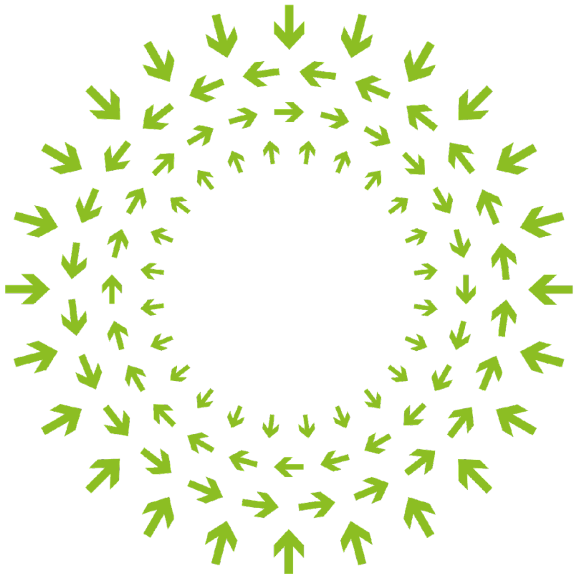


OPEN Compute Project

RDMA over Falcon Transport Specification

Revision 0.9

Date Submitted: 14 February, 2024
Date Approved: TBD

Authors: Prashant Chandra, Google

Table of Contents

1. License

Contributions to this Specification are made under the terms and conditions set forth in Open Web Foundation Contributor License Agreement (“OWF CLA 1.0”) (“Contribution License”) by:

Google

Usage of this Specification is governed by the terms and conditions set forth in the Open Web Foundation Final Specification Agreement (“OWFa 1.0”).

2. Compliance with OCP Tenets

2.1 Openness

The specification complies with the tenet of Openness by empowering the Community with Google’s production learnings to help modernize Ethernet. This includes leveraging production-proven technologies at scale including Carousel, Snap, Swift, Protective Load Balancing, and Congestion Signaling (CSIG) that have been openly published previously.

2.2 Efficiency

The specification complies with the tenet of Efficiency. Falcon achieves high performance by combining three key insights that achieve low latency in high-bandwidth, yet lossy, standard Ethernet data center networks. Fine-grained hardware-assisted round-trip time (RTT) measurements with flexible, per-flow hardware-enforced traffic shaping, and fast and accurate packet retransmissions, are combined with multipath-capable and PSP-encrypted Falcon connections. On top of this foundation, Falcon has been designed from the ground up as a multi-protocol transport capable of supporting Upper Layer Protocols (ULPs) with widely varying performance requirements and application semantics. The ULP mapping layer not only provides out-of-the-box compatibility with Infiniband Verbs RDMA and NVMe ULPs, but also includes additional innovations critical for warehouse-scale applications such as flexible ordering semantics and graceful error handling. Last but not least, the hardware and software are co-designed to work together to help achieve the desired attributes of high message rate, low latency, and high bandwidth, while maintaining flexibility for programmability and continued innovation.

2.3 Impact

The specification complies with the tenet of Impact by introducing a new technology that helps the industry modernize Ethernet. Falcon provides a helpful solution to address demanding workloads that have high burst bandwidth, high Operations per second, and low latency in

massive scale AI/ML training, High Performance Computing, and real-time analytics.

2.4 Scale

The specification complies with the tenet of Scale by being designed from the ground up to deliver high bandwidth and low latency in high-bandwidth, yet lossy, Ethernet data center networks. Additionally, it is composed of production-proven technologies delivered at scale including Carousel, Snap, Swift, Protective Load Balancing, and Congestion Signaling (CSIG).

2.5 Sustainability

The specification complies with the tenet of Sustainability by delivering an efficient Ethernet transport technology that minimizes retransmissions and other wasted effort and energy within an Ethernet network. Additionally, the technology allows a wide range of high performance workloads to be run on standard Ethernet networks.

3. Change Log

Date	Version #	Author	Description
14 FEB 2024	0.9	Prashant Chandra	Defines the RDMA over Falcon transport protocol

4. Scope

This specification describes the mapping of the RDMA ULP to the Falcon transport protocol including packet formats, supported operations and error handling modes.

Not in scope are:

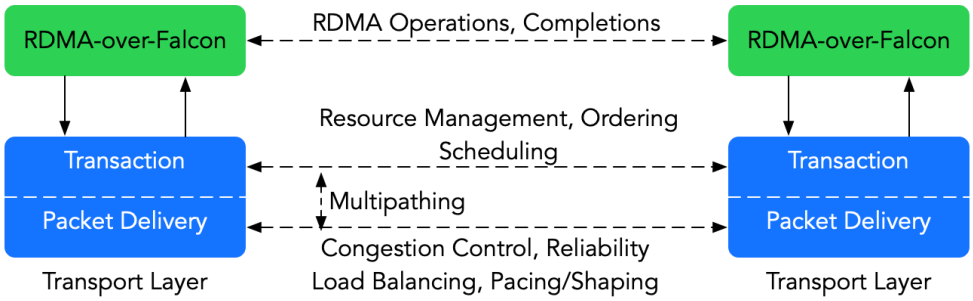
- Details of Falcon protocol.
- Details of applications' use of RDMA or Falcon.
- Details of implementing RDMA/Falcon in software stacks or hardware NICs.

5. Overview

This specification describes the mapping of the RDMA ULP to the Falcon transport protocol including packet formats, supported operations and error handling modes. The RDMA ULP is defined by the Infiniband Verbs specification and the mapping of RDMA over the Falcon transport supports the Reliable Connection (RC) and Unreliable Datagram (UD) modes of the Verbs specification.

6. Protocol Architecture

6.1 Protocol Layers



The figure above shows the protocol layering of RDMA as an upper layer protocol on top of Falcon. Falcon itself is composed of two sublayers: transaction sublayer and the packet delivery sublayer. The transaction sublayer primarily deals with ULP transactions and is responsible for resource allocation and transaction ordering. The packet delivery sublayer primarily deals with network packets and is responsible for reliable delivery and congestion control.

The RDMA-over-Falcon layer defines the mapping of RDMA commands and completions to Falcon packets and the wire-protocol used to communicate between RDMA-over-Falcon peers.

6.2 Ordering Modes

RDMA-over-Falcon supports three ordering modes described below:

- Strongly ordered mode: This mode is equivalent to the Infiniband Verbs ordering model and requires in-order data placement and in-order completions.
- Weakly ordered mode: This mode is based on the iWarp ordering model and supports out-of-order data placement with in-order completions.
- Unordered mode: This mode supports out-of-order data placement with out-of-order completions.

The mapping of RDMA ordering modes to Falcon ordering modes are described in the table below:

RDMA Ordering Mode	**Falcon Ordering Mode**
Strongly Ordered Queue Pair	Ordered Connection
Weakly Ordered Queue Pair	Unordered Connection
Unordered Queue Pair	Unordered Connection

6.3 Error Handling Modes

RDMA-over-Falcon defines two error handling modes described below:

- Verbs compatible mode: In this mode, the error handling is conformant to the model defined by the Infiniband Specification. Errors are typically reported out-of-band via Asynchronous Events (AEs) and many errors are typically fatal to a QP.
- Complete-in-error mode: In this mode, errors are reported in-band via regular completions and results in the errored operation failing while the QP continues to operate.

