OpenCPI
Datagram RDMA (DG/RDMA)
Protocol Specification

Authors:
John Miller, Lyndeborough Object Technologies, LLC (john_f_miller@me.com)
James Kulp, Parera Information Services, Inc. (jek@me.com)

Revision History

Revision	Description of Change	By	Date
1.01	Creation	jmiller	2012-06-07
1.02	Revised per review by Jim Kulp	jmiller	2012-06-12
1.03	Revised per review by Jim Kulp	jmiller	2012-06-13
1.04	Revised per review by Shep Siegel	jmiller	2012-06-14
1.05	Update, fixed header alignment, added more clarity	jmiller	2012-06-14
1.06	Update after initial software implementation	jmiller	2012-06-25
1.07	Move L2 and upper layer issues to separate section to “boil down” the actual protocol	jkulp	2012-11-06

Table of Contents

1 Introduction

This document defines a protocol used to perform ***RDMA write*** transactions over a transmission layer providing unreliable ***datagram service*** that does not guarantee delivery, does not guarantee order of delivery, but does guarantee error-free delivery.

While the concept of Remote DMA is usually associated with hardware-assisted protocol implementations, this protocol does not assume that. It simply implements a model of supporting "remote write transactions" that enable a ***source*** to write data into a ***destination***'s memory at addresses in that memory".

The protocol was created as a layer between a datagram service below (e.g. link-layer Ethernet or IP/UDP) and communications services built in RDMA write transactions above. In particular, it was created to support the OpenCPI RDMA protocol.

1.1 References

This document refers to several others, but none contain normative content.

Title	Published By	Link
OpenCPI Technical Summary	OpenCPI	https://github.com/opencpi/opencpi/raw/master/doc/OpenCPI_Technical_Summary.pdf
OpenCPI RDMA Protocol	OpenCPI	https://github.com/opencpi/opencpi/raw/master/doc/OpenCPI_Technical_Summary.pdf

1.2 Terms

The following terms are defined for use in the context of this DG-RDMA specification.

- ***Endpoint:*** A source or destination for DG/RDMA traffic. A ***destination*** endpoint has an address space into which messages are written.
- ***Endpoint ID***: An identifier of an endpoint.
- ***Endpoint Address:*** A location in the address space of a ***destination*** endpoint
- ***Message:*** A unit of data (bytes) written to a destination endpoint at an address.
- ***Transaction:*** A set of messages from source to destination, delivered atomically.
- ***Frame:*** Unit of data sent as a datagram from source to destination, using the transmission layer. Frames contain zero or more messages.
- ***Acknowledgement:*** An indication that a frame has been received successfully.
- ***Sender***: The actor that transmits frames (of messages) to a destination.
- ***Receiver*** - The actor that receives frames (of messages) and acknowledges them.
- ***Transmission Layer:*** The layer below this protocol, providing datagram service.

1.3 Transmission layer assumptions

- The protocol is datagram-based but "transmission layer" agnostic. The transmission layer must provide datagram transmission services that are assumed to be unreliable, possibly out of order, but error free when delivered.

- The "transmission layer" limits the size of datagrams. Frames will be sized to fit within the transmission layer limit, but a frame may carry more than one message.

2 Overview

This DG-RDMA protocol is defined to implement the OpenCPI RDMA data plane model when the underlying transmission layer provides (only) datagram service. That model is a credit-based paradigm with Time Zero credits that are distributed at setup, and is described in detail in the OpenCPI RDMA Protocol specification. Understanding the OpenCPI RDMA Protocol is not a prerequisite, but may clarify motivations for this protocol, which is designed to help support it.

2.1 Endpoints

For this protocol, the concept of an endpoint is either end of a source-to-destination transaction consisting of a set of messages. This concept is a simplification of the complete "endpoint" model used in the higher-level OpenCPI RDMA model. In this protocol the destination endpoint has an address space, whereas the source does not: it is a RDMA write-only protocol.

2.2 Protocol Overview

A DG-RDMA transaction consists of zero or more data messages plus a single completion message. The difference between data messages and completion messages is simply that a completion message is only acted on *after* all the data messages in the transaction have been received and acted on. Also, the completion messages are fixed small size. All messages describe data bytes written to a destination endpoint address.

