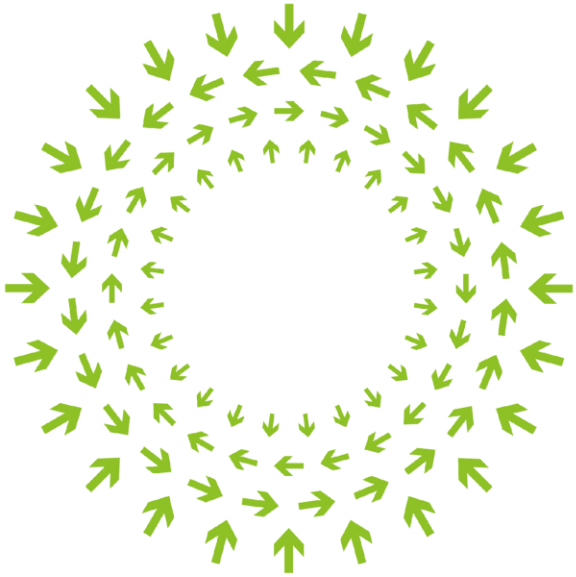

OPEN
Compute Project

Multipath Reliable Connection (MRC) Specification

Revision 1.0
Publication Date: 03/21/26

Rip Sohan[1], Eric Spada[1], Eric Davis[1], Mark Handley[1], Idan Burstein, Tony Hurson, Jithin Jose, Vivek Kashyap, Rong Pan, Sayantan Sur

Joint contribution from: AMD, Broadcom, Intel, Microsoft, NVIDIA, OpenAI

[1] Served as lead authors and technical contributors for this specification.

Table of Contents

Table of Figures

Table of Tables

Revision History

Authors	Date	Ver	Change details
Rip Sohan		0.1	Skeleton outline
Karen Schramm		0.2	Added packet reliability section
Rip Sohan David Riddoch Shane O'Neil		0.5	Transport skeleton
Eric Spada Karen Schramm Mark Handley		0.7	Added Reliable Delivery, Congestion Control content
Rip Sohan		0.75	Integrated Transport and other content
Eric Spada Mark Handley Guglielmo Morandin Eric Davis		0.80	Integrated new CC, review notes, resolved comments
Rip Sohan		0.90	Resolve comments, remove unnecessary ops, more cleanup and join up
Rip Sohan		1.04	Ver 1.04; external release 1
Rip Sohan		1.05	Bug fixes; cleanup
Idan Burstein		1.06	Fix MSN to comply with existing ACK format, fix BTH format, fix QPID to represent QPN of 24 bit, …
Idan Burnstein Eric Davis Mark Handley Vipin Jain Rip Sohan Eric Spada		1.07rc1	Changes from MSFT's feedback – Internal review candidate Enhanced/dual WIMM limiting Updated congestion control chapter Software API New reliable delivery packet formats Spelling/grammar
Idan Burnstein Eric Davis Mark Handley Vipin Jain Rip Sohan Eric Spada		1.07	Ver 1.07 Release
Rip Sohan		1.08	Include most of INTC's feedback from 1.07. Unresolved comments remain. SmaRTTrack pseudocode updates Ver 1.08 For Release To MSFT
Rip Sohan		1.08a	Fixup intro to correctly represent companies involved in MRC
Rip Sohan Rong Pan Eric Davis Eric Spada		1.09	Addresses 1.08 feedback from MSFT EV Probes; Reliability Probes; SRv6; Software API; Profiles; EV Events Cleanup

Rip Sohan Rong Pan Eric Spada Eric Davis Yamin Friedman Idan Burnstein Mark Handley Elazar Cohen Yuval Shpigelman		1.10 (MRC_REL_1_0-RC1)	Incorporate feedback from NVIDIA, OAI - Clarify Dynamic MPR negotiation and enforcement rules - Update definition of ePSN tracking window - Control packet EV generation is now implementation-defined - Update and clarify SACK generation rules - Make generation of TRIM NACKs optional and introduce neg. capability - Introduce class of Endpoint operations (EV Probe, Port Status Mask) - Clarify use of NSCC algorithm; parameters; signals - Added DSCP-to-named codepoints mapping - Clarify and simplify service time support - Various header cleanups - Add support for cross-plane control packets
Rip Sohan Eric Spada		1.11 (MRC_REL_1_0-RC2)	Incorporate feedback from review - Updated Section 8: Clarified EV selection and generation rules - Tightened definitions for entropy handling and EV profiles - Removed NACK m bit; ECN signaling remains in SACK only - Finalized SACK header M-bit semantics for ECN congestion - Clarified per-QP attributes requiring exchange, directionality - Described EV state machine with recovery - SRv6: clarified uSID stack handling and optional SRH behavior
Rip Sohan		1.12 (MRC_REL_1_0-RC3)	Incorporate feedback review: - Update cover page and logos - Clarify how forward-path entropy is reflected in control packets - Clarified behavior and formula for local ACK timeout - Removed stale text from Entropy Generation Table (5-2)
Rip Sohan		MRC_REL_1_0	- Fix timeout formula description - Add 2026 copyright update - Clarify Difference between EV Probes and Port Status Update - Drop internal versioning
Eric Spada Rip Sohan	03/06/26	MRC-OCP-rc2	Convert to OCP format
Eric Spada Rip Sohan	03/10/26	MRC-OCP-rc3	Update based on OCP (Alicia, Russ) comments

Eric Davis			
Rip Sohan Eric Spada Eric Davis	03/18/26	MRC-OCP-r4	Update based on internal (MRC group) comments; updated author list
Rip Sohan	03/21/26	MRC-OCP-1.0	Release 1.0 spec in OCP format

1. License

1.1. Open Web Foundation (OWF) CLA

Contributions to this Specification are made under the terms and conditions set forth in Modified Open Web Foundation Agreement 0.9 (OWFa 0.9). (As of October 16, 2024) (“Contribution License”) by:

- Advanced Micro Devices, Inc
- Avago Technologies International Sales Pte. Limited
- Intel Corporation
- Microsoft Corporation
- OpenAI OpCo, LLC
- NVidia Corporation

Usage of this Specification is governed by the terms and conditions set forth in Modified OWFa 0.9 Final Specification Agreement (FSA) (As of October 16, 2024) (“Specification License”).

You can review the applicable Specification License(s) referenced above by the contributors to this Specification on the OCP website at https://www.opencompute.org/contributions/templates-agreements.

For actual executed copies of either agreement, please contact OCP directly.
Notes:

The above license does not apply to the Appendix or Appendices. The information in the Appendix or Appendices is for reference only and non-normative in nature.

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT

(INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.2. Acknowledgements

This specification includes contributions from:

Author	Company
Rip Sohan	AMD
Vipin Jain	AMD
Rong Pan	AMD
David Riddoch	AMD
Shane O'Neil	AMD
Siva Santosh Pyla	AMD
Raghava Sivaramu	AMD
Yanfang Le	AMD
Eric Spada	Broadcom
Eric Davis	Broadcom
Karen Schramm	Broadcom
Guglielmo Morandin	Broadcom
Costin Raiciu	Broadcom
Jithin Jose	Microsoft
Abdul Kabbani	Microsoft
Torsten Hoefler	Microsoft
Shahaf Shuler	NVIDIA
Sayantan Sur	NVIDIA
Idan Burnstein	NVIDIA
Yamin Friedman	NVIDIA
Elazar Cohen	NVIDIA
Yuval Shpigelman	NVIDIA
Mark Handley	OpenAI
Amin Tootoonchian	OpenAI
Michael Papamichael	OpenAI
Adrian Caulfield	OpenAI

2. Copyright / Trademark

Trademarks:

- InfiniBand™and RoCE™ are trademarks of the Infiniband Trade Association
- UltraEthernet™ is a trademark of the UltraEthernet Consortium
- PCIe™ is a trademark of PCI-SIG

Materials utilized in this specification carry the following copyright:

- The Infiniband Architecture Specification Volume 1 Release 1.8 is copyright the Infiniband Trade Association
- The UltraEthernet Specification Version 1.01 is copyright The UltraEthernet Consortium

3. Compliance with OCP Tenets

3.1. Openness

This specification embodies OCP's tenet of openness by publishing the combined design of six organizations that have designed and operated hyperscale AI/ML systems across multiple generations of interconnect protocols and transports. It is grounded in existing, widely adopted protocols, transports, and software abstractions to enable rapid implementation, interoperability, and reuse of established tooling, operational practices, and engineering expertise. The specification defines protocol semantics, transport mappings, and software APIs that have been validated in large-scale production deployments and makes these interfaces available for broad adoption, extension, and integration. Wherever possible, it leverages and extends existing technologies rather than introducing proprietary mechanisms, thereby reducing integration friction and enabling third parties to implement, modify, optimize, and evolve the architecture over time.

3.2. Efficiency

This specification advances OCP's efficiency tenet by improving AI/ML system efficiency across multiple dimensions. MRC increases overall network utilization and endpoint goodput, allowing existing infrastructure to carry more effective AI/ML traffic without requiring specialized fabrics. It enhances endpoint resilience by making endpoints more tolerant of transient and intermittent network faults (such as link flaps), reducing performance degradation and recovery overhead. All of these gains are realized over standard best-effort Ethernet, enabling operators to achieve higher efficiency using broadly deployed, commodity networking technologies.

3.3. Impact

MRC advances OCP's impact tenet by defining an open, implementation-ready design for resilient, high-utilization AI/ML networking that can be deployed over standard Ethernet and integrated into existing hardware, firmware, and software stacks. By specifying interoperable protocol behavior at the endpoints, MRC enables a broad ecosystem of NIC, switch, accelerator, and system vendors to deliver consistent semantics and performance, reducing integration risk and time-to-market for large-scale AI/ML deployments. Its reliance on commodity Ethernet, Infiniband and UltraEthernet concepts and standardized APIs makes the technology accessible to a wide range of operators and solution providers globally, fostering multi-vendor implementations, promoting supply-chain diversity, and enabling widespread adoption of more dependable, efficient AI/ML infrastructure within the OCP community.

3.4. Scale

MRC advances OCP's scale tenet by defining a transport that autonomously detects and mitigates congestion, intermittent faults, and hard link failures, including dynamic path health evaluation and automatic traffic re-routing without operator or fabric intervention. The transport allows endpoints to probe peer and path liveness and to advertise their own reachability, enabling self-healing connectivity across arbitrarily large topologies. Its routing modes support transparent network expansion, contraction, and reconfiguration without downtime or manual reprogramming. A control-plane API exposes telemetry and controls for integration with monitoring and orchestration agents, providing unified observability, runtime traffic management, failure notification, and coordinated recovery, so that deployments can scale from small clusters to hyperscale fabrics with minimal operational overhead.

3.5. Sustainability

MRC advances OCP's sustainability tenet by enabling AI/ML systems to fully utilize the existing network fabric, even in the presence of link faults and localized congestion, thereby reducing wasted accelerator cycles and avoiding unnecessary job aborts and reruns. By maintaining high effective throughput under fabric impairments and preventing fabric issues from cascading into system-wide job failures, MRC improves end-to-end accelerator utilization. This more efficient use of already-installed networking and compute capacity reduces the need for overprovisioning, lowers the marginal energy and resource cost per successful training or inference job, and supports more sustainable scaling of large AI/ML clusters over time.