6.4 Supported Operations

RDMA-over-Falcon supports all of the RDMA opcodes as defined in the Infiniband Specification in the strongly ordered mode with Verbs compatible error handling mode. However, there are some constraints when using other ordering and error handling mode combinations. The supported operations are described in the table below. Like in Infiniband, UD Op sizes are limited to 1 MTU.

Ordering Mode	QP Type	Completion Ordering	Data Ordering	Error Semantics	Read/Write Size Limits	Send Size Limit
Strongly Ordered	RC	Strictly in-order	Strictly in-order	Configurable	Any Size	Any Size
Strongly Ordered	UD	Strictly in-order	Strictly in-order	Configurable	N/A	1 MTU
Weakly Ordered	RC	Strictly in-order	Out-of-order	Configurable	Any Size	Any Size
Weakly Ordered	UD	Strictly in-order	Out-of-order	Configurable	N/A	1 MTU
Unordered	RC	Out-of-order	Out-of-order	Complete-in-Error	Any Size (except Write with Immediate data)	1MTU
Unordered	UD	Out-of-order	Out-of-order	Complete-in-Error	N/A	1 MTU

6.5 Flow Control

RDMA-over-Falcon must implement the credit based flow control between Falcon and ULP as described in the ULP Resource Management section of the Falcon Protocol Specification. The credit-based flow control can be used to limit the use of Falcon resources by individual RDMA QPs. It can also be used at a coarser grain to control resource consumption at a SRIOV VF level or a PF level.

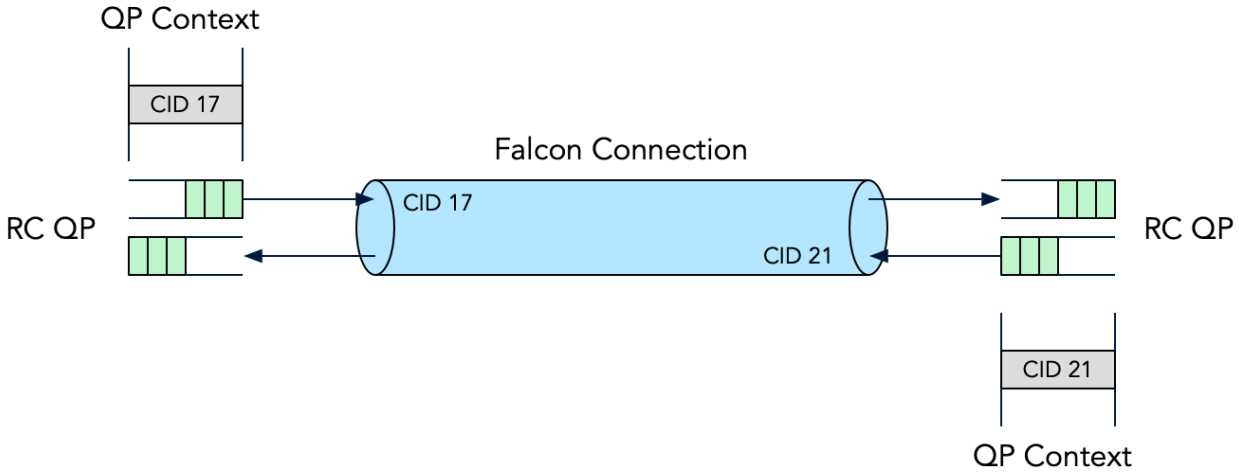
In addition to the credit based flow control, an implementation may choose to implement a global Xon/Xoff signal at the interface between RDMA and Falcon. In such a case, the implementation must ensure that the assertion of the Xoff signal does not block Read Responses from being sent from RDMA to Falcon. This is required for protocol deadlock avoidance.

6.6 Mapping RDMA QP Types to Falcon Connections

The mapping between RDMA QPs and Falcon connections depends on the QP type and is described in the following sections. A pair of connection IDs (CIDs) one for each direction of the

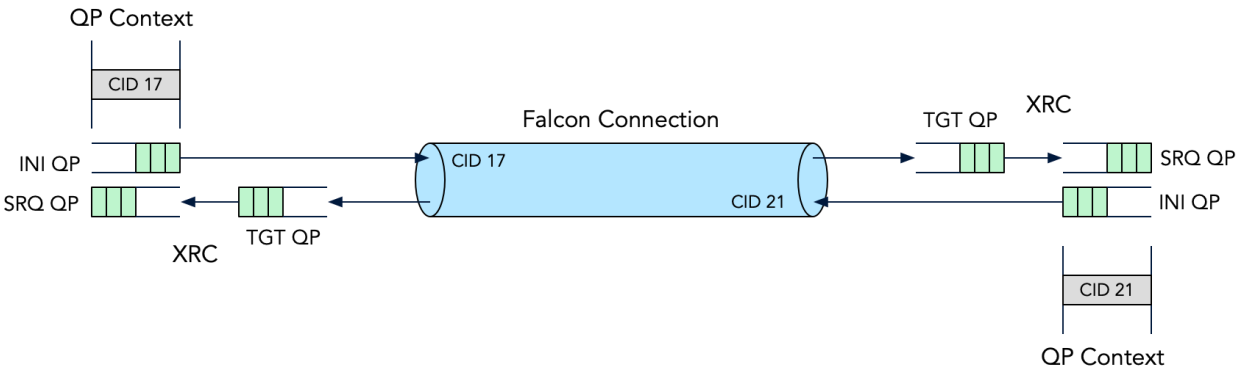
data flow identifies each Falcon connection. The mapping between a RDMA QP and a Falcon connection defines how the CID is determined for Send Queue (SQ) operations.

6.6.1 RC Queue Pairs



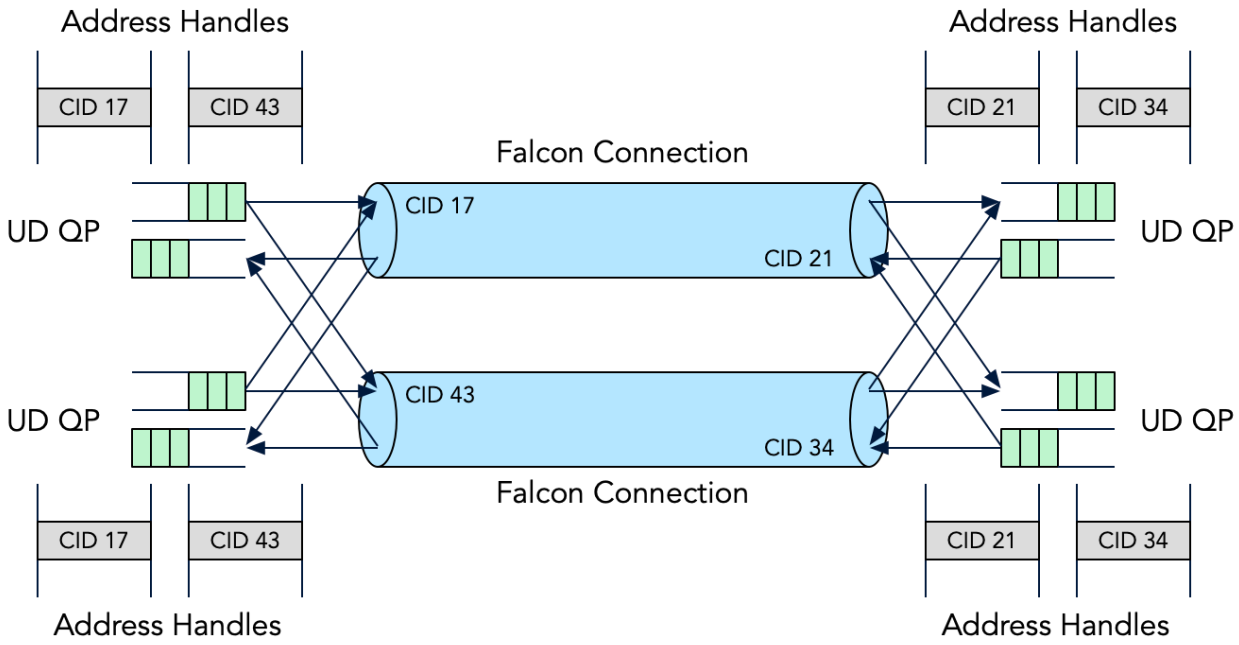
For reliable connection (RC) queue pairs, there is a 1:1 association between a QP and a Falcon connection as shown in the figure above. The mapping is specified by storing the associated CID in the QP context of the RC QP as shown above.

