionality. This message flows from one endpoint to another. See Section 4.3.2.
	0b0001	Rate Control Stream Management Message: Rate flow control messages to support the rate control protocol. See Section 4.3.3.
	0b0010	Credit Control Stream Management Message: Credit Control messages to support the credit based flow control protocol. Section 4.3.4
	0b0011	Application Defined Stream Management Message: A message from one end point to another end point. The use of this message is application defined.
	0b0100-1111	Reserved
wildcard	0bnxx	VSID dest wildcard: If this bit is set, the message applies to all destinations. If clear, the message applies to the specified destination. See Section 4.3.1.
	0bxnx	VSID class wildcard: If this bit is set the message applies to all classes. If clear, the message applies to the specified class or classes as specified by the mask bits. See Section 4.3.1.
	0bxxn	VSID stream wildcard: If this bit is set the message applies to all streams. If clear, the message applies to the specified stream. See Section 4.3.1.
mask	—	Class Mask: Used to mask portions of the class of service (COS) field to allow groups of classes to be included in the message. The mask is left justified to identify specific class bits as don't cares: Mask / Class 0b00000000 0bnnnnnnnn 256 classes — n = valid class bits 0b00000001 0bnnnnnnnx 128 classes — x = don't cares 0b00000011 0bnnnnnnxx 64 classes 0b00000111 0bnnnnnxxx 32 classes 0b00001111 0bnnnnxxxx 16 classes 0b00011111 0bnnnxxxxx 8 classes 0b00111111 0bnnxxxxxx 4 classes 0b01111111 0bnxxxxxxx 2 classes 0b11111111 0bxxxxxxxx 1 class
Parameter1	—	Parameter1: Argument specific to the TM message operation. See below
Parameter2	—	Parameter2: Argument specific to the TM message operation. See below

4.3.1 TM Operand

The operand for the specific TM operation is defined by the following fields:

<Source or Destination ID> <CoS> <StreamID> + <wild cards> + <Mask>

The follow operands are valid:

Specific Stream:
<DestID><CoS><StreamID> + <wc=000> + <m = 0x00>

All Streams in a specific class:
<DestID><CoS>< xx > + <wc = 001> + <m = 0x00>
where StreamID is a don't care

All Streams in a group of classes:
<DestID><CoS>< xx > + <wc = 001> + <m = 0xnn>
where mask is one of the non-zero values identified in table 4-3

All Streams and Classes (all traffic) to a specific destination
<DestID>< xx >< xx > + <wc = 011> + <m = xx>
where CoS, StreamID and mask are don't cares

All traffic from this source
< xx >< xx >< xx > + <wc = 111> + <m = xx>

No other combinations of these fields is permitted.

4.3.2 Basic Traffic Management

Basic traffic management message formats are as follows:

Table 4-4. Basic Traffic Management Message Formats

TM OP	Parameter 1	Parameter 2	Definition
0b0000	0b0000 0000	0b0000 0000	XOFF: Transmit off
		0b1111 1111	XON: Transmit on.
		0b0000 0001- 0b1111 1110	User/application defined
	0b0000 0011	0b0000 0000 - 0b1111 1111	Q_Status: Source queue is n/255 full (where n is parameter 2). 0 = empty, 0xFF = full

All other parameter 1 values are reserved.

The implementer may overload the parameter 2 field (Parameter1 = 0) for other messages. Only 0x00 shall be assured to be XOFF, and 0xFF shall assured to be XON.

4.3.3 Rate Based Traffic Management

Rate based traffic management messages are as follows:

Table 4-5. Rate Based Traffic Management Message Formats

TM OP	Parameter 1	Parameter 2	Definition
0b0001	0b0000 0000	0b0000 0000	XOFF: Transmit off
		0b1111 1111	XON: Transmit On. Start transmitting at the rate prior to receiving the XOFF
		0b0000 0001 - 0b1111 1110	User Defined: May be overloaded with any implementation specific message.
	0b0000 0Y01	0b0000 0000	Maintain_Rate: Maintain the current rate
		0b0000 0001	REDUCE: Reduce the current rate to = CurrentRate x (1-1/256)
		0b0000 0010	REDUCE: Reduce the current rate to = CurrentRate x (1-2/256)
		0b0000 0011- 0b1111 1111	REDUCE: Reduce the current rate to = CurrentRate x (1-n/256) (where n is parameter 2)
	0b0000 0Y10	0b0000 0001- 0b1111 1110	INCREASE: Increase the current rate to = CurrentRate x (1+ n/256) (additive increase where n is parameter 2)
		0b1111 1111	DOUBLE: Double the current rate.
	0b0000 0011	0b0000 0001- 0b1111 1111	Q_Status: Source queue is n/255 full (where n is parameter 2). 0x00 = Empty, 0xFF = Full

For ’Y’ = 0 (Param 1 = 0x01, 0x02) the messages apply to average rate.