Each data message has a header describing the transaction it is part of and the address and length of its data. This transaction information is the same for all data messages in the transaction and includes:

- A transaction ID assigned by the sender, scoped by the source/destination pair.
- The number of data messages in the transaction
- A description of the completion message for the transaction, including its data.

This information is sufficient for the receiver to understand when a transaction has been started and when it is complete. When the receiver determines that all data messages in a transaction have been received, it acts on the completion message (i.e. it writes the completion message's data to the completion message's endpoint address).

Messages are conveyed in frames sent as datagrams from source to destination using the transmission layer. Multiple messages, possibly from different transactions, can be conveyed in each frame. Messages (including their headers) must fit in frames, and frames must fit in the datagrams supported by the transmission layer.

2.3 Protocol Reliability

It is assumed that the underlying transmission layer is unreliable and may in fact cause the destination endpoint to acquire messages out of order. Therefore the protocol must implement reliability itself. The sender may pack one or more messages into each frame and send them to the receiver. The receiver will send frames back to the sender containing acknowledgements of the frames received that contain messages. Since

endpoints may be simultaneously sending transactions (messages) to each other, frames may contain both messages in one direction *and* acknowledgements for frames in the other direction. Frames containing messages are acknowledged. Frames containing only acknowledgements are not acknowledged.

When a frame containing messages is received it is the responsibility of the receiver to send back acknowledgments. Therefore it is the responsibility of the sender to retain the ability to retransmit frames, until they are acknowledged. This makes the re-transmission of frames simple but also requires the release of resources when an acknowledgement is received. A protocol implementation may choose to either regenerate frames by remembering which messages the frame contained, or simply store frames for retransmission.

This in an implementation tradeoff between more complex state management vs. more storage. An implementation may choose to pack as many messages into a frame as possible from different transactions and thus optimize for throughput while keeping track of which transactions and messages were placed in each frame. Alternatively, and simpler, the implementation could send frames containing messages from a single transaction.

This specification does not define the timeout period that a sender should wait for an acknowledgements since this time is dependent on the transmission bandwidth, traffic patterns and the receivers work load.

3 Protocol Data Structures

All values in protocol data structures that are larger than 1 byte are little endian.

3.1 Frame Header Format

Each DG-RDMA frame contains the following header at the beginning of the frame. Note that this header is 10 bytes in length. This is due to the fact that the L2 header is most commonly 14 bytes in length so this will put any messages in the frame on an 8-byte boundary when the L2 header is itself aligned on an 8-byte boundary. When this protocol is mapped to a particular transmission layer, the position of this frame header in the underlying datagram is specified (i.e. whether there is any initial padding).

Description	Size
Destination ID: ID of destination endpoint	2 octets
Source ID: ID of source endpoint	2 octets
Frame ID: Rolling sequence frame number	2 octets
ACKStart: Starting ID of ACK sequence	2 octets
ACKCount: Total Number of ACKS	1 octets
Flags	1 octet

The destination id for this protocol is similar to a port used for UDP. It routes the frame to the associated endpoint. Most hardware implementations of DG-RDMA would likely only support a single endpoint per interface. However, a software implementation on a GPP could have an endpoint per process, which requires the destination ID to be used to route frames to the appropriate destination. Both the destination and source ID's are globally unique within a communication cluster and are numbers that are greater than 0 and assigned sequentially to the endpoints in the cluster. Therefore they can be used by the implementation as a convenient index value.

The frame ID must be a monotonically increasing value with roll over. The frame sequence IDs are contiguous values per source/destination pair, so the sender is responsible for maintaining a unique sequence value for each destination.

The Flags byte has the upper 7 bits reserved and they must be set to 0. The least significant bit is used to indicate if the frame contains at least 1 message. If the least significant bit is set to 0 it indicates that frame contains no messages.

The frame header may contain acknowledgements. If the ACKCount field is not 0 then the ACKStart value is used to determine what frames are being acknowledged.

Example ACK payload for 1 frame:

ACKCount - 1

ACKStart - 6577

ACK 1 frame with ID 6577.