4. Introduction

4.1. Purpose of the Document

This document defines the MRC specification, an extension to the InfiniBand Reliable Connection (RC) transport collaboratively developed by AMD, Broadcom, Intel, Microsoft, OpenAI, and NVIDIA. Its purpose is to enable scalable, high-performance AI/ML workloads over multipath best-effort networks by specifying architectural extensions for multipath load balancing, reliability, and congestion control, along with the associated protocol behavior and software APIs.

The document aims to provide a clear, interoperable definition of MRC features, limitations, message formats, error handling, and control-plane interfaces so that implementers can build compatible endpoints and systems that deliver efficient, reliable performance at scale.

4.2. Scope

This document specifies the MRC transport including its architecture, protocol semantics, transport behavior, wire-visible state machines, error handling, and the control-plane and data-plane APIs required for interoperable implementations. It covers endpoint behavior, multipath load balancing, reliability and recovery mechanisms, congestion control, routing modes, and integration hooks for system-level monitoring, orchestration, and management software.

The document does not define specific hardware microarchitectures, link or physical layer specifications, or non-MRC transport modes. Operational policies and configurations are out of scope, except where used as illustrative examples.

4.3. Audience

This document is intended for architects, systems engineers, network engineers, switch vendors and software developers responsible for designing, implementing, or integrating MRC in their products. It is also targeted at operators and SREs who deploy and manage MRC clusters, and at technical program and product managers who need to understand MRC capabilities, constraints, and interoperability requirements for planning and decision-making.

4.4. Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 (and as updated by RFC 8174) when, and only when, they appear in all capitals.

Normative statements in this specification define requirements for compliant implementations. All other text is informative and does not create conformance obligations.

Unless explicitly stated otherwise:

- All numeric values are in decimal
- Sizes and offsets are in bytes
- Bit positions are numbered with bit 0 as the least significant bit
- Multi-bit fields are described using inclusive ranges in the form bit[m:n], where m is the most significant bit and n is the least significant bit
- Multi-byte integer fields are represented in network byte order unless otherwise specified

All diagrams showing fields and headers are for illustrative purposes only; the normative definition is given by the field descriptions and bit/byte ranges.

Pseudocode and code fragments are informative.

Examples are non-normative and do not impose additional requirements beyond those stated explicitly.

5. Overview

The MRC specification defines a transport for AI/ML systems that operates over standard best-effort Ethernet and existing software interfaces. MRC extends the reliable connection (RC) model with explicit multipath support, congestion control, path health tracking, and failure recovery mechanisms to maintain high goodput in the presence of congestion and link or path failures. The specification describes endpoint behavior, protocol state machines, control- and data-plane semantics, and the software APIs required to implement MRC in NICs, accelerators, and host stacks.

5.1. Description

MRC is a transport protocol designed to provide reliable, high-goodput connectivity for large-scale AI/ML workloads over standard best-effort Ethernet. MRC extends RC transport with explicit multipath operation, path health monitoring, and failure recovery so that endpoints can sustain effective throughput in the presence of congestion, link failures, and other fabric impairments.

Within the larger system stack, MRC sits between existing protocol layers and AI/ML frameworks or runtime libraries, exposing APIs that allow NICs, accelerators, and host software to utilize multiple network paths while preserving ordered, reliable delivery semantics where required.

MRC addresses problems such as underutilized network capacity, job failures or stalls due to transient fabric issues, and tight coupling between endpoint behavior and specific network topologies.

In the Open Compute Project ecosystem, MRC provides an open, implementation-ready specification that enables interoperable, multi-vendor AI/ML fabrics, improves effective accelerator utilization on commodity Ethernet infrastructure, and supports scalable, fault-tolerant deployments from small clusters to hyperscale systems.

MRC uses ECMP hashing or source-based routing for multipath load balancing. It achieves high transport efficiency through:

- Entropy-based rotation of request packets
- Responder-side tracking of out-of-order packets
- Direct placement of payload into target memory
- Distinct traffic classes for data, retransmissions, and control messages

Supported source-routing modes:

- Structured EV: Combines IPv6 flow label and UDP source port into a single entropy field for deterministic path selection from the sender to the spine switch

- SRv6: Uses uSID-based addressing with optional CSID encoded Segment Routing Header

For congestion control, reliability, and scalability, MRC provides:

- Independent Reliability SACKs: Separate from RDMA Transport ACKs, SACKs convey packet reception state and fabric and responder congestion signals for transport and CC algorithms
- Dynamic PSN Window: Responder can adjust the maximum in-flight packet range at runtime
- NSCC Congestion Control: Implements UltraEthernet sender-based algorithm with support for trimmed and untrimmed packet flows
- Reliability Probes: Allow requestors to query responder packet reception state
- Entropy Value Probes: Measure path health and latency across network paths
- Port Status Mask: Communicates local port reachability to peers

MRC exposes two APIs:

- Application API: Mirrors libibverbs semantics for creating and managing MRC queue pairs
- Control-Plane API: Enables administrators to configure per-QP multipath policies, adjust load-balancing dynamically and query path reachability

MRC connections implement the following core capabilities:

- Multipath Transmission: Distributes request and reliability control packets (SACKs) across multiple network paths using ECMP or source routing, with ECN marking support
- Reliability Control: Uses selective acknowledgments (SACKs) and negative acknowledgments (NACKs) to ensure timely packet delivery and convey congestion state
- WriteIMM Limiting: Enforces a user-defined maximum number of in-flight Write-with-Immediate operations per connection
- Congestion Control: Implements NSCC, a sender-based congestion control algorithm specified in the UltraEthernet Transport Specification [UESPEC]
- Selective Retransmission: Allows retransmission of specific packets based on SACK/NACK feedback.

Implementations MAY support the following optional features:

- Dynamic Maximum PSN Range (MPR): Enables responders to adjust the maximum number of outstanding request packets in-flight during connection runtime
- Trim NACK: Responders generate trim notifications when packets are truncated in the network
- Service Time: Responders report local request service time

Limitations of MRC Compared to RC:

- Restricted Operations: Supports only RDMA Write and Write-with-Immediate (WriteIMM); Read, Send, and Atomic operations are not supported

- WriteIMM Resource Bound: The number of in-flight WriteIMM operations is limited by responder-side tracking resources
- No RNR-NAK Semantics: RDMA transport-level Receiver Not Ready (RNR-NAK) flow-control is not supported

5.2. Goals, Visions and Objectives

The long-term vision of MRC is to provide an open, transport-level mechanism, inclusive of application and management APIs, that enables large-scale AI/ML systems to achieve high goodput, resilience to fabric impairments, and efficient accelerator utilization over standard best-effort Ethernet. MRC is intended to serve as a common foundation for interoperable, multi-vendor AI/ML fabrics within the Open Compute Project ecosystem, reducing dependence on proprietary transports and specialized networks.

The primary goals are:

- Reliable multipath operation over Ethernet: Provide a transport that supports multipath communication, maintains reliable delivery semantics where required, and continues to make progress in the presence of congestion, intermittent faults, and link or path failures
- High effective utilization of existing infrastructure: Increase usable network capacity and accelerator utilization without requiring changes to underlying physical layers or wholesale replacement of deployed Ethernet fabrics
- Interoperability and openness: Enable consistent endpoint behavior, control-plane integration, and observability across NICs, accelerators, and switches from different vendors, using a publicly available, implementation-ready specification
- Operational scalability: Support deployments ranging from small clusters to hyperscale fabrics with minimal operator intervention through well-defined control-plane APIs and integration hooks for monitoring, orchestration, and automated remediation

Measurable objectives guiding the design and evolution of MRC include:

- Improving realized goodput and network utilization for representative AI/ML workloads compared to single-path reliable transports operating over the same Ethernet fabric
- Reducing job failures, stalls, or restarts attributable to transient or localized fabric impairments
- Demonstrating interoperable implementations across multiple independent vendors and system configurations
- Enabling incremental, non-disruptive deployment of MRC endpoints into existing Ethernet-based environments

6. Transport Extensions

MRC modifies RC transport as follows:

6.1. MRC QP Properties

The following per-QP MRC properties are defined:

Table 6-1 QP Dimensions

Property	Units	Comments
max_psn_range (mpr) Maximum number of inflight packets tracked by the responder request tracking bitmap	128 packets	This dimension is propagated from responder to requestor on QP initialization
max_wimm_inflight Maximum number of inflight WriteIMM requests tracked by responder	Number of inflight WriteIMM requests	This dimension is propagated from responder to requestor on QP initialization

6.2. Requestor Message Processing

6.2.1. Work Request Spraying

MRC relies on switch level ECMP hashing or source routing to route packets through different paths in the network.

MRC WRs MUST be completed at the requestor in posted order.

6.2.2. MRC Supported Operations

MRC opcode [7:5] is 0b110.

MRC header stacking is:

Table 6-2 MRC Supported Operation Opcodes & Header Stacks

Opcode[7:0]	Description	Headers Following BTH
0xC6	RDMA WRITE First	METH, [TSETH], RETH, PayLd
0xC7	RDMA WRITE Middle	METH, [TSETH], RETH, PayLd
0xC8	RDMA WRITE Last	METH, [TSETH], RETH, PayLd
0xC9	RDMA WRITE Last with Immediate	METH, [TSETH], RETH, ImmDt, PayLd
0xCA	RDMA WRITE Only	METH, [TSETH], RETH, PayLd
0xCB	RDMA WRITE Only with Immediate	METH, [TSETH], RETH, ImmDt, PayLd
0xD1	Acknowledge	AETH

0xD8	Endpoint Request	ERTH
0xD9	Endpoint Response	EETH
0xDC	Reliability SACK	SETH, CC_STATE
0xDD	Reliability NACK	NETH
0xDE	Reliability PROBE Request	PETH

6.2.2.1. BTH Changes

The standard BTH header is modified as follows:

- An RTX field is added to indicate the packet is a retransmit
- A TSETH field is added to indicate the existence of the TSETH header
- The following table depicts the RoCE BTH header used for MRC reliability messages.

A valid iCRC MUST be calculated per IBTA specification.

The type of reliability header in the BTH payload is defined based on the OPCODE.

Table 6-3 RoCE BTH Header

Byte 0					Byte 1				Byte 2	Byte 3
Opcode=MRC Opcode							pad	tver=0	p_key	
var-res					dqp					
a	R	rtx	ts	inv-res	psn					

Table 6-4 Changed or Modified RoCE BTH fields

Field	Size (bits)	Description
opcode	8	MRC Data header opcodes from Table 6-2
dqp	24	Value depends on packet type (see Table 7-6)
R	1	reserved
rtx	1	Set if the packet is a retransmission
ts	1	Set if the packet includes the TSETH header
inv-res	4	Invariant reserved
psn	24	Value depends on packet type (see Table 7-6)

6.2.2.2. RETH Changes

For Write requests, the RETH header semantics are modified relative to the definition in [IBTASPEC] to provide sufficient information for direct placement of the packet payload as follows:

Table 6-5 RETH Fields

Field Abbreviation	Description

va	Base virtual address to deliver the payload for this packet. Updated per packet for multiple packet messages.
r_key	Remote key that authorizes access for this packet. Constant across all packets for a given message.
dmalen	The length (in bytes) for the entire DMA operation (as defined by the WR)

Messages MUST be segmented into PMTU size packets except for the last or only packets per IBTA rules. The VA address MUST be updated (incremented by the PMTU) to allow for proper placement in the memory region at the responder in the case of multiple packet messages.