For ’Y’ = 1 (Param 1 = 0x05, 0x06) the messages apply to peak rate.

All other parameter 1 values are reserved.

The implementer may overload the parameter 2 field (Parameter1=0) for other messages. Only 0x00 shall be assured to be XOFF, and 0xFF shall assured to be XON.

Notes on reducing precision:

REDUCE values should be rounded UP to the nearest precision.

Example: for n/16 precision - 0b0000 0011 would be rounded to 0b0001 xxxx

INCREASE values should be rounded DOWN to the nearest precision.

Example for n/32 precision 0b1100 1110 would be rounded to 0b1100 1xxx

Q_STATUS values should be rounded UP to the nearest precision.

Whatever the minimum precision the ingress uses for rate scheduling, a decrease must never reduce the rate completely to zero. The egress must use XOFF to stop the flow completely.

4.3.4 Credit Based Traffic Management

The message formats for credit based traffic management are as follows:

Table 4-6. Credit Based Traffic Management Message Formats

TM OP	Parameter 1	Parameter 2	Description
0b0010	0b00000000	0b00000000	XOFF: Transmit off
		0b11111111	XON: Transmit On. Start transmitting at the rate prior to receiving the XOFF
		0b0000 0001 - 0b1111 1110	User Defined: May be overloaded with any implementation specific message.
	0b0001 nnnn	0b0000 0000 - 0b1111 1111	Allocate: nnnn - Allocation Unit Parameter 2 - number of credits
	0b0010 nnnn	0b0000 0000 - 0b1111 1111	Credit Status: nnnn - allocation Unit Parameter 2 - number of Credits
	0b0011 0000	0b0000 0000 - 0b1111 1111	Queue Status: Source Queue is n/255 full (where n is parameter 2) 0 = Empty; 0xFF = Full

All other parameter 1 values are reserved.

The implementer may overload the parameter 2 field (Param1=0) for other messages. Only 0x00 shall be assured to be XOFF, and 0xFF shall assured to be XON.

Chapter 5 Data Streaming Registers

5.1 Introduction

This chapter describes the visible register set that allows an external processing element to determine the capabilities, configuration, and status of a processing element using this logical specification. This chapter only describes registers or register bits defined by this specification. Refer to the other RapidIO logical, transport, physical, and extension specifications of interest to determine a complete list of registers and bit definitions. All registers are 32 bits and aligned to a 32 bit boundary.

5.2 Register Summary

Table 5-1 shows the register map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using *RapidIO Interconnect Specification Part 1: Input/Output Logical Specification* maintenance operations. Any register offsets not defined are considered reserved for this specification unless otherwise stated. Other registers required for a processing element are defined in other applicable RapidIO specifications and by the requirements of the specific device and are beyond the scope of this specification. Read and write accesses to reserved register offsets shall terminate normally and not cause an error condition in the target device. Writes to CAR (read-only) space shall terminate normally and not cause an error condition in the target device.

Register bits defined as reserved are considered reserved for this specification only. Bits that are reserved in this specification may be defined in another RapidIO specification.

Table 5-1. Data Streaming Register Map

Configuration Space Byte Offset	Register Name
0x0-14	Reserved
0x18	Source Operations CAR
0x1C	Destination Operations CAR
0x20–38	Reserved
0x3C	Data Streaming Information CAR

Table 5-1. Data Streaming Register Map (Continued)

Configuration Space Byte Offset	Register Name
0x40–44	Reserved
0x48	Data Streaming Logical Layer Control CSR
0x4C–FC	Reserved
0x100–FFFC	Extended Features Space
0x10000–FFFFFC	Implementation-defined Space

5.3 Reserved Register, Bit and Bit Field Value Behavior

Table 5-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 5-2. Configuration Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x0–3C	Capability Register Space (CAR Space - this space is read-only)	Reserved bit	read - ignore returned value[1]	read - return logic 0
			write -	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write -	write - ignored
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x40–FC	Command and Status Register Space (CSR Space)	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value[2]	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored

Table 5-2. Configuration Space Reserved Access Behavior (Continued)

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x100–FFFC	Extended Features Space	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x10000–FFFFFC	Implementation-defined Space	Reserved bit and register	All behavior implementation-defined	

[1]Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.