6.6.2 XRC Queue Pairs



For extended reliable connection (XRC) queue pairs, there is a 1:1 association between a XRC initiator (INI) QP and a Falcon connection as shown in the figure above. The mapping is specified by storing the associated CID in the QP context of the XRC INI QP as shown above. We note that XRC versus RC differences do not impact RDMA-over-Falcon ULP mapping. These differences are taken care of in RDMA ULP as described in the Infiniband specification.

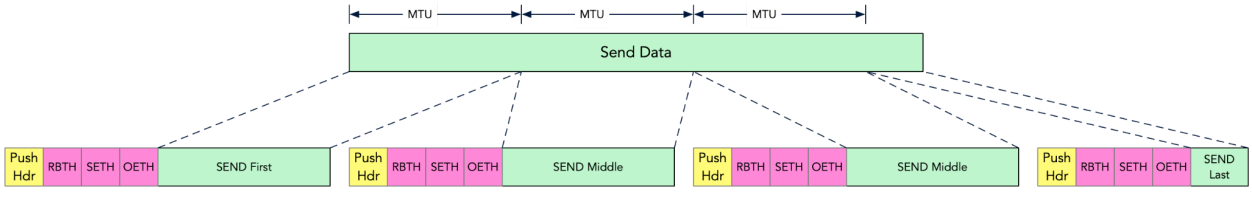
6.6.3 UD Queue Pairs



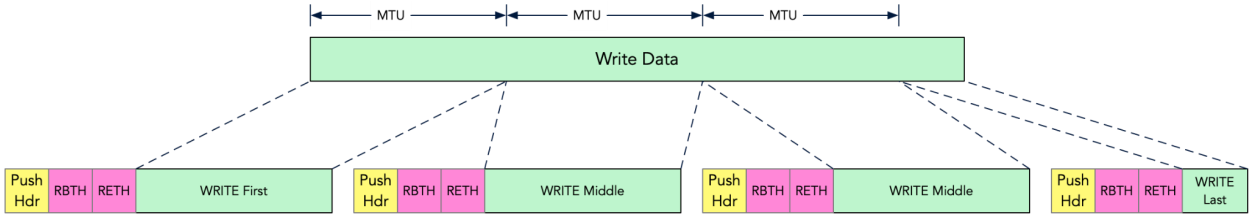
For unreliable datagram (UD) queue pairs, the association between QPs and connections can be many-to-many (or M:N). The above figure shows an example of a 2:2 mapping where 2 UD QPs are mapped to 2 Falcon connections. Send operations on a UD QP can target multiple destinations (and hence multiple Falcon connections). For UD QPs, the mapping to a Falcon connection is specified by storing the CID in the Address Handle (AH) object associated with the send operation as shown in the above figure.

6.7 Op Segmentation and Reassembly

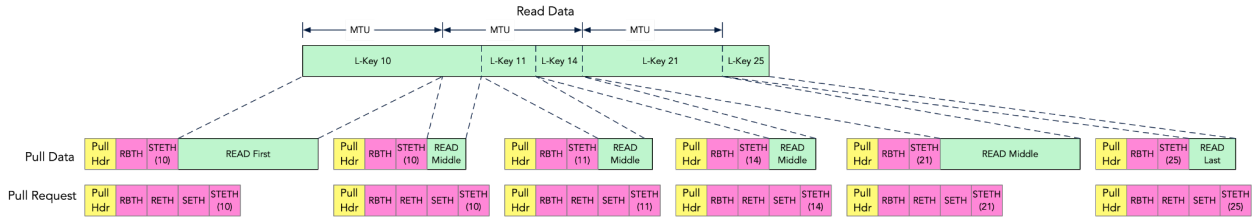
Falcon transactions can be at most MTU-sized. Therefore, the RDMA-over-Falcon layer is responsible for segmenting large Verbs operations posted by software into MTU-sized Falcon transactions and aggregate individual completions received from Falcon into the op-level completion back to software. This segmentation and reassembly is only required for RC queue pairs since all UD operations are limited to 1 MTU per the Infiniband specification.



The above figure shows an example of the segmentation of a multi-MTU SEND operation. The RDMA-over-Falcon layer breaks up the large SEND operation into 4 push transactions. The SEND first and SEND middle operations are MTU-sized push transactions. The SEND last operation can be less than or equal to a MTU-sized push transaction. Each push transaction generated by the RDMA-over-Falcon layer contains the RDMA base header (RBTH) along with the Sequence Number Extended Transport Header (SETH) and the Offset Extended Transport Header (OETH). The information contained in the SETH and OETH headers allows for out-of-order data placement at the target in the weakly ordered mode.



The above figure shows an example of the segmentation of a multi-MTU WRITE operation. The RDMA-over-Falcon layer breaks up the large WRITE operation into 4 push transactions. The WRITE first and WRITE middle operations are MTU-sized push transactions. The WRITE last operation can be less than or equal to a MTU-sized push transaction. Each push transaction generated by the RDMA-over-Falcon layer contains the RDMA base header (RBTH) along with the RDMA Extended Transport Header (RETH). The RETH is included in each push request so that the target can place the incoming data out-of-order to support both the weakly ordered mode and the unordered mode.



The above figure shows an example of the segmentation of a multi-MTU READ operation. Unlike multi-MTU SEND and WRITE operations, the segmentation of a multi-MTU READ operation must take into account the scatter-gather list (SGL) specified in the Read WQE to place the read response. The Infiniband specification allows a SGL list to contain multiple fragments each with its own L-Key. To support the unordered mode of operation, the RDMA-over-Falcon layer must break up a large READ operation into multiple pull transactions based on both the MTU size and the fragment boundaries. A pull transaction must not be larger than an MTU and must also not cross fragment boundaries.

In the above example, the large READ operation is segmented into 6 pull transactions. Unlike the WRITE and SEND case, the READ first and READ middle transactions can be smaller than an MTU due to the constraint that fragment boundaries cannot be crossed. Each pull transaction generated by the RDMA-over-Falcon layer contains the RDMA base header (RBTH) along with the RDMA Extended Transport Header (RETH), Sequence Number Extended Transport Header (SETH) and the Sink Tag Extended Transport Header (STETH). The RETH is included in each pull request so that the target can read the data out-of-order to support both the weakly ordered mode and the unordered mode. The information contained in the SETH and OETH headers allows for out-of-order data placement of the read response at the initiator in the weakly ordered and unordered modes.