Note: this is different than standard IBTA behavior since the RETH is included for all packets of the write. MR boundary checks MUST be performed.

6.2.2.3. TSETH

All requests with the BTH.tseth bit set contain a Requestor Timestamp Extended Header (TSETH) defined as follows:

Table 6-6 TSETH Header

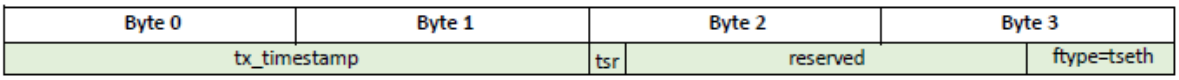

Table 6-7 TSETH Header Fields

Field	Width (b)	Description
tx_timestamp	16	Requestor Timestamp. Field resolution is implementation defined.
tsr	1	Timestamp resolution: 0x0 = 128ns 0x1 = Implementation defined (should not be interpreted or used by responder)
reserved	11	Reserved
ftype	4	TSETH type, ftype=0x1 if the tx_timestamp is carried

This header's tx_timestamp field is reflected in the Reliability SACK packet allowing the requestor to calculate the request RTT. The tx_timestamp SHOULD be collected at a point during transmission which minimizes jitter. Implementations SHOULD transmit timestamped packets within 0.5us of timestamping. The HW time clock SHOULD have a resolution of at least 128ns. If tsr is 0x0, the responder MAY interpret and use the tx_timestamp field.

6.2.2.4. METH

All request packets must include the Message Extended Transport Header (METH) defined as follows:

Table 6-8 METH Header

Byte 0	Byte 1	Byte 2	Byte 3
RQMSN		MSN	

Table 6-9 METH Header Fields

Field	Width(b)	Description
RQMSN	16	Receive Queue MSN. Incremented for every WriteIMM message and carried in every packet of the message. Unused and undefined in other message types.
MSN	16	Requestor MSN

Implementations may use MSN, RQMSN or both fields to track in-flight WriteIMM requests.

6.2.3. Loss Detection & Retransmission

Support for loss-detection and retransmission is defined as follows:

Table 6-10 MRC Loss Detection and Retransmission

Mechanism	Required Support	Comments
Logical timer	MUST	May be per-packet or per-connection
Fast-loss Detection and Retransmit	MUST	Algorithm is implementation defined
TRIM packet	MUST	Support for receiving UET TRIM packets [UESPEC] is mandatory; generating TRIM NACKs is optional

6.2.4. Requestor Error Behavior

MRC requestor error handling differs from RC requestor error behavior (as defined in [IBTASPEC]) as follows:

Table 6-11 Requestor Error Behavior

Error	Syndrome	Fault Behavior Class
Packet sequence error. Retry limit not exceeded	Not applicable	Not applicable
Packet sequence error. Retry limit exceeded	Not applicable	Not applicable
Implied NAK sequence error. Retry limit not exceeded	Not applicable	Not applicable
Implied NAK sequence error. Retry limit exceeded	Not applicable	Not applicable
Local Ack Timeout error. Retry limit not exceeded.	Same as RC	Same as RC

Local Ack Timeout error. Retry limit exceeded.	Same as RC	Same as RC
RNR NAK Retry error. Retry limit not exceeded.	Not applicable	Not applicable
RNR NAK Retry error. Retry limit exceeded.	Not applicable	Not applicable
Unsupported Opcode	Same as RC	Same as RC
Unexpected Opcode	Same as RC	Same as RC
Local Memory Protection Error	Same as RC	Same as RC
R_Key Violation	Same as RC	Same as RC
Remote Operation Error	Same as RC	Same as RC
Local Operation Error - WQE	Same as RC	Same as RC
Local Operation Error – affiliated or unaffiliated	Same as RC	Same as RC
Length error	Same as RC	Same as RC
Bad response	Same as RC	Same as RC
Ghost Acknowledge	Same as RC	Same as RC
CQ overflow	Same as RC	Same as RC

MRC introduces the following additional requestor error behaviors:

Table 6-12 MRC Additional Requestor Errors

Error	Syndrome	Fault Behavior Class
Resources Unavailable	NAK-Remote Operational Error	Requestor Class B
Inflight WriteIMM limit exceeded	NAK-Invalid Request	Requestor Class B
Retry Counter Exceeded	Locally detected	Requestor Class B

“Resources Unavailable” and “Inflight WriteImm limit exceeded” NACKs are generated by the responder (Section 6.3.5). “Retry Counter Exceeded” is generated when the requestor reaches the QP retry limit due to receiving reliability NACKs (Section 7.5.5.6).

6.3. Responder Message Processing

Responders MUST be able to:

1. Receive, track and process request packets OOO across message boundaries
2. Track and stash Immediate Data for WriteIMM messages while the message is incomplete
3. Generate and transmit RDMA Transport ACKs, SACKs and NACKs

6.3.1. Received Requests

Received packets undergo the standard RC validation checks and processing flow defined in [IBTASPEC].

Received Write and WriteIMM requests are self-describing; every packet has sufficient information that its payload can be directly placed in memory without requiring the responder to derive the packet's offset within its associated message.

RC Inbound request packet validation rules are modified as follows:

1. The responder MUST maintain a logical ePSN, where ePSN = cack_psn + 1, representing the next expected PSN for cumulative acknowledgment
2. Packets arriving out of order due to multipath spraying MUST NOT be discarded solely because their PSN does not equal ePSN
3. Any packet with a PSN within the responder's tracking window, defined as [ePSN, ePSN + max_psn_range), MUST be considered valid and processed according to reliability rules in Sec. 7
4. If a packet is accepted and meets the IBTA-defined criteria for ACK/NACK generation, the responder MUST issue a Transport ACK as specified in [IBTASPEC]
5. Already received and processed packets in the range (ePSN – 2^{23}, ePSN) and [ePSN, ePSN + max_psn_range) MUST be treated as valid duplicates and acknowledged with a Transport ACK and Reliability ACK as appropriate
6. Packets outside the tracking window (PSN < ePSN – 2^{23} or PSN >= ePSN + max_psn_range) MUST be silently dropped

6.3.2. Placement, Ordering and Completion

Write and WriteIMM packet payload SHOULD be placed OOO in destination memory.

The logical rules governing ordering and completion are:

1. WriteIMM completions are deferred until all previous requests have been received and processed
2. WriteIMM completions are delivered at the responder in requestor send order

The complete placement and completion ordering guarantees for MRC operations are as follows:

Table 6-13 Placement and Ordering Guarantees

First Operation	Second Operation	Placement Guarantee at Responder	Placement Guarantee at Requestor	Ordering Guarantee at Responder
Write	Write	No placement guarantee	Not applicable	Not applicable
Write	WriteIMM	No placement guarantee	Not applicable	WriteIMM MUST be completed after data has

				been placed for both Write operations.
WriteIMM	Write	No placement guarantee	Not applicable	Not applicable
WriteIMM	WriteIMM	No placement guarantee	Not applicable	Completed in order at responder

6.3.3. WriteIMM Resource Management

Every MRC QP allocates resources to track and stash WriteIMM Immediate Data (ImmDt) values that may be received OOO. The number of resources per-connection are in the range of the max_wimm_inflight dimension. As WriteIMM requests are received, the ImmDt values MUST be stashed in these resources until a responder completion is generated for the request.

It is the responsibility of software (e.g. *CCL) to ensure the number of inflight WriteIMM requests does not exceed the outstanding operations advertised by the responder.

MRC QPs do not support RNR-NAK retry semantics. If a WriteIMM completion cannot be delivered to the responder RQ due to lack of a descriptor, the responder QP is transitioned to error state, and a NAK-Remote Operational Error is sent to the requestor.

6.3.4. RDMA Transport ACKs and Reliability SACKs

Reliability SACKs are logically separate from Transport ACKs and are generated by the responder when a mandatory condition for SACK generation is met. SACKs augment Transport ACKs by providing responder bitmap state but are mainly used to support fast detection and retransmission of lost request packets as well as informing the CC algorithm of network and responder state. SACKs are covered in more depth in Sec. 7.4.5.

6.3.5. Responder Error Behavior

MRC responder error handling differs from RC responder error behavior (as defined in [IBTASPEC]) as follows:

Table 6-14 Responder Error Behavior

Error	Syndrome	Fault Behavior Class
Malformed WQE	Same as RC	Same as RC
Unsupported or Reserved OpCode[2]	Same as RC	Same as RC
Misaligned ATOMIC	Not applicable	Not applicable
Too many RDMA READ or ATOMIC Requests	Not applicable	Not applicable

[2] Includes opcodes for unsupported MRC operations

Out of Sequence Request Packet	Not applicable	Not applicable
Out of Sequence OpCode, current packet is “first” or “only”	Not applicable	Not applicable
Out of Sequence Opcode, current packet is not “first” or “only”	Not applicable	Not applicable
R_Key Violation	Same as RC	Same as RC
Local QP Error	Same as RC	Same as RC
Packet Header Violation	Same as RC	Same as RC
Resources Not Ready Error	Not applicable	Not applicable
Length Errors 1) Inbound “Send” request message exceeded the responder’s available buffer space: “Local Length Error” 2) RDMA WRITE packet payload contained too much or too little payload data compared to the DMA length advertised in the current packet3 3) Payload length was not consistent with the opcode: a: 0 byte <= “only” <= PMTU bytes b: (“first” or “middle”) == PMTU bytes c: 1byte <= “last” <= PMTU bytes 4) Inbound message exceeded the size supported by the CA port	1) Not applicable 2) Same as RC 3) Same as RC 4) Same as RC	1) Not applicable 2) Same as RC 3) Same as RC 4) Same as RC
Invalid duplicate ATOMIC Request	Not applicable	Not applicable
CQ Overflow	Same as RC	Same as RC
Remote Invalidate Error	Not applicable	Not applicable

MRC introduces the following additional responder error behaviors:

Table 6-15 MRC Additional Responder Errors

Error	Syndrome	Fault Behavior Class

[3] This condition is modified compared to [IBTAV18] which compares the RDMA WRITE request message length with the DMA length advertised in the first or only packet.

Inflight WriteIMM limit exceeded	NAK-Invalid Request	Responder Class C
Resource Unavailable	NAK-Remote Operational Error	Responder Class A

“Inflight WriteIMM limit exceeded” NAKs are generated when there are no per-QP resources to stash received ImmDt entries. “Resource Unavailable” NAKs are generated when there are no destination semantic resources (e.g. RQ buffers) available to fulfil the requestor operation.

6.4. End-to-end Flow Control

In MRC, end-to-end flow control of WriteIMM requests is managed by the application. MRC QPs do not support or advertise message level flow control credits. Implementations MUST always populate the AETH header flow control credit field with the value 0x1F, indicating end-to-end flow-control is not supported.

6.5. Probes

MRC supports requestors probing paths in the network for reliability. Two probe types are defined:

6.5.1. Reliability Probes

Reliability Probes are part of the normal connection-oriented control flow between a Requestor and Responder. A Reliability Probe request is encoded using a PETH header and is addressed to the peer QP. The response is a SACK generated by the Responder (Sec. 5.4.2). Reliability Probe requests and responses are best-effort control messages and do not consume PSNs.

6.5.2. EV Probes

EV Probes are Endpoint Operations (Sec. 7.3) used to determine forward-path reachability and return-trip latency. EV Probes may be transmitted on any traffic class. Requestors match EV Probe responses to outstanding requests using requestor-private identifiers. Return-trip latency is calculated at the requestor.