[2]All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

When a writable bit field is set to a reserved value, device behavior is implementation specific.

5.4 Additions to Existing Registers

The following bits are added to the Logical/Transport Layer Error Detect CSR (see *RapidIO Interconnect Specification Part 8: Error Management Extensions Specification*).

Table 5-3. Bit Settings for Logical/Transport Layer Error Detect CSR

Bit	Name	Reset Value	Description
10	Missing data streaming context	0b0	A continuation or end data streaming segment was received for a closed or non-existent segmentation context (end point device only)
11	Open existing data streaming context	0b0	A start or single data streaming segment was received for an already open segmentation context (end point device only)
12	Long data streaming segment	0b0	Received a data streaming segment with a payload size greater than the MTU (end point device only)
13	Short data streaming segment	0b0	Received a start or continuation data streaming segment with a payload size less than the MTU (end point device only)
14	Data streaming PDU length error	0b0	The length of a reassembled PDU differs from the PDU length carried in the end data streaming segment packet header (end point device only)

The following bits are added to the Logical/Transport Layer Error Enable CSR (see *RapidIO Interconnect Specification Part 8: Error Management Extensions Specification*).

Table 5-4. Bit Settings for Logical/Transport Layer Error Enable CSR

Bit	Name	Reset Value	Description
10	Missing data streaming context error enable	0b0	Enable reporting of a continuation or end data streaming segment received for a closed or non-existent segmentation context. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)
11	Open existing data streaming context error enable	0b0	Enable reporting of a start or single data streaming segment received for an already open segmentation context. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)
12	Long data streaming segment error enable	0b0	Enable reporting of a any data streaming segment received with a payload size greater then the MTU. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)

Table 5-4. Bit Settings for Logical/Transport Layer Error Enable CSR

Bit	Name	Reset Value	Description
13	Short data streaming segment error enable	0b0	Enable reporting of a start or continuation data streaming segment received with a payload size less that the MTU. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)
14	Data streaming PDU length error enable	0b0	Enable reporting of a reassembled PDU length that differs from the PDU length carried in the end data streaming segment packet header. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)

5.5 Capability Registers (CARs)

Every processing element shall contain a set of registers that allows an external processing element to determine its capabilities through maintenance read operations. All registers are 32 bits wide and are organized and accessed in 32 bit (4 byte) quantities, although some processing elements may optionally allow larger accesses. CARs are read-only. Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 and Word 0 respectively the most significant bit and word.

5.5.1 Source Operations CAR (Configuration Space Offset 0x18)

This register defines the set of RapidIO data streaming logical operations that can be issued by this processing element; see Table 5-5. It is assumed that a processing element can generate I/O logical maintenance read and write requests if it is required to access CARs and CSRs in other processing elements. The Source Operations CAR is applicable for end point devices only. RapidIO switches shall be able to route any packet.

Table 5-5. Bit Settings for Source Operations CAR

Bit	Field Name	Description
0–11	—	Reserved
12	Data streaming traffic management	PE can support data streaming traffic management
13	Data streaming	PE can support a data streaming operation
14-15	Implementation defined	Defined by the device implementation
16-29	—	Reserved
30–31	Implementation defined	Defined by the device implementation

5.5.2 Destination Operations CAR (Configuration Space Offset 0x1C)

This register defines the set of RapidIO data streaming operations that can be supported by this processing element; see Table 5-6. It is required that all processing elements can respond to maintenance read and write requests in order to access these registers. The Destination Operations CAR is applicable for end point devices only. RapidIO switches shall be able to route any packet.