6.8 RDMA Falcon Contract

The RDMA-over-Falcon layer assumes the following requirements that all implementations of the Falcon transport must meet:

1. The Falcon implementation must deliver a packet at most once to the RDMA ULP. All packet retransmissions and loss-recovery must be handled within the Falcon transport and must be transparent to the RDMA ULP. If Falcon is unable to deliver a packet for whatever reason, an error must be reported to the RDMA ULP which may be signaled to software as a connection tear-down event.
2. The Falcon implementation must complete a push or pull transaction at most once to the RDMA ULP.
3. In the strongly ordered mode, Falcon must deliver packets and completions to the RDMA ULP in strict order. The Falcon implementation is expected to reorder packets and completions as necessary to maintain ordering in this mode.
4. The Infiniband Invariant Cyclic Redundancy Check (ICRC) is not supported and the RDMA ULP does not perform ICRC checks on Falcon packets delivered to the ULP.
5. The RDMA ULP does not implement any congestion control mechanisms. Congestion control is expected to be fully implemented within the Falcon transport.

6.9 Security

In addition to the basic security mechanisms provided within the Falcon transport (authentication, encryption and replay protection), the RDMA-over-Falcon layer implements additional security checks for reliable connection (RC) queue pairs to prevent a compromised host from spoofing traffic. These security checks are described below:

- On receiving a packet from Falcon, the RDMA ULP must check the Falcon connection ID (CID) in the received packet against the CID stored in the QP context associated with that CID.

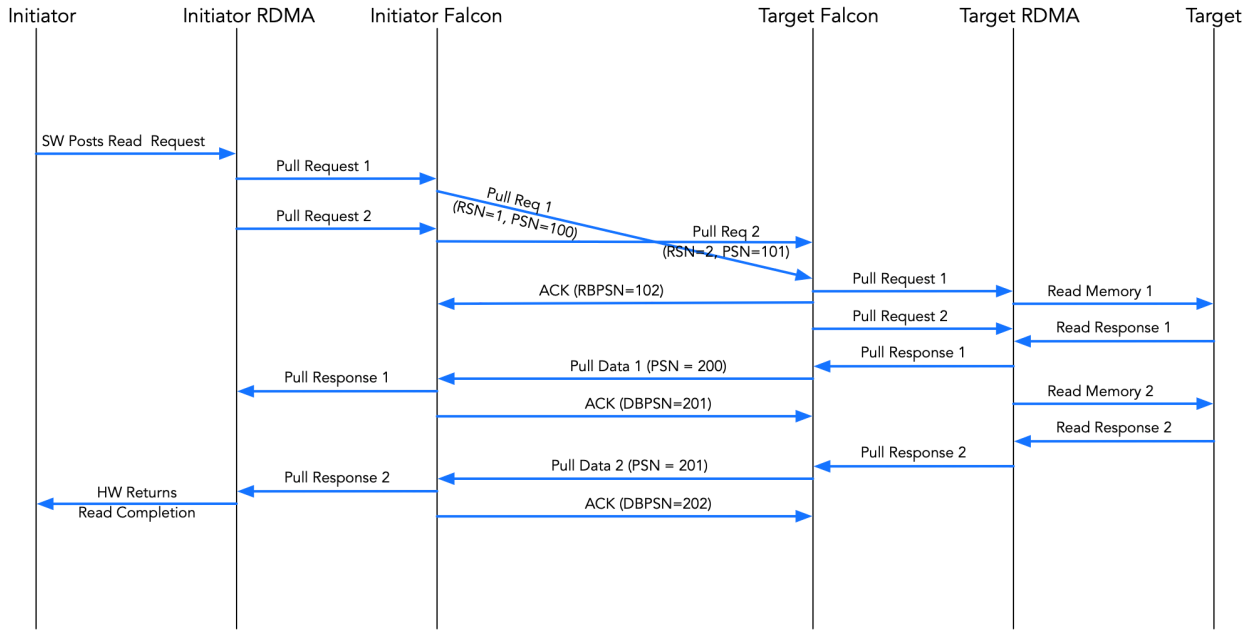
- If the CID does not match for a request (pull or push), the RDMA ULP must return a NACK CID response back to Falcon. This NACK CID response must not stop processing on the queue pair since the connection associated with the queue pair is different than that indicated by the CID in the incoming packet.
- If the CID does not match for a response (push completion or pull response), the response must be dropped and no NACK indication must be provided to Falcon.

These checks prevent an attacker from using an existing Falcon connection to deliver data to an incorrect QP that may belong to a different application or Virtual Machine.

7. RDMA Flows

The following sections describe the RDMA operation flow between the initiator and the target.

7.1 RDMA Read Flow



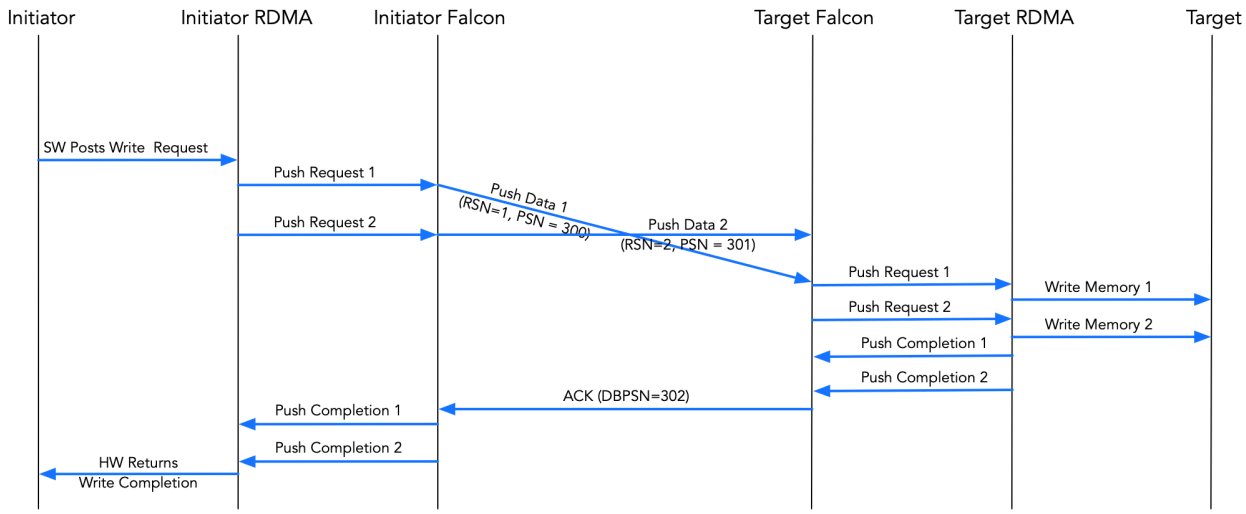
The life of a RDMA Read transaction is shown in the figure above. The connection is assumed to be an ordered connection. The following sequence of operations are performed at the initiator and target sides:

1. Software posts a RDMA read request to a send queue. This results in the RDMA protocol engine issuing two pull requests to Falcon. The two pull requests are created

because each pull request is limited to one MTU in length and the original read request is larger than one MTU.