6.6. Port Status Updates

Port Status Updates are Endpoint Operations (Sec. 7.3) that allow a node to communicate the operational state of its local ports to a peer.

7. Reliable Delivery

7.1. Multipath Spraying

One of the key features of the specification is supporting multiple paths from source to target and distributing (spraying) packets across these paths. The mechanism to distribute across these paths is implementation specific. Entropy is used throughout this specification to specify a field used to enable different packets from a QP to take different paths in the network. The size and format of the entropy field varies based on the network configuration (Sec. 9—9.2).

Control Messages (SACK/NACK) include a dedicated entropy field that mirrors the forward-path entropy of the triggering packet. This facilitates requestor path skipping (Sec. 9.3.1).

7.2. Reliability SACKs and NACKs

MRC separates semantic processing from packet delivery through Reliability control packets: SACKs and NACKs. Reliability SACKs/NACKs are transmitted from responder to requestor and provide packet delivery and congestion control state. They are formatted as SETH (Sec. 7.5.5.5) and NETH (Sec. 7.5.5.6) control packets.

Key information provided by Reliability SACKs is:

- Requestor and responder QPIDs
- Cumulative ACK value
- Forward-path and responder congestion state
- Bitmap resources available at responder
- Indication of which packets spatially close to the acknowledged packet have been received
- Reflected request timestamp

Key information provided by reliability NACKs is:

- Requestor and responder QPIDs
- Forward-path and responder congestion state
- Reliability layer NACK error code (Sec. 7.4.3)
- Reflected request packet timestamp

Reliability SACKs and NACKs are logically independent from RDMA transport ACK and NAK control packets. It is valid for an RDMA request packet to be received at the responder and reliability SACKed but semantically NAKed.

7.2.1. Common State

Implementations MAY infer the semantic-layer reception state of request packets from information carried in Reliability control packets. However, implementations MUST NOT infer

the semantic-layer processing outcome of request packets from Reliability control packets and MUST instead obtain that outcome from the appropriate transport-layer control packets.

Implementations MAY infer reliability-layer reception and (successful) processing outcome of request packets from the corresponding transport-layer control packet.

7.3. Endpoint Operations

Endpoint operations are defined as connectionless, best-effort request-response control primitives that operate at node scope rather than connection scope. They enable an endpoint to exchange reachability, telemetry, or status information with a peer without consuming PSNs or relying on QP-level semantics.

Endpoint operations MUST NOT assume reliable delivery, ordering, or retransmission guarantees. Supported Endpoint operations are:

- EV Probes: report path reachability and latency
- Port Status Update: communicates the operational state of local ports

Endpoint requests differ from connection-scope requests as follows:

- BTH.QPN MUST be the reserved value 0x2
- BTH.PSN[15:0] is a requestor-private identifier
- May be transmitted on Data or Control Traffic Classes depending on the type of operation

Endpoint responses differ from connection-scope responses as follows:

- BTH.QPN MUST be the reserved value 0x2
- BTH.PSN[15:0] is populated from the corresponding request packet field

Responders that support service time SHOULD adjust requestor timestamps or report service time as specified in Sec. 7.5.5.8.

7.4. Requestor Requirements

7.4.1. Local ACK Timeout

Each RoCE Requestor MUST maintain a Local ACK Timeout timer for unacknowledged outstanding requests. This timer MAY be implemented as per-packet or per-QP and MUST be configurable as per-QP fidelity. Implementations MAY populate the timer with a default value.

Applications configure Local ACK timeout using a dimensionless timeout parameter; the timer value is derived from this parameter and the retry iteration as provided in in Table 7-1.

7.4.2. Retry Count

MRC QPs implement a per-QP request retry counter composed of two mandatory components: a linear and an exponential retry count. When a packet is transmitted, the appropriate packet or QP timer is primed with a fixed value for up to linear retries after which the value is doubled for up to exponential retries. If an acknowledgement is not received within (linear+exponential) retries, the QP enters ERROR state unless exponential retry count is set to infinite retry.

Configuration bounds for retry count are:

Table 7-1 Timeout Calculation and Retry Count Bounds

Mode	Min Value	Max Value	Infinite Retry	Comments
Linear (linear_ack_to)	0	7	No	Local ACK Timeout value is: $(1.024 * 2^{timeout})$ us
Exponential	0	25	Yes	Local ACK Timeout value is: min (linear_ack_to * $2^{(i+1)}$, 1.024 * 2^{24}) us, where *i* is the current exponential mode retry iteration. 25 indicates infinite retry mode.

7.4.3. Entropy Generation

Requestors MUST adhere to the following rules when generating request packet entropies:

Table 7-2 Entropy Generation

Routing Mode	Populated Fields
UDP Source Port (ECMP)	IPv6.flow_id (if present) udp.sport
Structured EV	IPv6.flow_id (if present) udp.sport
SRv6	Outer header dstip field and SRH (if present)

7.4.4. Retransmit and NACK Processing

When a SACK message arrives, the ack-psn is calculated and bit map is processed. For packets which are selected for retransmission the retransmission flag (rtx) in the BTH header MUST be set.

For each packet that is newly acked, whether by selective acknowledgment or by cumulative ack, if that packet was previously marked for retransmission but has not yet been retransmitted, then it SHOULD not be retransmitted.

If the packet indicated by the NACK is retriable, the Requestor SHOULD retransmit until the configured retry limit is reached. The Requestor MAY skip retransmission if reliable state (e.g., responder bitmap) confirms the packet was already processed. If the packet indicated by the NACK is non-retriable or the retry limit is exceeded, the Requestor MUST transition the QP to ERROR state immediately.

Requestors MUST implement receipt and processing of all NACK codes defined in Table 7-3. Responders MUST support all codes marked “Mandatory”.

Table 7-3 Requestor NACK Handling Behavior

NACK Mnemonic	Responder Support	NACK Description	Requestor Behaviour
TRIMMED	Optional	Request packet was trimmed	Requestor retries, increasing retry counter. If max retries reached, QP enters Error state ("Retry Counter Exceeded").
TRIMMED_LASTHOP	Optional	Request packet was trimmed by the last-hop switch	Requestor retries, increasing retry counter. If max retries reached, QP enters Error state ("Retry Counter Exceeded").
NO_BITMAP	Mandatory	Packet tracking resource temporarily unavailable at responder	Requestor retries, increasing retry counter. If max retries reached, QP enters Error state ("Retry Counter Exceeded").
NO_PKT_BUFFER	Mandatory	Packet buffering resource temporarily unavailable at responder	Requestor retries, increasing retry counter. If max retries reached, QP enters Error state ("Retry Counter Exceeded").
NO_RESOURCE	Mandatory	Implementation specific resource temporarily unavailable at responder	Requestor retries, increasing retry counter. If max retries reached, QP enters Error state ("Retry Counter Exceeded").
PSN_OOR_WINDOW	Mandatory	PSN outside responder request tracking window	Requestor retries, increasing retry counter. If max retries reached, QP enters Error state ("Retry Counter Exceeded").
UNEXP_EVENT	Mandatory	Implementation specific unexpected event occurred processing packet	QP enters Error state (“Remote Operation Error”)

When a packet is retransmitted the SRC_TIMER for that packet MUST be reset to the original, default timer value (not exponentially increased). The entropy value from the NACK packet SHOULD NOT be used for the retransmitted packet.

Each transmitted RoCE packet MAY include the timestamp extension header carrying tx_timestamp field. When present, this timestamp MUST be returned in a SACK packet based on rules in Section 7.5.2.

A Requestor MUST support accepting a maximum bitmap size from the corresponding Responder. When supported, the max_psn_range field from the SACK packets is used to set the maximum PSN window in packets. That is, packets using a PSN larger than {unack_psn + max_psn_range} are withheld from transmission until the unacknowledged PSN advances.

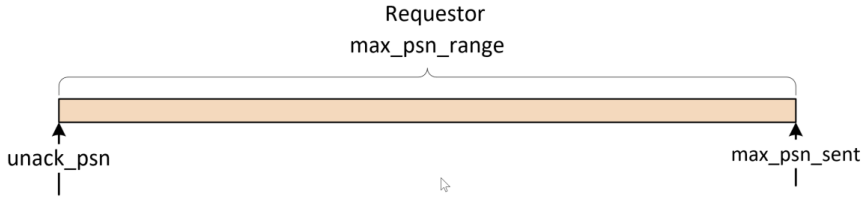

Figure 1 Requestor Bitmap

Retransmitted packets are subject to the same constraints and rules as first-time transmits (e.g. congestion window limitations). Retransmitted packets differ from first-time transmits only as follows:

- They MAY use a different DSCP for re-transmitted packets
- The BTH.RTX flag is set

Table 7-4 Requestor per Connection Reliability State

Field (per QP)	Size (bits)	Initial	Description
unack_psn	24	rq_psn-1	Lowest sequence number of the first hole on the requestor (left side of retransmission window).
max_psn_sent	24	0	Most advanced PSN sent by Requestor

7.4.5. SACK Based Loss Detection

Although spraying causes packets to be reordered, load balancing will generally bound the path differences to around one BDP. Therefore, assuming the cwnd has not been rapidly reduced, roughly cwnd bytes worth of packets will be sent each RTT.

cwnd is calculated by the congestion control algorithm described in section 8.

An implementation MUST provide a mechanism to avoid spurious retransmissions (re-transmitting packets that are queued in the network but not lost) and ensure that a packet is retransmitted at most once due to SACK based loss detection.

NOTE: Section 3.5.15 of UET 1.01 [UESPEC] specification provides additional discussion.

7.4.6. Reliability Probes

Reliability probes are transmitted as request packets without a PSN. Requestors can target specific paths by varying the request entropy value. When a Reliability Probe request arrives, the Responder MUST generate a SACK (SETH + CC_STATE) with the probe response flag (pr=1) set. Reliability Probes belong to the normal control flow of the QP and MUST follow reliability SACK rules (Sec. 7.5.2).

The BTH.PSN field for Reliability Probe Requests and associated SACK responses are considered reserved and MUST NOT be interpreted or reflected by the remote peer.

Each Reliability Probe request includes a unique identifier in the PETH.probe_id field. Responders MUST echo this identifier in the corresponding SACK by populating the ack_psn_offset field, allowing the requestor to match probe responses to outstanding probes.

Implementations SHOULD handle Reliability Probes in the fast path to minimize latency.

Reliability Probe usage is implementation-defined.

7.4.7. EV Probes

EV Probes are Endpoint Operations (Sec. 7.3) used to assess the reachability and latency of forward paths. Requestors construct and transmit EV Probe Requests (Sec. 7.5.6.1). Responders receive Requests and construct and transmit corresponding Responses (Sec. 7.5.6.2). Requests may be transmitted using any Traffic Class. Responses are always transmitted using the Control Traffic class.

Based on network routing configuration (Sec. 7.5.2.1), EV formats may be one of: UDP source port, Structured EV, SRv6 uSID, or SRv6 uSID with SRH. The responder uses the Probe Request's source IP address, which is the destination of the probe response, selects the EV format of the Response.

Requestors tag each outstanding request with a unique identifier in BTH.PSN[15:0]. Responders MUST echo this identifier in the corresponding reply to enable precise request–response matching.

Implementations MAY support EV Probes in compliance with the MRC Controller API (Sec. 10.2.2). Usage beyond the Controller API is implementation-defined.