Table 5-6. Bit Settings for Destination Operations CAR

Bit	Field Name	Description
0-11	—	Reserved
12	Data streaming traffic management	PE can support data streaming traffic management
13	Data streaming	PE can support a data streaming operation
14-15	Implementation defined	Defined by the device implementation
16-29	—	Reserved
30-31	Implementation defined	Defined by the device implementation

5.5.3 Data Streaming Information CAR (Configuration Space Offset 0x3C)

This register defines the data streaming capabilities of a processing element. It is required for destination end point devices.

Table 5-7. Bit Settings for Data Streaming Information CAR

Bit	Field Name	Description
0–15	MaxPDU	Maximum PDU - The maximum PDU size in bytes supported by the destination end point 0x0000 - 64kbytes 0x0001 - 1 byte 0x0002 - 2 bytes ... 0xFFFF - 64kbytes - 1
16–31	SegSupport	Segmentation Support - The number of segmentation contexts supported by the destination end point 0x0000 - 64k segmentation contexts 0x0001 - 1 segmentation context 0x0002 - 2 segmentation contexts ... 0xFFFF - 64k - 1 segmentation contexts

5.6 Command and Status Registers (CSRs)

A processing element shall contain a set of command and status registers (CSRs) that allows an external processing element to control and determine the status of its internal hardware. All registers are 32 bits wide and are organized and accessed in the same way as the CARs. Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

5.6.1 Data Streaming Logical Layer Control CSR (Configuration Space Offset 0x48)

The Data Streaming Logical Layer Control CSR is used for general command and status information for the logical interface.

Table 5-8. Bit Settings for Data Streaming Logical Layer Control CSR

Bit	Field Name	Description
0-3	TM Types Supported (read only)	Bit 0 = 1, Basic Type Supported Bit 1 = 1, Rate Type Supported Bit 2 = 1, Credit Type Supported Bit 3 = Reserved Valid Combinations: 0b1000, 0b1100, 0b1010, 0b1110. All others invalid
4 - 7	TM Mode	Traffic Management Mode of operation 0b0000 = TM Disabled 0b0001 = Basic 0b0010 = Rate 0b0011 = Credit 0b0100 = Credit + Rate 0b0101 - 0b0111 = Reserved 0b1000 - 0b1111 = allowed for user defined modes
8 - 23		Reserved
24-31	MTU	Maximum Transmission Unit - controls the data payload size for segments of an encapsulated PDU. Only single segment PDUs and end segments are permitted to have a data payload that is less than this value. The MTU can be specified in increments of 4 bytes. Support for the entire range is required. 0b0000_0000 - reserved ... 0b0000_0111 - reserved 0b0000_1000 - 32 byte block size 0b0000_1001 - 36 byte block size 0b0000_1010 - 40 byte block size ... 0b0100_0000 - 256 byte block size 0b0100_0001 - Reserved ... 0b1111_1111 - Reserved All other encodings reserved

Blank page

Annex A VSID Usage Examples

A.1 Introduction

The virtual stream identification (VSID) mechanism provides multiple features condensed in a single 32 bit key. These features include:

- A mechanism to manage traffic for ingress to the fabric
- A mechanism to manage traffic in transit within the fabric
- A protocol independent tag to reclassify packets on fabric egress
- A flexible "sub-port" addressing mechanism
- Independence in buffer management

A.2 Background

The VSID is a composite of the port, class, and streamID fields as described in Section 3.2.2. The port address used in the VSID is either the destination ID or the source ID depending on which side of the fabric the packet is on. At the ingress to the fabric (source) the destination IDs are unique. At the egress from the fabric, the source IDs are unique.

By including the source/destination IDs in the VSID, these keys are unique for each source and destination pairing. This allows the other fields (class and streamID) to be set up independently without consideration of how these fields are used with any other port pairings.

The usage of the VSID can vary depending on the sophistication of the fabric and the demands of the application, from very simplistic port or queue steering to conveying significant amounts of information (requiring intensive computation) as to the content of the PDU.

A.3 Packet Classification

All PDUs require some form of classification for ingress to the fabric. Fields in the PDU specific to the protocol are examined and routing information is produced. The VSID produced is a 32 bit tag as opposed to just a port address. At the destination, this 32 bit tag can be used to re-associate the PDU with a target buffer. This can be done by direct addressing, or using a single key table lookup.