Example ACK payload for 3 frames:

ACKCount - 3

ACKStart - 0xffffffff

ACK 3 frames with ID 0xffffffff to rolled over IDs 0 and 1.

Note that the sender always owns the Frame ID that is sent to the receiver. Even if a frame carries nothing but an acknowledgement, its Frame ID is generated by the sender and has no association with the IDs of frames that are being acknowledged in the frame header.

3.2 Message Header Format

Description	Size
Transaction ID: rolling ID that is scoped by source and destination IDs	4 octets
Completion Address: Address in the destination endpoint where the completion data value will be written	4 octets
Completion Data Value: Completion value to be written	4 octets
Number of data messages in this transaction: ***can be zero***	2 octets
Fields above here are the same in all messages in the same transaction	
Message sequence: UNUSED WILL BE REMOVED	2 octets
Data Address: Endpoint address where message data will be written	4 octets
Data Length: Length of message data following this header in bytes	2 octets
Message Type: UNUSED WILL BE REMOVED	1 octet
Trailing message: Boolean, non-zero if there is another msg *after* this one	1 octet

Each message header is optionally followed by data. It is the responsibility of the sender to pad out the message data to ensure that the next message header is aligned on an 8-byte boundary within the frame.

The Transaction IDs are scoped to the source and destination IDs from the frame header, and the sender should assign transaction IDs sequentially for each destination destination. Since endpoint IDs are assigned globally using small numbers, receiver implementations can use a simple two-level table lookup to find transaction state.

3.2.1 Error Handling

This specification requires retransmission of frames that are not acknowledged after a certain time has passed. This also means that receivers may receive the same frame more than once. This can occur when a sender is too aggressive with timeouts. This

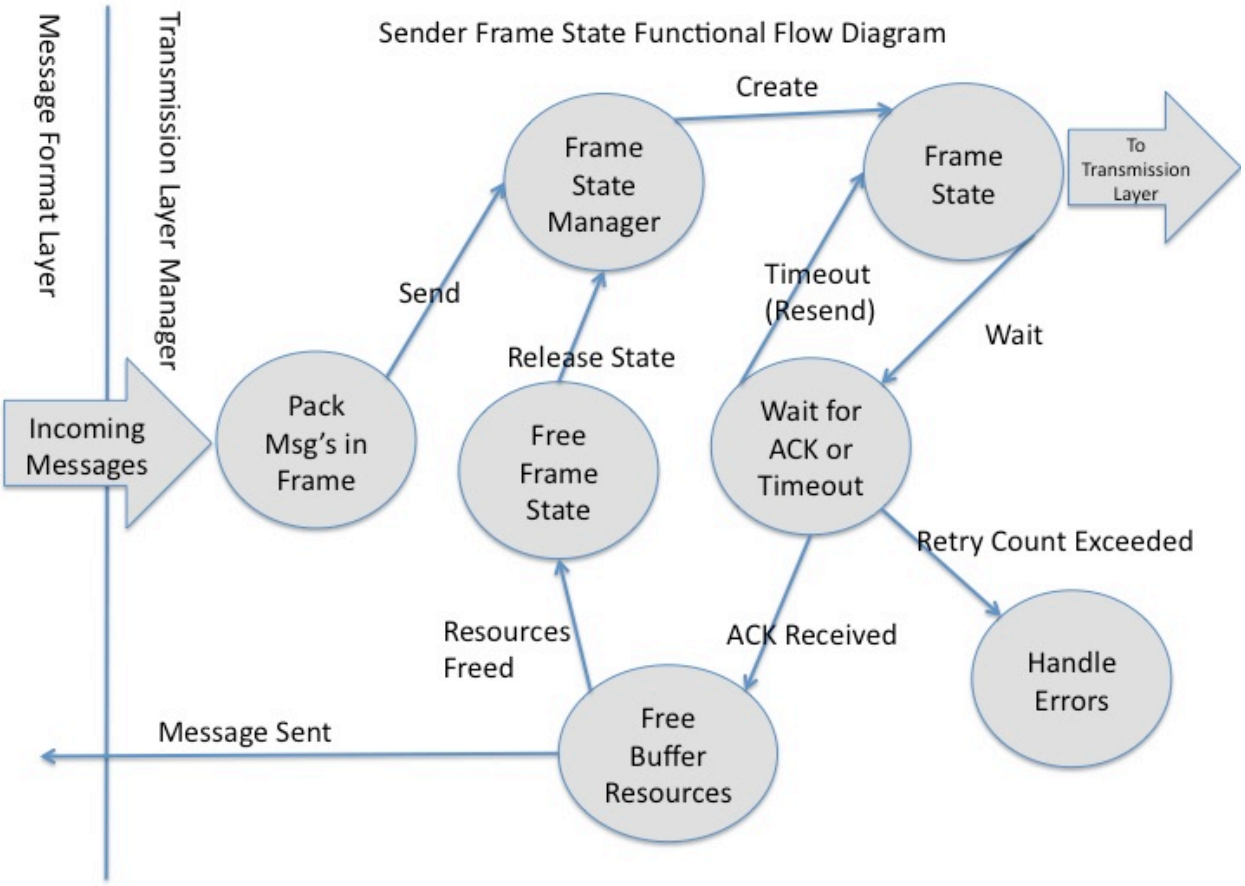
case occurs when a frame has been sent, the sender times out and sends the frame again because the acknowledgement from the original frame is sent too late. It is the responsibility of the receiver to detect this condition and ignore the duplicate frame. This same scenario can occur if the frame containing the acknowledgement gets lost. Thus the receiver is still responsible for sending an acknowledgement for the duplicate frame.