7.4.8. Port Status Update

Port Status Updates are Endpoint Operations (Sec. 7.3) that communicate the operational state of responder ports to a peer. These updates enable the peer to make informed decisions about path selection, failover, and EV spraying in multi-plane environments.

To initiate a Port Status Update, the requestor constructs an Endpoint Request (ERTH) packet with op='Port Status Update' and BTH.PSN[15:0] carries a requestor-private identifier for matching the response. The ERTH header includes the port_status_mask bitmap, where each bit represents the reachability state of a local port.

Upon receipt, the responder generates an Endpoint Response (EETH) packet, echoing the identifier in BTH.PSN[15:0]. Both request and response are transmitted as best-effort, connectionless messages without consuming PSNs, and are sent on Control traffic class.

7.5. Responder Requirements

Each Responder MUST maintain state tracking for packets which have been received from the Requestor based on RoCE Packet Sequence Number (PSN) (e.g., bitmap). The packets may arrive out of order with respect to the PSN. This initial PSN (rq_psn/sq_psn) is negotiated during the RoCE queue pair modify (ibv_modify_qp) call moving the QP to RTS state.

Note: All math using the PSN MUST be unsigned modulo the size of PSN (24bit).

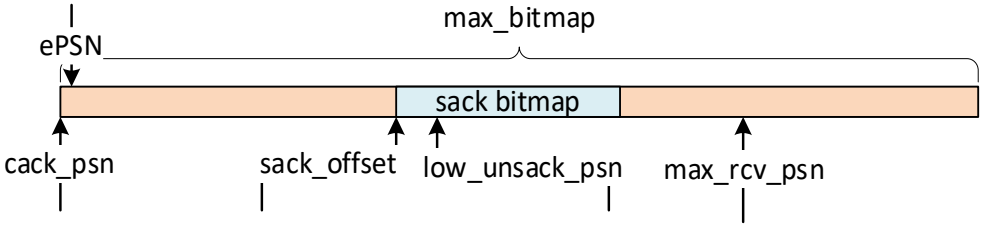

Figure 2 Responder bitmap

Each Responder MUST maintain the cumulative acknowledge PSN, called cack_psn, where cack_psn indicates that all packets with PSN prior to and including cack_psn have been received (i.e., no holes in bitmap before cack_psn). The initial value of cack_psn MUST be set to rq_psn-1 at connection start.

Each Responder MUST maintain the highest, non-trimmed, received PSN, max_rcv_psn and the lowest unsacked PSN, lowest_unsacked_psn. The lowest PSN which arrives but has not yet been SACKed is the lowest_unsacked_psn. lowest_unsacked_psn can be inside the range of the last SACK bitmap if the packet arrived after the last SACK was sent. Each Responder MUST maintain the highest PSN, max_rcv_psn received for packets which are not trimmed.max_rcv_psn MUST be initialized to the value rq_psn-1 at connection start.

A summary of the required state per QP is shown in the following table. Size is implementation specific as some PSN fields may be stored as relative to cack_psn, etc.

Table 7-5 Responder per Connection Reliability State

Field (per QP)	Size (bits)	Initial	Description
cack_psn	24	rq_psn-1	Cumulative ack; this is the highest sequence number received before the first hole (left side of bitmap).
rx_rcvd_bytes	32	0	Count of received bytes (bytes). Rcv_bytes field support 256B*2^24 bytes. If a smaller number of bits is required an implementation MAY store a smaller number of bits.
max_rcv_psn	24	rq_psn-1	Highest sequence number received (ignoring trims). AKA the right side of bitmap.
lowest_unsacked_psn	24	rq_psn-1	Lowest PSN which has not been SACKed at the responder
sack_trigger_cnt	24	0	Count of the number bytes/packets received since sending the last SACK
prev_ar	1	0	This flag triggers a SACK at the end of a message when reordering has occurred
max_psn_range (mpr)	n/a	0	Size of the bitmap supported on this QP, used to set a boundary for ACK'ing duplicate packets
rx_ooo_count	15	0	Out-of-order count

7.5.1. Responder Flow-control

The Responder can modulate the Requestor transmit rate using a receiver window penalty, rcv_cwnd_pen. This flow control is intended to be used if the Responder is itself congested and is not keeping up with the arrival rate of traffic. The rcv_cwnd_pen field is returned in the SACK and MUST be set to a value between 0 and 127. For example:

- Setting it to 0 will not flow control senders, i.e., this field has no effect.
- Setting it to 127 will slow all the senders by the maximum amount, down to a window of one packet per RTT.

- Setting it to 64 in all SACKs sent for an RTT will halve all senders' windows (cwnd) thus reducing the bandwidth to the responder

The restore_cwnd flag in the SACK indicates whether the sender should save the current value of cwnd when rcv_cwnd_pen indicates a period of responder flow control has started and restore the original cwnd value when responder flow control ends.

Responders MAY dynamically determine rcv_cwnd_pen depending on the degree of receiver congestion. The specifics of how the rcv_cwnd_pen is calculated is implementation dependent. The field may be set to 0 if it is not supported by the Responder.
The sender responds to non-zero values of rcv_cwnd_pen by modifying the congestion control response.

It MUST be possible to disable Responder Flow-Control on a per-QP basis.

The Responder Flow-Control capability described in this section is identical to UltraEthernet Destination Flow Control mechanism described in the UltraEthernet Specification [UESPEC]. Similarly, the way a requestor changes its transmission behavior in response to the receiver window penalty is congestion-control algorithm specific. Details on NSCC's use of the receiver window penalty is available in section 3.6.13.6 of the UEC Specification [UESPEC].

7.5.1.1. Address for Control Messages

For control messages (SACK, NACK) Control message addressing information determines the requestor QP. For these messages addressing information is populated as follows

- Responder QPID are taken from the packet BTH.DQP and copied into the Reliability Packet (Table 7-6) spdcid.
- Requestor QPID is determined from the QPID is copied from Queue Pair Context and copied into the BTH.DQP

7.5.2. SACK Generation

Each Responder SHOULD generate a SACK packet when a valid packet arrives which is not trimmed and any one or more of the following conditions are met:

- Cumulative SACK generation: when sack_trigger_cnt > sack_gen_threshold
- A packet arrives with RoCE BTH AR flag set (including requestor idle described below)
- A packet arrives with ECN marked indicating congestion.
- A packet arrives which was previously (re)transmitted (BTH.rtx set).
- A reliability probe packet arrives.

Cumulative SACK generation relies on maintaining a sack_trigger_cnt count per QP. The trigger counter is used to generate a SACK message based on the number of bytes or packets received by the QP. The sack_trigger_cnt counter is incremented by the larger of {packet

payload, MIN_ACK_PACKET_SIZE} and resets to zero when a SACK message is transmitted for any reason. The MIN_ACK_PACKET_SIZE defaults to 1KB.

To ensure that all packets are acknowledged at the end of the flow when a sender goes idle, a requestor MUST set the BTH.AR field. Note that reordering of the packet may cause the AR to arrive before other packets in the flow and the intent is to ensure that when all packets before the AR PSN are received a SACK is generated. A responder MUST implement an algorithm that ensures a SACK is generated after all packets before PSN of the AR are received. The below algorithm is an example implementation executed when a packet arrives.

```
if cack_psn is not equal to max_rcv_psn then
      store (prev_ar) that this occurred
if prev_ar is set and cack_psn becomes max_rcv_psn
      send a SACK
```

Responder MUST populate the max_psn_range (mpr) field in each SACK to indicate to the Requestor the maximum range of out-of-order packet tracking available for the QP. The field is encoded as an unsigned integer in units of 128 packets available. A value of zero indicates the field is unpopulated.

7.5.2.1. Request Entropy Reflection and Reliability and Transport S/ACK Entropy Generation

Responders MUST reflect the request packet's entropy fields—IPv6 flow label (if present) and UDP source port—in all reliability control messages (SACK and NACK). If a request omits any entropy-related field, the responder MUST set the corresponding field in the response to zero.

This rule applies uniformly across all routing modes: ECMP using UDP source port, Structured EV, and SRv6. The exact entropy value carried in SACK/NACK and Transport ACK packets is implementation-defined. Implementations SHOULD ensure that the entropy value in control packets corresponds to a currently valid path, including across planes where applicable. The mechanism for selecting entropy values is implementation-defined.

7.5.2.2. SACK Bitmap Calculation

The goal of the SACK bitmap is to efficiently acknowledge reception of packets at the Responder with a bias to acknowledging lower PSNs. When multiple SACK messages are coalesced by the Responder, the SACK bitmap is biased such that the requestor can free up the old resources. Lowest unacknowledged PSN, lowest_unsacked_psn, keeps the bitmap on the lower side to avoid missing received packets while also attempting to rotate across the bitmap when the arriving PSNs vary across a large range.

As packets arrive which do not cause a SACK to be generated the lowest_unsacked_psn is updated as follows:

```
if cack_psn advances then
        if cack_psn >= lowest_unsacked_psn then
                lowest_unsacked_psn = cack_psn
else if pkt.PSN < lowest_unsacked_psn then
        lowest_unsacked_psn = pkt.PSN
```

When a SACK is generated, the anchor of the SACK bitmap, sack_base is calculated as follows:

```
sack_base = lowest_unsacked_psn
        if sack_base+64 >= max_rcv_psn then
                sack_base = max(max_rcv_psn–64, cack_psn)
gen_sack(sack_base, cack_psn)
lowest_unsacked_psn = sack_base+64
```

The rationale here is that if the arriving packet causes a SACK to be generated, then lowest_unsacked_psn does not need to be decreased, because the arriving packet will be SACKed using acked_psn, even if it is not covered by the SACK bitmap. Thus, if a SACK is generated, lowest_unsacked_psn will always advance.

If a SACK is not to be generated for the arriving packet, lowest_unsacked_psn may need to be decreased. If the cumulative ACK is advanced, this happens because pkt.PSN fills the lowest hole in the sequence space, so in this case lowest_unsacked_psn does not need to be reduced because pkt.PSN is now covered by the cumulative ACK. Only if the cumulative ack is not advanced, pkt.PSN is lower than the current lowest_unsacked_psn, and a SACK is not to be immediately generated, does lowest_unsacked_psn need to be decreased.

This approach leaves lowest_unsacked_psn unchanged if a single lower PSN arrives and either triggers a SACK or causes the cumulative ack to advance. This typically happens when a late packet such as a retransmission arrives. It also rotates up by 64 to cover the next 64 bits in the next SACK when lower PSN do not arrive.

Note: sack_base is a temporary field and used to calculate the sack_offset in the SACK message.

Responder MUST populate the SACK bitmap with 64-bit of state using a starting (lowest) number PSN based on the SACK_BASE. The sack_offset is calculated as a 2’compliment signed 16b subtraction:

- sack_offset = sack_base – cack_psn

Note: That valid range of sack_base is [cack_psn, max_psn_range-64].

For example, a SACK is generated and then 4 more packets arrive with PSN: {673, 704, 705, 680} which trigger a SACK. Given a cack_psn = 405 and the lowest_unacked_psn is greater than 673 when the four packets arrived. In this case, lowest_unsacked_psn= 673 since this is the lowest PSN that arrived since the last SACK was generated and the sack_offset would equal 268.