This mechanism provides a finely grained and protocol independent way to sort traffic, and a virtual mechanism for buffer pool management. Without a virtual tag, the packet would have to undergo a re-classification based on the protocol specific portion of the PDU. In multi-service platforms, this could involve numerous and elaborate processes, duplicating what was already done at the source.

The following sections illustrate in degrees of increasing complexity, the versatility of the VSID scheme.

A.3.1 Sub-port Addressing at the Destination

The simplest use of the VSID is to de-multiplex the traffic into coarse sub-ports at the destination. These may be to separate traffic by protocol, or into multiple sub-ports of the same protocol.

A.3.1.1 DSLAM application

Assume that each line card contain 128 user ports. The system could expose each of these as independent destinations to the RapidIO fabric, requiring the use of an excessively large number of destination IDs in the system, and imposing the associated cost in overhead. Alternatively the ATM traffic can be encapsulated into 128 VSIDs, one for each port. The line card would then expose a single port to the RapidIO fabric. The VSID would be used as the address to fan out the traffic on various UTOPIA busses to the user ports. This also has an advantage for fault recovery. Should a line card fail, a single port entry in routing tables in the fabric needs to be updated rather than all 128 sub-ports.

A.3.1.2 VOIP application

The VSID can be used to separate the traffic into just 2 channels, one destined for a control processor to handle control messages and one channel that goes to a network processor to be distributed to DSPs. The VSID could contain the address of the target DSP, to further off-load the network processor on distribution. The VSID could also contain the user channel within the DSP de-multiplexing the traffic even further.

A.3.2 Virtual Output Queueing - Fabric On-ramp

Applications involving larger numbers of flows can use the class field to regulate the ingress to the fabric (known as virtual output queueing). For example, the RapidIO fabric interface could contain 256 queues for 64 destination ports with 4 traffic classes. Traffic for each destination of the same class is fairly weighted. The weighting between classes can be application unique.

The traffic is kept sorted by destination. If traffic was just dumped into 4 queues, and a destination port was to fail, the traffic could head of line block the traffic to the other ports, or it would have to be discarded while the port is being recovered or

re-routed. By keeping the traffic sorted by destination at the fabric ingress, that destination can be re-routed with minimal traffic loss.

Virtual output queueing can be expanded to 2K or even 16K buffers depending on how large the fabric is, and how many different traffic classes are involved. This fabric ingress management can be a simple mechanism to add some quality of service to a system using the destination ID and the class portion of the VSID. Note that this can be done separately from the use of the streamID at the destination for de-multiplexing.

A.4 System Requirements

The use of the VSID is determined by all three elements in a system, the source, the fabric, and the destination. This section contains descriptions of some example source devices.

A.4.1 UTOPIA to RapidIO ATM bridge

The UTOPIA to RapidIO ATM bridge classifies traffic using the VPI field as the destination port, and the VCI as a sub-port address. It maps all (type 9) traffic to a single RapidIO flow, setting the class to 0 and the streamID to the VCI. The fabric switches on flows. The destination uses the streamID portion of the VSID as a hard-wired sub-port address.

A.4.2 Network processor

The network processor (NP) contains a OC-48 link aggregating traffic to and from multiple 1MB/s ports distributed on line cards. The NP classifies traffic for each user into two classes: high priority for voice (using RTP) and low priority for all others. It sets the class field to 0 or 1, the port to the proper line card, and the streamID to the desired sub-port.

A.4.3 CSIX to RapidIO interface

The CSIX packet contains the destination and class fields (the source is a preset parameter in the interface chip). The streamID is the first 16 bits of the CSIX payload. The RapidIO packet is easily constructed from this information. The fabric interface contains multiple virtual output queues, 2 per destination port. Since the CSIX to NP interface is also a segmented interface, PDUs are reassembled in the virtual queues until enough information is available to form the required MTU on the RapidIO fabric.

The fabric maps the class to a higher or lower priority flow. The destination uses the streamID to map the traffic to the correct user sub-port. Each sub-port contains two class queues to collect traffic as it is reassembled.

A.4.4 10Gb Metropolitan Area Network interface

A specialized classification processor creates the 32 bit VSID based on IP, TCP/UDP, and application information. The tag is prepended to a SPI4.2 packet. The interface to the fabric is a SPI4.2 to RapidIO bridge, which contains virtual output queues.