2. The initiator creates two pull request packets (with RSN=1 and RSN=2) and transmits them to the target. The packet delivery sublayer assigns PSN=100 and PSN=101 to the two pull request packets.
3. The two pull request packets are reordered by the network and arrive out of order at the target. The target reorders the two pull requests by RSN and delivers them to the RDMA engine. The target also triggers the generation of the ACK packet acknowledging the receipt of the two pull request packets. In the figure above, a single coalesced ACK is sent from the target to the initiator.
4. The RDMA block performs the memory read operation for each pull request and returns pull responses to the target.
5. The target creates two pull data packets and transmits them to the initiator. The packet delivery sublayer assigns PSN=200 and PSN=201 to the pull responses.
6. The initiator receives the two pull data packets and creates pull responses back to the RDMA block. The initiator also triggers the generation of ACK packets by the packet delivery sublayer acknowledging the receipt of the pull data packets.
7. After the RDMA block at the initiator receives both pull responses, it creates a read completion and posts it to the completion queue.

7.2 RDMA Write Flow

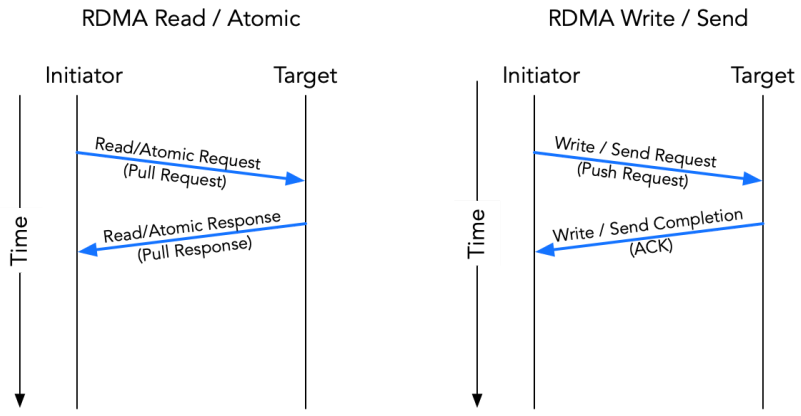


The life of a RDMA Write transaction is shown in the figure above. The connection is assumed to be an ordered connection. The following sequence of operations are performed at the initiator and target sides:

1. Software posts a RDMA write request to a send queue. This results in the RDMA protocol engine issuing two push requests to Falcon. The two push requests are created because each push request is limited to one MTU in length and the original write request is larger than one MTU.
2. The initiator creates two push data packets (with RSN=1 and RSN=2) and transmits them to the target. The packet delivery sublayer assigns PSN=300 and PSN=301 to the two push data packets.
3. The two push data packets are reordered by the network and arrive out of order at the target. The target reorders the two push data packets by RSN and delivers them to the RDMA engine.
4. The RDMA block performs the memory write operation for each push request and returns push completions to Falcon. After receiving the push completions, the target Falcon triggers the generation of the ACK packet acknowledging the receipt of the two push data packets. In the figure above, a single coalesced ACK is sent from the target to the initiator.
5. Upon receiving the ACK for the push data packets, the initiator delivers two push completions back to the RDMA block.
6. After the RDMA block at the initiator receives both push completions, it creates a write completion and posts it to the completion queue.

8. Wire Protocol

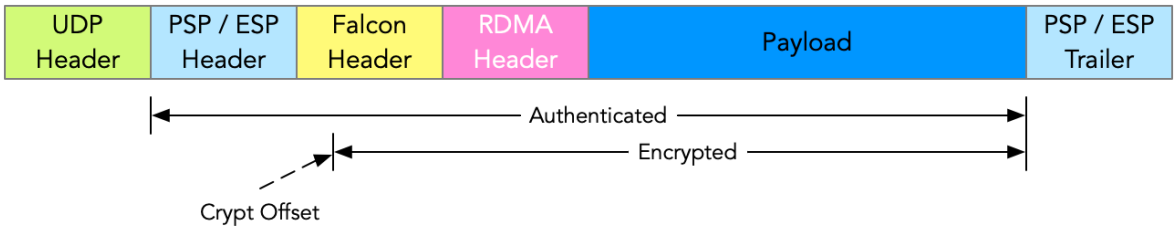
This RDMA-over-Falcon wire protocol defines the messages exchanged between the initiator and target systems to support RDMA operations and completions. The RDMA-over-Falcon wire protocol leverages Falcon for reliable transfer of messages across the network and can use the strictly ordered mode or the unordered mode of Falcon.



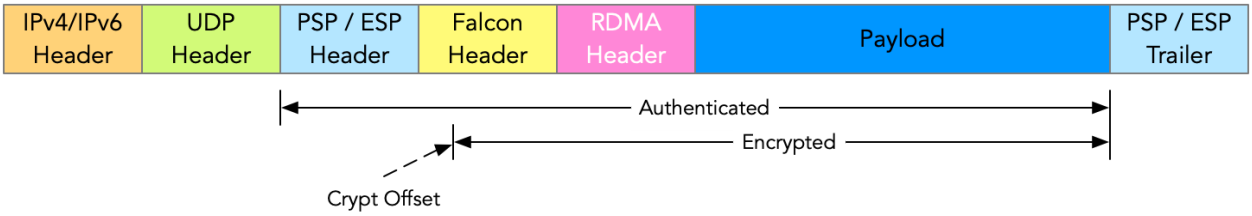
The high-level protocol exchange between the RDMA initiator and target is shown in the figure above.

8.1 Packet Format

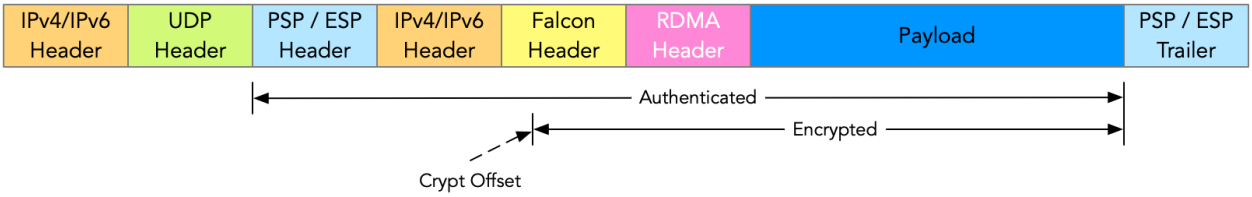
The packet format used by RDMA-over-Falcon is shown in the figure below. RDMA packets are encapsulated within the payload of a Falcon packet. RDMA defines multiple packet types for communication between the initiator and target that are listed in the table below and are described in detail in subsequent sections.