The PSN of the message which caused the SACK to be generated is represented as the ack_psn which is copied directly from the packet. The SACK messages use a compressed offset from the cumulative ACK using the following 16b 2'compliment representation:

- ack_psn_offset = pkt.psn – cack_psn

The out of order count, rx_ooo_count, is the number of received packets in the bitmap from the cumulative ACK, cack_psn to the max_rcv_psn. The value of rx_ooo_count MUST be returned in the ooo_count field in the CC_STATE of SACK Messages.

Note this is similar to "SACK Bitmap – Coalesced ACKs" section 3.5.12.4.3 of the UET 1.01[UESPEC] specification.

7.5.2.3. SACK Message

The reliability SACK message uses a SACK reliability message packet format described in 7.5.5.5. The entropy value for the SACK MUST be copied from the packet causing the SACK. If the original packet includes a tx_timestamp it is copied (reflected) into the SACK message. The RTX field in the SACK message is copied from the RTX field of the packet causing the SACK. The m field is set if the packet was ECN marked. 7.5.5.5

7.5.3. Trimmed Packet Processing and NACK Generation

Trimming is a way for a switch to signal congestion information to the responder using only the header information from the original packet. This is like ECN, except the trimmed packet is sent at higher priority and includes the request packet headers.

UET 1.01 section 4.1 describes how switches implement packet trimming. DSCP code points are used to indicate to the switches if a packet is trimmable (DSCP_TRIMMABLE) and how a trimmed packet is indicated to a receiving NIC (DSCP_TRIMMED or DSCP_TRIMMED_LAST_HOP). In addition, the MIN_TRIM_SIZE describes the minimum frame that a packet is trimmed to. For MRC, this size SHOULD be set large enough to contain the BTH and TSETH and include all L2-L4 encapsulation.

Responders MUST detect trimmed packets and process them strictly according to reliability rules. Under no circumstances SHALL a responder attempt payload placement or semantic processing for a trimmed packet.

The following mandatory events MUST cause a NACK message to be generated back to requestor.

- Error in processing bitmap. In this case the nack_reason = PSN error (PSNE).
- When a packet is dropped due to limited resources (bitmap, buffer, etc.). The nack_reason MUST be populated based on specific the resource: No receiver bitmap (NO_BITMAP) or no receiver buffer resource (NO_PKT_BUFFER).
- When a packet is dropped for other reasons a general responder error MAY be used.
- All other errors at receiver which can identify the sender should use nack_reason = general error (GERR).

Support for generating TRIMMED and TRIMMED_LASTHOP NACKs (Sec. 7.4.4) is an optional capability as follows:

Capability negotiation:

- Devices report support for generating TRIMMED/TRIMMED_LASTHOP NACKs to application SW
- On a per-connection basis, both nodes set a SW modified QP attribute indicating whether the peer supports generation of TRIMMED/TRIMMED_LASTHOP NACKs.
    - This capability is direction specific – either one or both ends may support it

Trim NACK generation unsupported:

- Requestors MUST handle NACKs for all mandatory events as stated previously.
- Requestors MAY apply implementation-defined optimizations to reduce packet loss and improve loss detection (e.g. mark packets as non-trimmable or sending periodic Reliability Probes)
- Responders MAY use implementation-defined techniques to signal state promptly (e.g. send a SACK when a trimmed packet is received)

Trim NACK generation supported:

- Responders MUST generate a NACK based on the following events:
    - When a trimmed packet is received based on DSCP= DSCP_TRIMMED, the nack_reason MUST be set to TRIM. This DSCP MUST be configurable.
    - When a trimmed last hop packet is received based on DSCP= DSCP_TRIMMED_LAST_HOP, it MAY be treated as a regular TRIM. The nack_reason MAY be set to TRIM_LAST_HOP if supported. This DSCP MUST be configurable if supported.
- Responders SHOULD generate a reliability NACK response packet as soon as practical using the reliability NACK message format. The nack_psn is the PSN of the packet which caused the NACK. The rtx field in the NACK is copied from the rtx field of the packet causing the NACK.
    - See trimming section (3.5.15.1) of UET 1.01 [UESPEC] specification for a definition of the TRIM packet format and end-to-end trimmed packet flows
- Requestors MUST be able to receive and process TRIM/TRIM_LASTHOP NACKs
- Requestors MUST retransmit packets indicated by TRIM/TRIM_LASTHOP NACKs as outlined in Sec. 7.4.4

7.5.4. Dynamic Maximum PSN Range

Dynamic MPR is an optional capability that allows a Responder to modify the maximum number of outstanding request packets (MPR) in-flight between a Requestor and Responder while the Responder QP is in RTR or RTS state.

- Capability negotiation:
  - On a per-connection basis, both nodes set a SW modified QP attribute indicating whether the peer node supports Dynamic MPR
  - Dynamic MPR is only enabled if both peers support the capability
- Dynamic MPR disabled:
  - The Responder MUST set SACK.mpr = 0
  - The Requestor MUST NOT validate or act on the SACK.mpr field
- Dynamic MPR enabled:
  - The Responder MUST include a non-zero mpr value in every SACK
    - If the Requestor receives a SACK with mpr = 0 it SHOULD notify SW (mechanism is implementation-defined)
    - The Responder MAY change this value at any time; change policy is implementation-defined
  - The Requestor MUST parse the SACK.mpr value in every received SACK and if SACK.cack_psn > QP.cack_psn and SACK.mpr != QP.mpr the Requestor MUST adjust its Requestor MPR resources to match the new value
  - The Requestor MUST enforce the invariant: No packet with PSN > (QP.cack_psn + QP.mpr) is transmitted. If the Requestor receives an MPR value that causes the Requestor to violate this invariant it MUST:
    - Pause transmission of new packets until the QP.cack_psn until the invariant is met.
    - Delay retransmission of unacknowledged or NACKed packets until they are in range.

7.5.5. Reliability Packet Formatting

The reliability layer requires sending congestion and reliability between requestor and responder. This contrasts with RoCE which only communicates CC information from responder to requestor. This solution extends the BTH opcode number space to add single packet reliability messages in both directions. In both cases, the reliability message is contained as a payload for a BTH header. The encapsulation uses the RoCE rules for CNP packets (IB A10.3.1). The Ethernet, IP and UDP headers are populated according to RoCE rules.

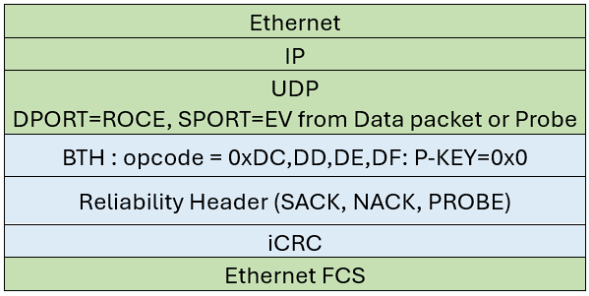

Figure 3 Reliability Packet Stack

Table 7-6 Reliability Header Field Population Summary

Packet Type	Entropy Value	BTH			SETH/NETH			
		opcode	qpn	psn	cack_psn	ack_psn	spdcid	dpdcid
RoCE ACK	RspGen[1]	0xD1	dpqn	cack_psn	–	–	–	–
SACK	RspGen[1]	0xDC	dqpn	cack_psn	cack_psn	pkt.psn	sqpn[4]	dqpn[5]
NACK	RspGen[1]	0xDD	dqpn	pkt.psn	0	pkt.psn	sqpn[4]	dqpn[5]
Rel. Probe Req.	ReqGen[6]	0xDE	dqpn	– [2]	–	–	–	–
Rel. Probe Resp.	RspGen[1]	0xDC	dqpn	–	cack_psn	–	sqpn[4]	dqpn[5]
EV Probe Req.	ReqGen[6]	0xD8	0x2	probe_id[3]	–	–	–	–
EV Probe Resp.	RspGen[1]	0XD9	0x2	probe_id[3,7]	–	–	–	–
Notes	1. Responder-generated EV 2. Reliability Probe requests do not consume PSNs 3. probe_id is 16 bits and maps to psn[15:0] 4. spdcid is populated with the source QPN (qpn[15:0]) 5. dpdcid is populated with the destination QPN (qpn[15:0]) 6. Requestor-generated EV 7. Responder reflects probe_id from corresponding request							

7.5.5.1. Class of Service Mapping

MRC is optimized to run over a best-effort network with at least two traffic classes. A control class is designed to carry control messages (SACK, NACK, and Rel Probes) and SHOULD be configured as a high-priority, best-effort class without trimming or WRED.

One or more data classes MAY be used to carry bulk data. At least one data class MUST be supported. Data traffic classes SHOULD be configured as best-effort and MAY support trimming. ECN marking SHOULD be configured on data traffic classes.

The following table describes the DSCP code points used in the specification. DSCP code points for data traffic are mapped through to service levels (SL) using the MRC verbs APIs.

Table 7-7 DSCP Codepoint definitions

DSCP Nonmonic	Support	Description
DSCP_NO_TRIM	Required	This code point is used when trimming is not supported.
DSCP_TRIMMABLE	Optional	Default if trimming is used
DSCP_TRIMMABLE_RETX	Optional	This class is used for retransmitted data with or without trimming. Note that BTH.rtx also includes a retransmission indication
DSCP_TRIMMED	Required	Used to indicate that a trimmable packet has been trimmed.
DSCP_TRIMMED_LASTHOP	Optional	Used to indicate that a trimmable packet has been trimmed at the last hop before reception.
DSCP_CONTROL	Required	Carries control traffic

The following table depicts the DSCP code points used for various packet types.

Table 7-8 DSCP and Traffic Class Mapping

Packet Type	Traffic Class	DSCP Code Points
Responder Control (SACK, NACK, RoCE ACK)	High	DSCP_CONTROL
Requestor Control (Reliability Probe request)	Data	DSCP_TRIMMABLE
Requestor Data (trimmable)	Data	DSCP_TRIMMABLE
Requestor Data (not trimmable)	Data	DSCP_NO_TRIM
Requestor Data Retransmission (trimmable or not)	Data	DSCP_TRIMMABLE_RETX
Trimmed Packet	High	DSCP_TRIMMED, DSCP_TRIMMED_LASTHOP
EV Probe Req	Any	Any
EV Probe Resp	Any	Any

7.5.5.2. UDP Header Considerations

The standard UDP destination port is used to indicate RoCE and MUST be configurable with a default value of 4971. The UDP checksum in MUST be set to zero (no UDP checksum). The source port MAY be populated via EV selection. For trimmed packets, the length field is used to signal the original length of the packet before it was trimmed.

Table 7-9 UDP Header

Byte 0	Byte 1	Byte 2	Byte 3
sport		dport=4791(default)	
length		checksum=0	

7.5.5.3. RoCE BTH Header For Control Packets

The following table depicts the RoCE BTH header used for MRC reliability messages.