The destination is a processor that only supports memory and IO logical transactions. The RapidIO to processor interface bridge contains the segmentation and reassembly buffers and look up tables and associated engines that maps the VSID to a DMA buffer address (and vice-versa).

The system contains multiple of these processing cards to support address translation, encryption, or firewall processing. The source classifies traffic based on which of these applications applies. A connection is created by allocating a buffer address in the destination, and assigning a streamID. The source table is created with the search tree requirements for the protocol, and setting up the VSID result.

Destinations may use the VSID in a hard-wired method, or it may be a flexible mapping to virtual buffers. In either case, the source must be flexible to assign the VSID according the destination's needs. This is normally not an issue as the source needs to classify the packet to determine the destination anyway. The use of the VSID can be to separate the traffic by protocol, sub-port, service class, or into as many virtual queues as necessary. If the destination is managing a large number of buffers, the VSID allows the destination to use a single protocol independent key to re-map the traffic and completely abstract any buffer management.

Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book.

B

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

C

Capability registers (CARs). A set of read-only registers that allows a processing element to determine another processing element's capabilities.

Class of service (cos) a term used to describe different treatment (quality of service) for different data streams. Support for class of service is provided by a class of service field in the data streaming protocol. The class of service field is used in the virtual stream ID and in identifying a virtual queue.

Command and status registers (CSRs). A set of registers that allows a processing element to control and determine the status of another processing element's internal hardware.

D

Deadlock. A situation in which two processing elements that are sharing resources prevent each other from accessing the resources, resulting in a halt of system operation.

Destination. The termination point of a packet on the RapidIO interconnect, also referred to as a target.

Device. A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a processing element.

Direct Memory Access (**DMA**). The process of accessing memory in a device by specifying the memory address directly.

Double-word. An eight byte or 64 bit quantity, aligned on eight byte boundaries.

E

Egress - Egress is the device or node where traffic exits the system. The egress node also becomes the destination for traffic out of the RapidIO fabric. The terms egress and destination may or may not be used interchangeably when considering a single end to end connection.

End point. A processing element which is the source or destination of transactions through a RapidIO fabric.

End point device. A processing element which contains end point functionality.

Ethernet. A common local area network (LAN) technology.

External processing element. A processing element other than the processing element in question.

F

Field or Field name. A sub-unit of a register, where bits in the register are named and defined.

H

Half-word. A two byte or 16 bit quantity, aligned on two byte boundaries.

Host. A processing element responsible for exploring and initializing all or a portion of a RapidIO based system.

I

Ingress - Ingress is the device or node where traffic enters the system. The ingress node also becomes the source for traffic into the RapidIO fabric. The terms ingress and source may or may not be used interchangeably when considering a single end to end connection.

Initiator. The origin of a packet on the RapidIO interconnect, also referred to as a source.

I/O. Input-output.

O

Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.

P

Packet. A set of information transmitted between devices in a RapidIO system.

PDU. Protocol Data Unit, the OSI term for a packet.

Priority. The relative importance of a transaction or packet; in most systems a higher priority transaction or packet will be serviced or transmitted before one of lower priority.

Processing Element (**PE**). A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a device.

Processor. The logic circuitry that responds to and processes the basic instructions that drive a computer.

R

Receiver. The RapidIO interface input port on a processing element.

S

SAR. Segmentation and Reassembly, a mechanism for encapsulating a PDU within multiple packets.

Segmentation. A process by which a PDU is transferred as a series of smaller *segments*.

Segmentation Context. Information that allows a receiver to associate a particular packet with the correct PDU.

Sender. The RapidIO interface output port on a processing element.

Sequence. Sequentially ordered, uni-directional group of messages that constitute the basic unit of data delivered from one end point to another.

StreamID. A specific field in the data streaming protocol that is combined with the data stream's transaction request flow ID and the sourceID or destinationID from the underlying packet transport fabric to form the virtual stream ID.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

Transaction request flow. A sequence of transactions between two processing elements that have a required completion order at the

destination processing element. There are no ordering requirements between transaction request flows.

V

Virtual Stream ID (VSID). An identifier comprised of several fields in the protocol to identify individual data streams.

Virtual input Queue (ViQ), Virtual output Queue (VoQ). An intermediate point in the system where one or more virtual streams may be concentrated.

W

Word. A four byte or 32 bit quantity, aligned on four byte boundaries.