8.1.1 Transport Mode



8.1.2 Tunnel Mode



8.1.3 RDMA Packet Types

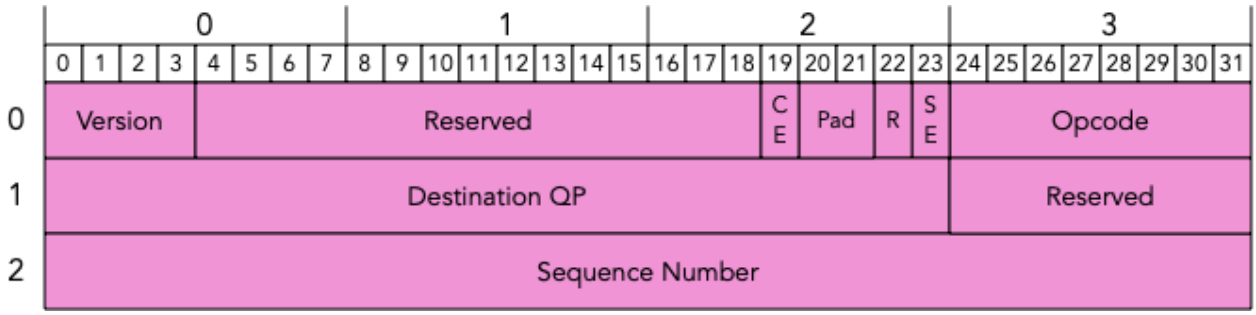
RDMA Opcode	RDMA Packet	RDMA Packet	Falcon Packet	RDMA Headers Present

[7:5]	[4:0]	Type	Source	Type	
000	00000	SEND First	Initiator	Push Request	RBTH, SETH, OETH
000	00001	SEND Middle	Initiator	Push Request	RBTH, SETH, OETH
000	00010	SEND Last	Initiator	Push Request	RBTH, SETH, OETH
000	00011	Send Last with Immediate	Initiator	Push Request	RBTH, SETH, OETH, ImmDt
000	00100	SEND Only	Initiator	Push Request	RBTH, SETH, OETH
000	00101	SEND Only with Immediate	Initiator	Push Request	RBTH, SETH, OETH, ImmDt
000	00110	WRITE First	Initiator	Push Request	RBTH, RETH
000	00111	WRITE Middle	Initiator	Push Request	RBTH, RETH
000	01000	WRITE Last	Initiator	Push Request	RBTH, RETH
000	01001	WRITE Last with Immediate	Initiator	Push Request	RBTH, RETH, SETH, ImmDt
000	01010	WRITE Only	Initiator	Push Request	RBTH, RETH
000	01011	WRITE Only with Immediate	Initiator	Push Request	RBTH, RETH, SETH, ImmDt
000	01100	READ Request	Initiator	Pull Request	RBTH, RETH, SETH, STETH
000	01101	READ Response	Target	Pull Data	RBTH, STETH

		First			
000	01110	READ Response Middle	Target	Pull Data	RBTH, STETH
000	01111	READ Response Last	Target	Pull Data	RBTH, STETH
000	10000	READ Response Only	Target	Pull Data	RBTH, STETH
000	10001	Reserved			
000	10010	ATOMIC Response	Target	Pull Data	RBTH, AtomicETH, STETH
000	10011	ATOMIC CmpSwap	Initiator	Pull Request	RBTH, AtomicAckETH, SETH, STETH
000	10100	ATOMIC FetchAdd	Initiator	Pull Request	RBTH, AtomicAckETH, SETH, STETH
000	10101	Reserved			
000	10110	SEND Last with Invalidate	Initiator	Push Request	RBTH, SETH, OETH, IETH
000	10111	SEND Only with Invalidate	Initiator	Push Request	RBTH, SETH, OETH, IETH
000	11000-11111	Reserved			
011	00000-00011	Reserved			
011	00100	SEND Only	Initiator	Push Request	RBTH, DETH
011	00101	SEND Only with Immediate	Initiator	Push Request	RBTH, DETH, ImmDt

011	00110-1 11111	Reserved

8.2 RDMA Base Transport Header (RBTH)



The RDMA base transport header (RBTH) must be present in every RDMA-over-Falcon packet. The format of the RBTH is as shown in the figure above. The following table documents the definition of various fields in the header.

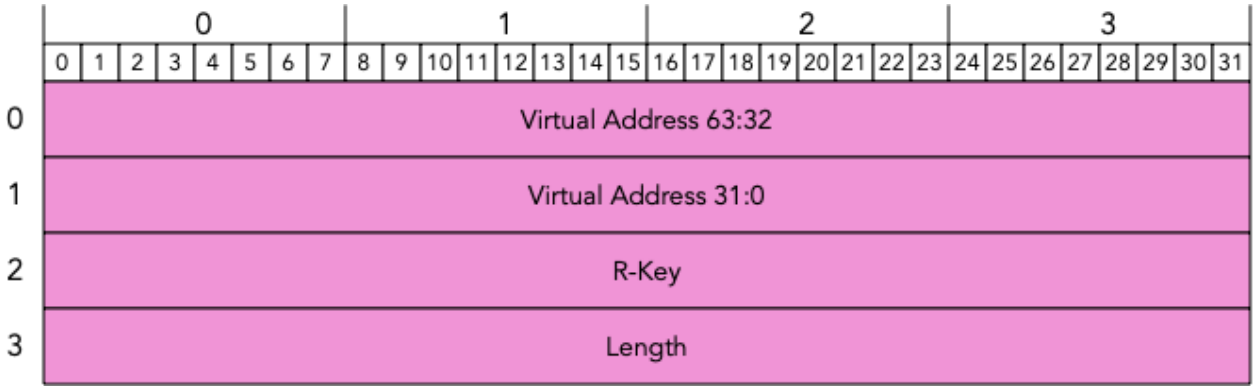
Field	Width	Description
Version	4	This field encodes the version of the RDMA-over-Falcon wire protocol. It must be set to 1.
Complete-in-Error (CE)	1	This bit must be set to 1 in the Last or Only packet of a message to indicate that the message must be completed in error on the initiator or target side.
Pad	2	This field encodes the pad bytes inserted to align the message to a 4 byte boundary.
AckReq (A)	1	This bit must be set to 1 to request an explicit acknowledgement from the target for this packet.
Reserved (R)	1	This field must be encoded as 0.
Solicited Event (SE)	1	This bit must be set by the initiator to indicate that the target shall invoke the CQ event handler. The rules for setting this bit are the same as those defined in the Infiniband Specification.
Opcode	8	This field specifies the RDMA operation and follows the same encoding as the RoCEv2 / Infiniband specifications. The mapping

		of RDMA opcodes to Falcon packet types is shown in the table here.
Destination QP	24	This field specifies the destination QP number.
Sequence Number (SN)	32	This field encodes a monotonically increasing sequence number. The first packet sent on a Falcon connection must be sent with SN = 1. SN is used to check that packets are delivered in order in strongly ordered mode and to detect missing packets in weakly ordered mode. RDMA Response packets must have the same SN as the corresponding RDMA Request packet.

8.3 RDMA Extended Transport Headers

RDMA-over-Falcon uses multiple extended transport headers that are defined in the following sections. The sequence of extended transport headers included in a RDMA-over-Falcon packet depends on the RDMA operation and is documented in the table here.

8.3.1 RDMA Extended Transport Header (RETH)

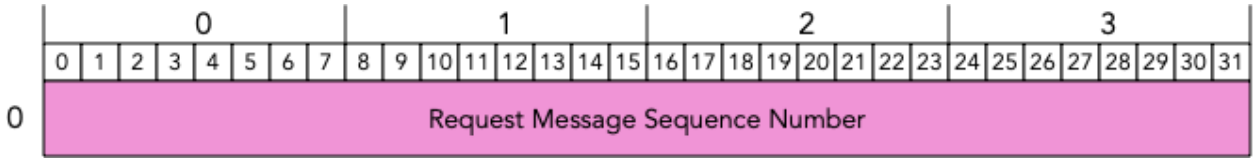


The RDMA extended transport header (RETH) is used in RDMA Read and RDMA Write operations and has the same format as defined in the Infiniband Specification (shown in the figure above). The following table describes the various fields in the header. The authoritative definition of the fields and the rules for encoding them are provided by the Infiniband Specification.