This protocol does not defined how many times or how often frames should be retransmitted in the absence of acknowledgements. This is an implementation decision. Both sides symmetrically persist in frame retransmission until acknowledgments are received. When a sender gives up (after some implementation-defined limit), the connection is considered unusable, and must be reinitialized by higher levels of software.

4 Protocol Functional Flow Diagrams

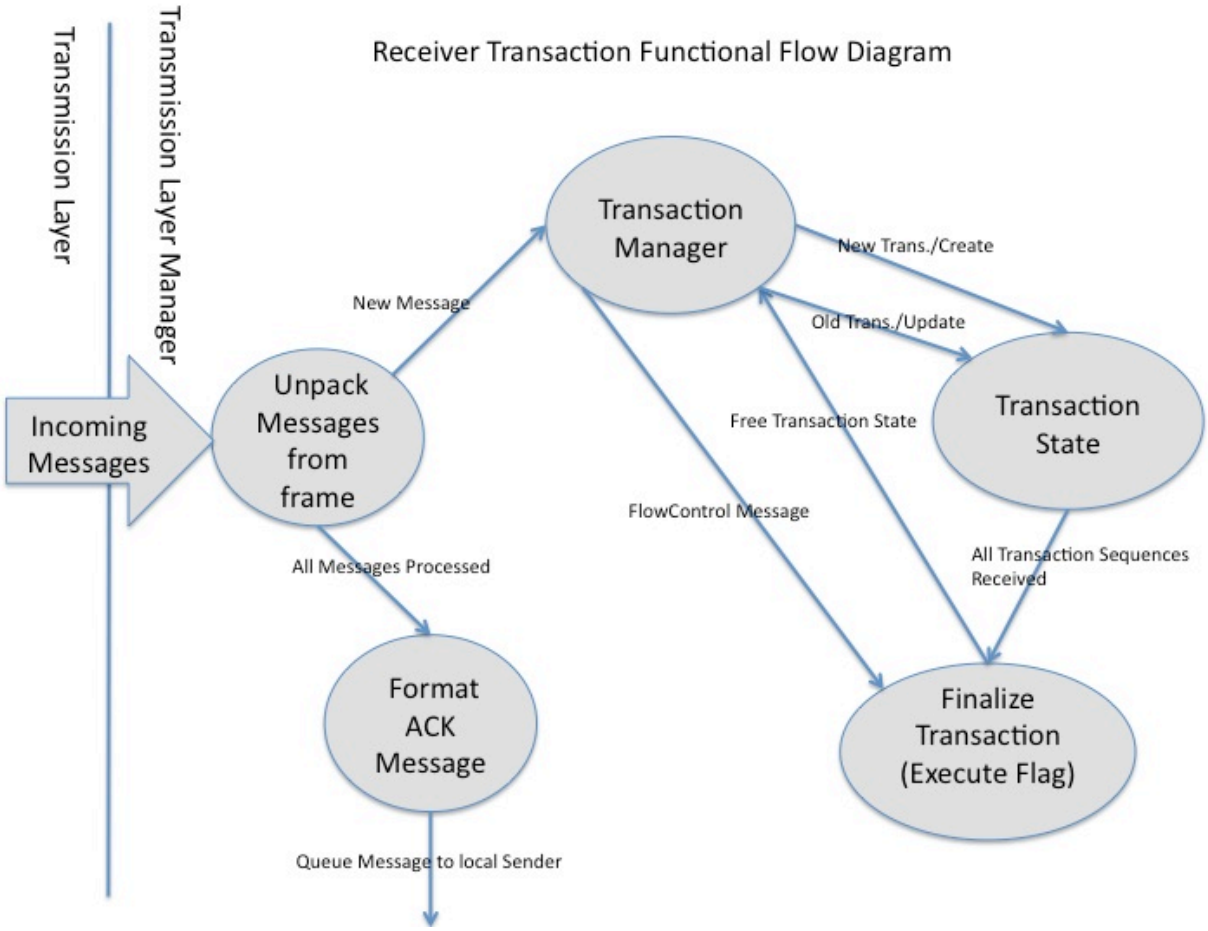
4.1 Sender

This diagram represents the functional flow for the protocol sender.



4.2 Receiver

This diagram represents the functional flow for the protocol receiver.



5 How this protocol is mapped onto transmission layers

5.1 Ethernet

5.1.1 Packet type ID

The OpenCPI DG/RDMA protocol will use type ID XXX when using the Ethernet L2 as its transmission layer.

The implementation must support a standard 1500 byte payload and optionally support a standard jumbo packet payload.

5.1.2 Header position

The DG-RDMA frame header starts immediately after the Ethernet packet type field.

5.1.3 OpenCPI RDMA Endpoint String Format

URLs representing OpenCPI RDMA Endpoints using this DG-RDMA protocol over Ethernet have the prefix:

```
ocpi-ether-rdma:<interface-name>/<mac address>
```

followed by the standard suffix:

```
;<endpoint-size>.<endpoint-id>.<max-endpoints>
```

5.2 IP/UDP

5.2.1 UDP Port

The OpenCPI DG/RDMA protocol will use dynamically assigned UDP port numbers from the typical “bind” API.

5.2.2 Header position

The DG-RDMA frame header starts 2 bytes into the UDP datagram. These two bytes are ignored.

5.2.3 OpenCPI RDMA Endpoint String Format

URLs representing OpenCPI RDMA Endpoints using this DG/RDMA protocol over IP/UDP have the prefix:

```
ocpi-udp-rdma:<ip-address>;<udp-port>
[needs verification vs. ipv6 addressing etc.]
```

followed by the standard suffix:

```
;<endpoint-size>.<endpoint-id>.<max-endpoints>
```

6 Notes on how this protocol is used by the OpenCPI RDMA upper layer

<describe how transactions are used to send data and metadata, and that flow control messages are transactions in the other direction>

The OpenCPI RDMA protocol uses endpoints to describe the globally unique address for a memory block that supports the reception and transmission of messages at worker ports. The endpoint can support as many worker ports as the resources allow. When the application establishes connections the endpoint and an offset uniquely describes a memory address that a remote sender can RDMA into. Therefore this protocol requires an endpoint format that is compatible. Note that when multiple endpoints are available on an interface, the mailbox is used to uniquely identify each endpoint. A “mailbox number” uniquely identifies endpoints at a higher level in the protocol stack. The string format of an endpoint provides sufficient addressing information for each side to contact the other. The string format of an endpoint is distributed to implementations by higher levels of software.

“ocpi-ether-l2-rdma:<mac address>:<memory block size>.<mail box>.<max mailbox>”

Where:

ocpi-ether-l2-rdma - The of the protocol

Mac address - The MAC address associated with the network interface

Memory block size – The overall length of the memory block supported by the endpoint in bytes.

Mail box – The mailbox number assigned to this endpoint.

Max mailbox – The maximum number of mailboxes allowed in the communication cluster.