Table 7-10 RoCE BTH Header for Control Messages

Byte 0					Byte 1				Byte 2	Byte 3
Opcode=MRC Opcode					m	se	pad	tver=0	p_key	
var-res					dqp					
a	R	rtx	tsh	inv-res	psn					

Table 7-11 RoCE BTH fields for SACK/NACK/Reliability Probe REQ

Field	Size (bits)	Description
opcode	8	MRC opcode: value depends on packet type (Sec. 6.2.2)
m	1	Unsupported field from IB BTH. Set to zero, ignore on recv
se	1	Unsupported field from IB BTH. Set to zero, ignore on recv
pad	2	Unsupported field from IB BTH. Set to zero, ignore on recv
tver	4	Version = 0
p_key	16	Provisioned reserved value to indicate a Reliability header is present
var-res	8	Variant reserved field - Unsupported field from IB BTH. Set to zero, ignore on recv.
dqp	24	Destination QP (requestor QP)
a	1	Ack Request
R	1	reserved
rtx	1	Set if the packet is a retransmission
tsh	1	Set if the packet includes the TSETH header; only valid for request packets
inv-res	4	Invariant reserved field - Unsupported field from IB BTH. Set to zero, ignore on recv.
psn	24	Cumulative PSN for SACK

7.5.5.4. CC State

The congestion control state (CC) is carried in SACK messages after the SETH header. MRC permits different CC state information to be defined. In this version of MRC, CC_TYPE=0x0 in the SACK header indicates the format below is being used.

Table 7-12 CC_STATE Information (CC_TYPE=0x0)

Byte 0		Byte 1	Byte 2		Byte 3
tx_timestamp			r	ooo_count	
r_c	rcv_cwnd_pen	rcvd_bytes			

Table 7-13 CC_STATE Fields

Field	Size (bits)	Description
tx_timestamp	16	Reflected request timestamp or reported responder service time
r	1	Reserved
ooo_count	15	Out of order count (from SACK/NACK generation)
r_c (restore_cwnd)	1	Set to 1 (true) if sender should restore cwnd after flow control
rcv_cwnd_pen	7	Receiver C-window penalty from Receiver Congestion calculation
rcvd_bytes	24	From SACK/NACK Generation logic (256B units)

The tx_timestamp field is used to measure the network round trip time. Increasing round trip time implies queues are growing in the network, and MRC uses this as a sign of congestion. Use is described in Section 8.4

The rcvd_bytes field is a cumulative count of bytes received at the Responder. It is derived from the rx_rcvd_bytes state at the Responder, but is in units of 256B, rounded up. It is used to drive the ack-clock for congestion control. Use is described in Sec. 8.3.1.

rcv_cwnd_pen is used by the Responder to request that the Requestor reduces its congestion window because the Responder is becoming backlogged. restore_cwnd indicates whether the Requestor should restore the original value of the congestion window when flow control ends. Use is described in Section 9.

ooo_count is a count of the number of received packets above the cumulative ack value. MRC specifies how this field is set, but its use is left for implementations. Generally, a Requestor may use a large ooo_count to know that many packets have arrived at the Responder, even if they have not yet been SACKed, and so infer that most packets are not sitting in a network queue. If this is the case, then packets missing just above the cumulative ACK for several RTTs have probably been lost.

7.5.5.5. SACK Header (SETH)

The BTH OPCODE = SETH, payload contains the SETH header.

Table 7-14 Reliability SACK Header (SETH)

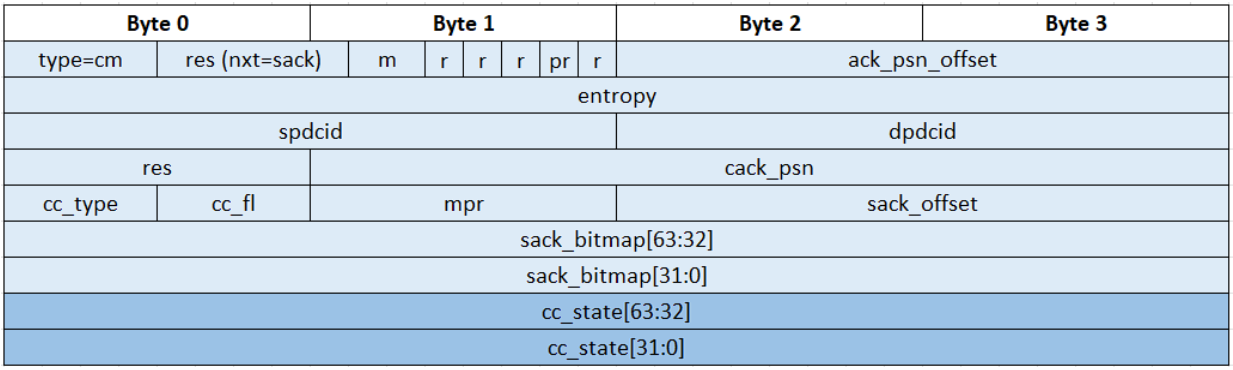

Table 7-15 Reliability SACK Fields

Field	Size (bits)	Description
res (type)	4	Reserved
res (nh)	5	Reserved
m	2	Set if the packet which triggered the SACK was ECN marked, this indicates the path associated with ENTROPY experienced congestion. 0b00 – none 0b01 – Skip once (ECN marked) “SKIP” 0b10 – Always skip (bad path) “ALWAYS_SKIP” 0b11 - reserved
r	1	Reserved
r	1	Reserved
r	1	Reserved
pr	1	Probe response (set when SACK is generated via a Probe Req)
r	1	Reserved
ack_psn_offset	16	Set to the RoCE PSN from the packet that triggered the SACK. This is encoded as a signed offset from the cumulative ACK psn (ack_psn = cack_psn + ack_psn_offset) (16b 2’compliment signed integer) For Reliability Probe responses this field is populated with the value of the Probe Identifier (PETH.probe_id)
entropy	32	Reflected request entropy value
spdcid	16	Source packet delivery context (set to requestor QP[15:0])
dpdcid	16	Destination packet delivery context (set to responder QP [15:0])
res	8	Reserved
cack_psn	24	Cumulative ack PSN
res	6	Reserved

cc_type	4	Defines the meaning of CC_STATE field, refer to congestion control section
cc_fl	4	CC flags (not used)
max_psn_range (mpr)	8	Allows Responder to indicate the size of the bitmap resource available to track arriving packets. 0x0 – Ignore value 0x1-0xFF – maximum bit map (128*n).
sack_ offset	16	Offset of SACK bitmap relative to cack_psn. (sack_ack = cack_psn + sack_offset) (16b 2'complement signed integer)
sack_bitmap	64	Bitmap indicating which packets have been received based on PSN, a bit is set to one indicates the packet with the corresponding PSN was received; a bit set to zero indicates the packet with the corresponding PSN has not been received. This is anchored based on the sack_ack = cack_psn + sack_offset.

7.5.5.6. NACK Packet Header (NETH)

The BTH OPCODE = NETH, payload contains the NETH header. Note original packet length from the trim packet MUST be copied into the NETH UDP header length field.

Table 7-16 Reliability NACK Format (NETH)

Byte 0		Byte 1	Byte 2	Byte 3
type=cm	res (nxt=nack)	reserved	nack_reason	vendor_info
entropy				
spdcid			dpdcid	
res		nack_psn		
cc_type=ts	cc_fl	reserved	tx_timestamp	

Table 7-17 Reliability NACK Fields

Field	Size (bits)	Description
res (type)	4	Reserved
res (nxt=nack)	5	Reserved
reserved	7	Reserved
nack_reason	8	Value indicating what event triggered the NACK packet. 0x01 – TRIMMED 0x02 – TRIMMED_LASTHOP 0x06 – NO_BITMAP 0x07 – NO_PKT_BUFFER 0x0A – NO_RESOURCE 0x0B – PSN_OOR_WINDOW

		0x19 – UNEXP_EVENT All other values are reserved
vendor_info	8	Vendor specific field for debug and/or statistics (optional)
entropy	32	Request entropy
spdcid	16	Source packet delivery context (set to requestor QP [15:0])
dpdcid	16	Destination packet delivery context (set to responder QP [15:0])
reserved	8	reserved
nack_psn	24	Set to the RoCE PSN from the packet that triggered the NACK
cc_type	4	Defines the meaning of CC_STATE field, refer to congestion control section. cc_type=0b0010 (Timestamp)
cc_fl	4	CC flags, not used
res	8	reserved
tx_timestamp	16	tx_timestamp copied from data packet if present, 0 otherwise

7.5.5.7. Probe request (PETH)

The format of Probe Request Headers (PETH) is:

Table 7-18 Reliability Probe Request Format (PETH)

Byte 0		Byte 1	Byte 2		Byte 3
res(type=cm)	res (nxt=probe)	reserved	reserved		vendor_info
probe_id			reserved		
spdcid			dpdcid		
tx_timestamp			tsr	reserved	ftype

Table 7-19 Reliability Probe Request Fields

Field	Size (bits)	Description
res (type)	4	Reserved
res (nh)	5	Reserved
reserved	7	Reserved
reserved	8	Reserved
vendor_info	8	Vendor specific field for debug and/or statistics
probe_id	16	Probe identifier
reserved	16	Reserved
spdcid	16	Source QPN [15:0]
dpdcid	16	Destination QPN [15:0]
tx_timestamp	16	Transmit timestamp
tsr	1	Timestamp resolution:

		0x0 = 128ns 0x1 = Implementation defined; should not be interpreted or used by responder
reserved3	11	reserved
ftype	4	0x1 if the tx_timestamp is carried

7.5.5.8. Service Time

Responders MAY perform service time compensation by adjusting requestor timestamps or reporting local Service Time. This capability is negotiated per-connection and is directional.

Capability negotiation:

- Devices that support service time compensation MUST support it in Requestor and Responder roles
- On a per-connection basis, both peers MUST set a SW modified QP attribute indicating whether the peer supports service time compensation
    - This capability is directional; neither, one or both ends MAY support it

Service Time support:

- Requestor role:
    - Reflected timestamps: If compensation is required, packet timestamp resolution MUST use the specification defined value (Sec. 6.2.2.3)
    - Stored timestamps: Requestor MUST parse and interpret the tx_timestamp field in every SACK and NACK packet for remote service time
- Responder role:
    - Packet contains TSETH: Responder MUST compensate for local service time where timestamp resolution is the specification defined value; compensated timestamp resolution MUST match requestors timestamp resolution
    - Packet does not contain TSETH: Responder MUST report local service time in the tx_timestamp field of the corresponding SACK or NACK using the specification defined resolution value (Sec. 6.2.2.3)

If the peer does not support service time, an endpoint MAY infer remote service time or mask local service time in an implementation-defined method.