Field	Width	Description

Virtual Address (VA)	64	This field encodes the starting virtual address of the memory region that is being accessed at the target by the RDMA Read or Write operation. The VA can start on any byte boundary.
R-Key	32	This field encodes the memory region key provided by the target to the initiator and is used by the target for access control.
Length	32	This field encodes the length in bytes of the RDMA Read or Write operation.

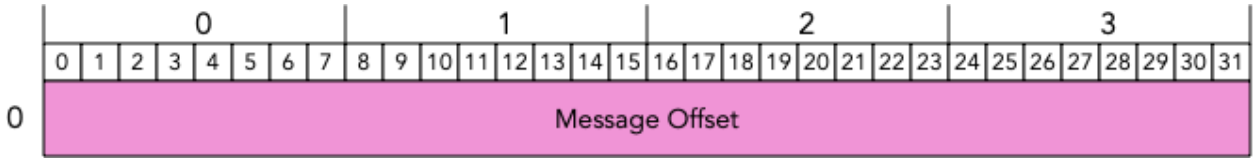
8.3.2 Sequence Number Extended Transport Header (SETH)



The Sequence number extended transport header (SETH) is used to support out-of-order data placement in the weakly-ordered mode and has the format as shown in the figure above. The following table describes the various fields in the header.

Field	Width	Description
Request Message Sequence Number (RMSN)	32	This field encodes a message sequence number that is assigned as follows: ● The initial value of RMSN must be set to 1. ● For outbound Send Immediate or Write Immediate operations, this field encodes (in the lower 8 bits) the index of the RQ WQE being consumed at the target. The value of RMSN must be incremented by 1 after the initiator transmits the last / only packet of an outbound Send or Write with Immediate data. ● For outbound Read or Atomic operations, the RMSN must be incremented by 1 for every Read or Atomic request packet. The RMSN encodes the index of the IRRQ entry at the responder that is used to store the Read or Atomic request. ● The initiator must maintain separate RMSN counters for Send/Write and Read/Atomic operations.

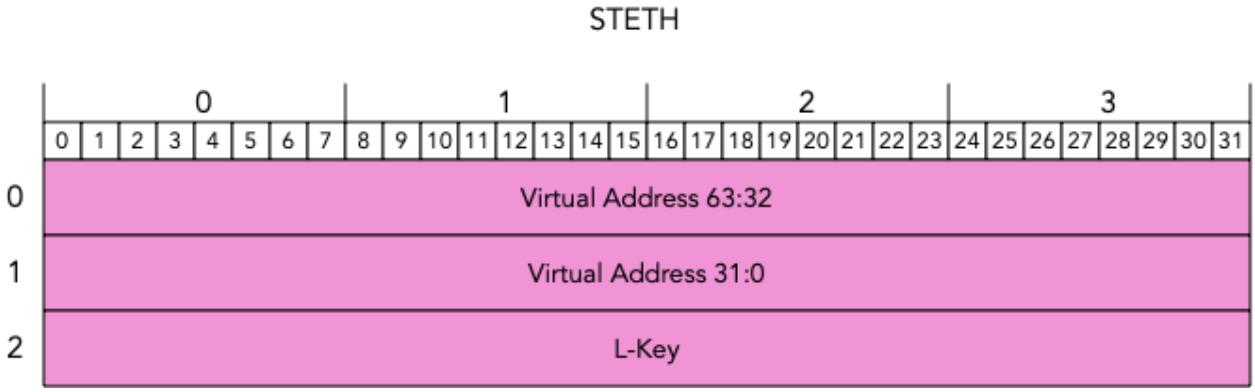
8.3.3 Offset Extended Transport Header (OETH)



The offset extended transport header (OETH) is used to support out-of-order data placement in the weakly-ordered mode and has the format as shown in the figure above. The following table describes the various fields in the header.

Field	Width	Description
Message Offset	32	This field encodes the byte offset into the message of the first byte of the Send packet. For out-of-order data placement, the target seeks into the relevant RQ WQE's (identified by SETH) scatter/gather list to place the packet bytes in the target buffer.

8.3.4 Sink Tag Extended Transport Header (STETH)



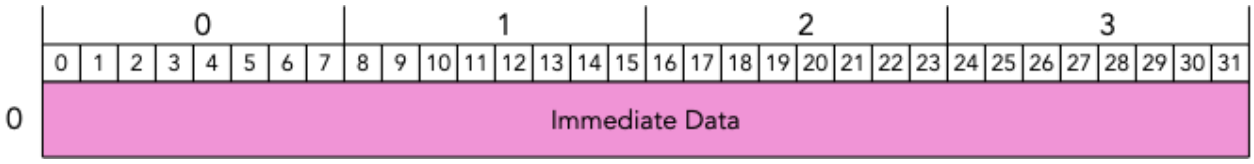
The Sink Tag extended transport header (STETH) is used to support out-of-order data placement of Read Response data at the initiator and has the format as shown in the figure above. The following table describes the various fields in the header.

Field	Width	Description
Virtual Address (VA)	64	This field encodes the virtual address of the initiator's sink buffer for the first byte of the read response data.

L-Key	32	This field encodes the memory region L-Key associated with the initiator's sink buffer.

The STETH is inserted into a Read Request packet by the initiator and must be returned by the target in the corresponding Read Response packet.

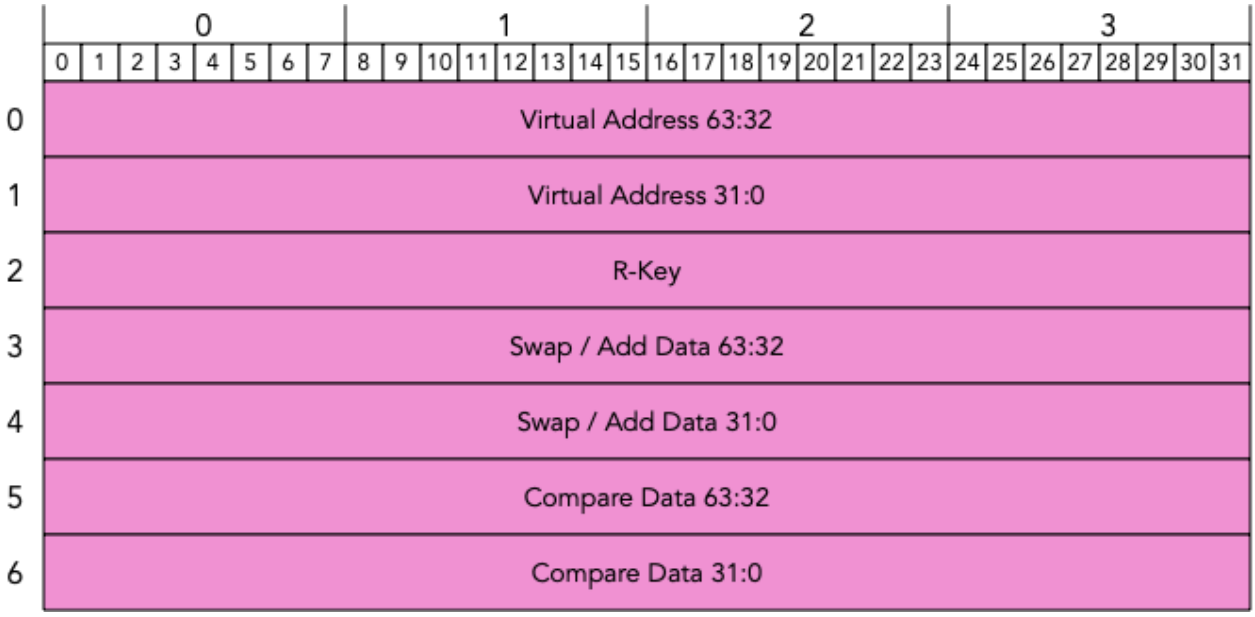
8.3.5 Immediate Extended Transport Header (ImmDt)



The Immediate extended transport header (ImmDt) is used in RDMA Send with Immediate and RDMA Write with Immediate operations and has the same format as defined in the Infiniband Specification (shown in the figure above). The following table describes the various fields in the header. The authoritative definition of the fields and the rules for encoding them are provided by the Infiniband Specification.

Field	Width	Description
Immediate Data	32	This field encodes the 32-bit immediate data that must be placed in the completion queue entry by the target.

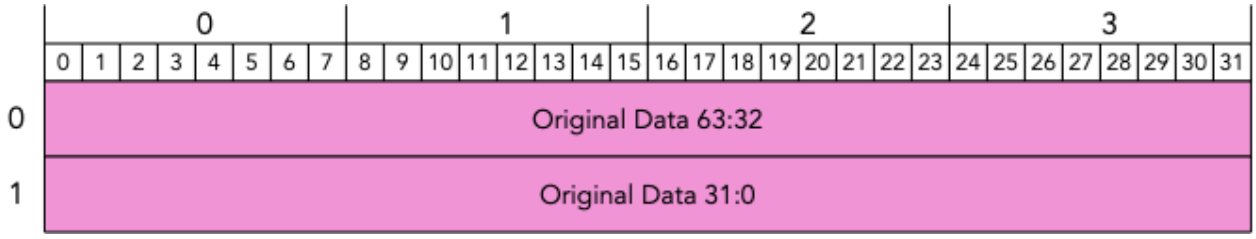
8.3.6 Atomic Extended Transport Header (AtomicETH)



The Atomic extended transport header (AtomicETH) is used in RDMA Atomic request packets and has the same format as defined in the Infiniband Specification (shown in the figure above). The following table describes the various fields in the header. The authoritative definition of the fields and the rules for encoding them are provided by the Infiniband Specification.

Field	Width	Description
Virtual Address (VA)	64	This field encodes the starting virtual address of the memory region that is being accessed at the target by the RDMA Read or Write operation. The VA can start on any byte boundary.
R-Key	32	This field encodes the memory region key provided by the target to the initiator and is used by the target for access control.
Swap / Add Data	64	In a CmpSwap operation this field is swapped into the addressed buffer if the Compare Data matches the existing buffer contents. In a FetchAdd operation this field is added to the contents of the addressed buffer.
Compare Data	64	The data operand used in Compare portion of the CmpSwap operation.

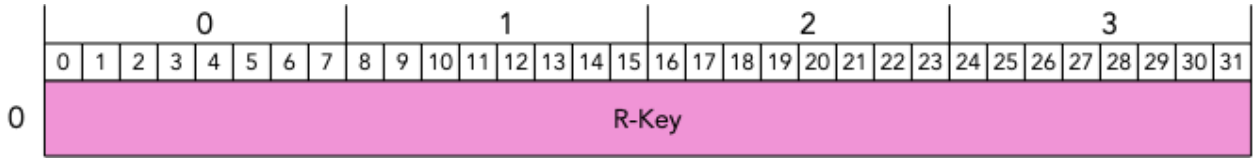
8.3.7 Atomic Acknowledgement Extended Transport Header (AtomicAckETH)



The Atomic Acknowledged extended transport header (AtomicAckETH) is used in RDMA Atomic response packets and has the same format as defined in the Infiniband Specification (shown in the figure above). The following table describes the various fields in the header. The authoritative definition of the fields and the rules for encoding them are provided by the Infiniband Specification.

Field	Width	Description
Original Data	64	The data result from an Atomic operation. This field encodes the original value at the address specified in the Atomic request.

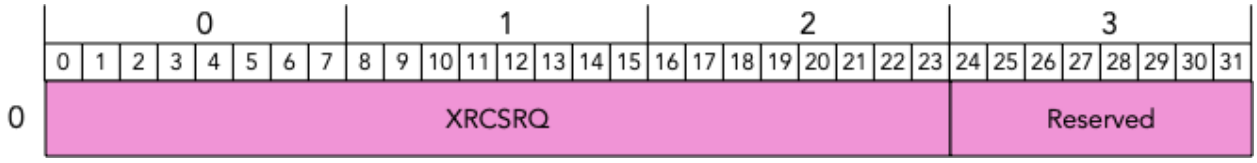
8.3.8 Invalidate Extended Transport Header (IETH)



The Invalidate extended transport header (IETH) is used in RDMA Send with Invalidate operations and has the same format as defined in the Infiniband Specification (shown in the figure above). The following table describes the various fields in the header. The authoritative definition of the fields and the rules for encoding them are provided by the Infiniband Specification.

Field	Width	Description
R-Key	32	This R-Key is used by the target to invalidate a memory region or memory window once it receives and executes the SEND with Invalidate request.

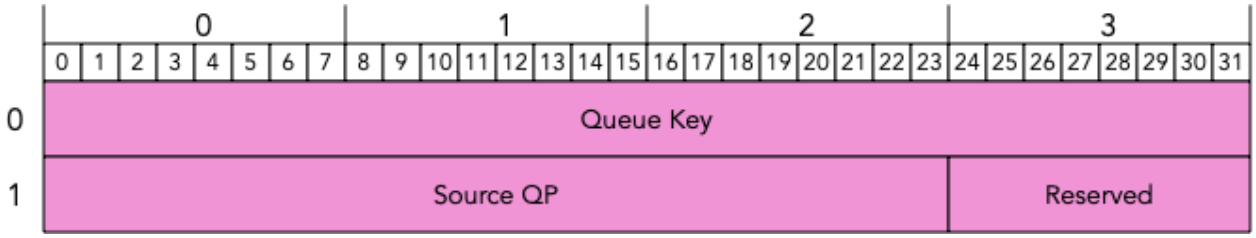
8.3.9 XRC Extended Transport Header (XRCETH)



The XRC extended transport header (XRCETH) is used in all request packets sent by the initiator that implements the XRC transport service and has the same format as defined in the Infiniband Specification (shown in the figure above). The following table describes the various fields in the header. The authoritative definition of the fields and the rules for encoding them are provided by the Infiniband Specification.

Field	Width	Description
XRCSRQ	24	This field encodes the XRC shared receive queue number to be used by the target for this packet.

8.3.10 Datagram Extended Transport Header (DETH)



The Datagram extended transport header (DETH) is used in RDMA Send operations on Unreliable Datagram (UD) queue pairs and has the same format as defined in the Infiniband Specification (shown in the figure above). The following table describes the various fields in the header. The authoritative definition of the fields and the rules for encoding them are provided by the Infiniband Specification.

Field	Width	Description
Queue Key	32	This field is required to authorize access to the destination queue. The target compares this field with the destination's QP Q_Key.

Source QP	24	This field specifies the source queue pair (QP) identifier. This is used as the destination QP for response packets.