7.5.6. Endpoint Packet Formatting

Endpoint packet stacks and population of Ethernet, IP and UDP headers is identical to that of Reliability Packet (Sec. 7.5.5) with the following exceptions:

- Endpoint Requests contain a requestor-private identifier in BTH.PSN[15:0] which is reflected in the corresponding Endpoint Response BTH.PSN field

7.5.6.1. Endpoint Request (ERTH)

The format of Endpoint Request Headers (ERTH) is:

Table 7-20 Endpoint Request Format (ERTH)

Byte 0		Byte 1		Byte 2	Byte 3
res(type=cm)	res (nxt=ep_req)	reserved	op	reserved	vendor_info
port_status_mask					
reserved					
tx_timestamp			tsr	reserved	ftype

Table 7-21 Endpoint Request Fields

Field	Size (bits)	Description
res (type)	4	Reserved
res (nh)	5	Reserved
reserved	5	Reserved
op	2	Endpoint Operation. Values: 0x00 = Port Status Update 0x01 = EV Probe
reserved	8	Reserved
vendor_info	8	Vendor specific field for debug and/or statistics
port_status_mask	32	Only populated for Port Status Update operation. Bitmap of requestor local port status; bit positions correspond directly to associated port numbers. A bit of value 1 indicates the port is reachable.
reserved	32	Reserved
tx_timestamp	16	Transmit timestamp
tsr	1	Timestamp resolution: 0x0 = 128ns 0x1 = Implementation defined; should not be interpreted or used by responder
reserved3	11	reserved
ftype	4	0x1 if the tx_timestamp is carried

7.5.6.2. Endpoint Response (EETH)

The format of Endpoint Response Header (EETH) is:

Table 7-22 Endpoint Response Format (EETH)

Byte 0		Byte 1			Byte 2	Byte 3
res(type=cm)	res (nxt=ep_rsp)	r	op	r	reserved	
reserved						
reserved						
reserved						
reserved						
reserved						
reserved						
tx_timestamp					reserved	
reserved						

Table 7-23 Endpoint Response Fields

Field	Size (bits)	Description
res (type)	4	Reserved
res (nh)	5	Reserved
reserved	3	Reserved
op	2	Endpoint Operation. Values: 0x00 = Port Status Update 0x01 = EV Probe
reserved	32	Reserved
reserved	32	Reserved
reserved	32	Reserved
reserved	32	Reserved
reserved	32	Reserved
reserved	32	Reserved
tx_timestamp	16	- If request carried timestamp and responder supports service time: Service time adjusted requestor timestamp - If request carried timestamp and responder does not support service time: Reflected requestor timestamp - If request did not carry timestamp and responder supports service time: Responder service time in 128ns units - Zero otherwise.
reserved	16	Reserved
reserved	32	Reserved

8. Congestion Control

8.1. Overview

MRC relies on congestion control algorithms to manage outstanding data and maintain fairness across paths and QPs. Congestion control operates per QP and adapts dynamically based on feedback from the responder and the network.

8.2. NSCC Congestion Control

MRC adopts the UET Network Signal Congestion Control (NSCC) algorithm for congestion control [UET101, Sec. 3.6.13.3-7]. This algorithm is a sender-side, SACK clocked, window-based algorithm. The goal of this algorithm is to dynamically adjust the connection congestion window to limit the amount of outstanding data in-flight so that the delay to the responder is within a user-provided Target Queueing Delay (target_Qdelay) bound.
NSCC relies on two mandatory network signals:

- Request RTT: A lagging indicator that estimates queueing delay by measuring the latency between a request packet and its subsequent acknowledgment
- ECN: A leading indicator that signals congestion in the fabric between the requestor and responder

NSCC adjusts the congestion window using three main mechanisms depending on the ECN and RTT values associated with received acknowledgements as outlined in Table 8-1.

Table 8-1 NSCC Congestion Window Adjustment

ECN	Request RTT	Inferred Network State	Adjustment Mechanism
NOT SET	RTT < target_Qdelay	Network is uncongested	Proportional increase
NOT SET	RTT >= target_Qdelay	Network was previously congested, but congestion has reduced	Fair increase
SET	RTT >= target_Qdelay	Network is congested	Multiplicative decrease
SET	RTT < target_Qdelay	Network is becoming congested, but this packet experienced no congestion	None

NSCC is also capable of leveraging the following optional signals to improve performance depending on network configuration and workload:

- Trimmed Packet notifications
- Achieved goodput
- Receiver Window Penalty

Implementations SHOULD implement the NSCC algorithm as defined in the UltraEthernet Specification [UET101, Sec. 3.6.13.3-7].

8.3. QP Congestion Controller

MRC specifies a QP Congestion Controller (QPCC), functionally equivalent to the UEC Congestion Control Context (CCC) [UET101, Sec. 3.6.13.3-7]. One or more QPCCs is instantiated per destination and used to manage congestion state for a subset of active QPs to the destination. The QP-to-QPCC mapping is implementation-defined. When all the QPs mapped to a QPCC are destroyed the QPCC is destroyed.

The sending NIC runs a scheduler to distribute access to the link between active QPs. Notionally, the scheduler maintains a list of ready QPs, a list of active QPs, a list of pending QPs, and a list of idle QPs.

- A QP is ready if there is data (new or retransmissions) ready to be sent, and the QPCC will permit sending at this time.
- A QP is active if there is data not yet sent, or data queued for retransmission, but the QPCC will not currently permit sending.
- A QP is pending if it is not active, but there is data that has been sent but not yet acknowledged.
- A QP is idle if it is not ready, active or pending.

The state machine below indicates the transitions between the four states. The scheduler only schedules traffic for QPs in ready state.

The state machine below indicates the transitions between the four states.

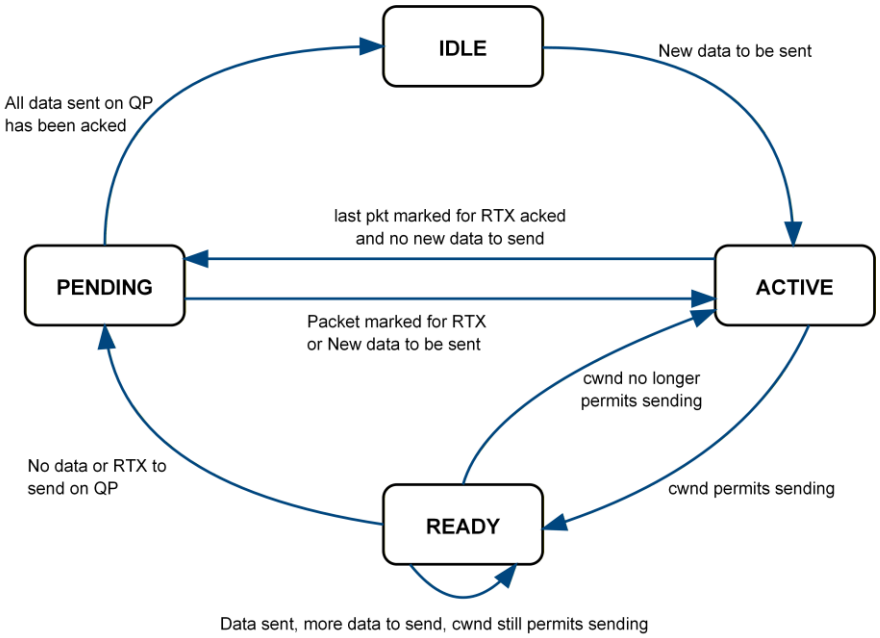

Figure 4 Schedule State-machine

To summarize the state transitions:

- A QP transitions from IDLE state to ACTIVE when new data arrives to be sent.
- A QP transitions from ACTIVE to READY state if there is space in the sender's cwnd.
- A QP transitions from READY back to ACTIVE if there is no longer space in the sender's cwnd. This typically occurs when data is sent, filling the cwnd. It can also occur when sending does not occur, if a SACK or NACK arrives causing the cwnd to be reduced.
- When in READY state, if data is sent but the cwnd still allows sending, the QP stays in READY state.
- A QP transitions from READY to PENDING state when there is no more new data to send and no data to retransmit.
- A QP transitions from PENDING to IDLE state when all data on the QP has been ACKed. QP state MUST NOT be removed except when in IDLE state, unless the QP has transitioned to ERROR state.
- A QP transitions from PENDING to ACTIVE state if a packet previously sent is marked for retransmission. This can be caused by receiving a NACK, inferring loss from a SACK, or due to a retransmission timeout.
- A QP transitions from ACTIVE to PENDING state if there are no new data packets to send, and the last data packet marked for retransmission is acknowledged. There will usually still be packets unacknowledged, but nothing currently needs sending. If everything is also acknowledged, the QP will immediately transition on from PENDING to IDLE.

The scheduler will rotate between QPs in ready state, sending packets from them in turn. How the scheduler chooses which QP to send next is implementation specific.

Packet arrivals and timer expirations drive state transitions in the sender's QP Congestion Controller (QPCC), invoking the appropriate congestion-control routines. Table 8-2 summarizes NSCC event-to-action mappings [UET101, Sec. 3.6.13.3-7].

Table 8-2 NSCC Event Table

Event	NSCC Action
A SACK packet arrived for this QP	OnACK()
A NACK packet arrived for this QP	OnNACK()
A QP retransmit timer expired, or a packet is inferred lost for this QPCC	OnInferredLoss()
New data was queued by the application to be sent by the corresponding QP	OnNewData()
The scheduler sent a packet from the corresponding QP	OnSend()

Each of these actions can cause the state machine for the QPCC to move between the ready/active/pending/idle states. These actions do not directly cause data packets to be sent; they only impact the state of the QP. When the QP is in, or transitions to the READY state, it

becomes eligible to transmit packets. The scheduler then selects from the QPs in the READY state to send packets.

In addition, when the scheduler is ready to send, it calls GetSendParams() [UET101, Sec 3.6.12.3] on the QPCC, which returns any fields from the packet that need to be filled in that may change between the QP becoming ready and the scheduler sending it. One of these fields is the EV that determines the packet's path. This can only be allocated when the packet is finally going to be sent because incoming ACKs and NACKs arriving between the QP being ready and the packet being sent may change the choice of path/EV and port.

8.3.1. CC Window State at a Requestor

Each Requestor MUST maintain a 32-bit count of all bytes sent on a QP that have not yet been acknowledged. This is known as inflight bytes. The requestor can only send when cwnd > inflight. This counter is incremented by nominal_pktsize on transmit and decremented by the amount of newly acked bytes (as calculated below) on SACK/NACK reception. nominal_pktsize is the UDP length plus nominal_hdrsize. nominal_hdrsize is a fixed constant which remains unchanged during the lifetime of the connection. Its value is 40 bytes. nominal_pktsize is used to ensure that the same packet size value is used by the Requestor to increase inflight and by the Responder to maintain rx_rcvd_bytes, even if the actual packet size changes in transit.

Each Requestor MUST maintain a 24-bit count of all acknowledged bytes on a QP as prev_rcvd_bytes. This value is updated when a SACK/NACK packet arrives and wraps at 0xFFFFFF to 0x0. At connection setup this is initialized to zero. As pkt.rcvd_bytes communicates the Responder's rx_rcvd_bytes value in chunks of 256 bytes, prev_rcvd_bytes is also maintained in chunks of 256 bytes.

When a SACK or NACK arrives at the Requestor, before the actions above are called, some data needs to be prepared. This preparation is detailed below.

Table 8-3 CWND State at Requestor

Field	Size (bits)	Description
inflight	32	Sum of the nominal_pktsizes of all packets sent and still thought to be neither received or lost
prev_rcvd_bytes	24	Previous largest value of rcvd_bytes field seen from SACK packets (modulo 2^24)

8.4. Requestor SACK preprocessing for Congestion Control

A selective ACK indicates the cumulative total of bytes that arrived using the rcvd_bytes field in the CC_STATE information, which is derived from the Responder's rx_rcvd_bytes count. When a SACK or NACK arrives, the Requestor calculates the number of bytes which have newly

arrived at the Responder. rx_rcvd_bytes monotonically increases at the Responder, so this calculation ignores any SACKs/NACKs where this field decreases as they must be out of date due to reordering or loss of SACK/NACK. The sender performs the following calculation:

```
if pkt.rcvd_bytes > qpcc.prev_rcvd_bytes:
      newly_rcvd_bytes = (pkt.rcvd_bytes - qpcc.prev_rcvd_bytes) << 8
      qpcc.prev_rcvd_bytes = pkt.rcvd_bytes
```

The left shift is required as pkt.rcvd_bytes is in units of 256 bytes, whereas newly_rcvd_bytes needs to be in bytes. As the rcvd_bytes field can wrap, modular arithmetic must be used for the comparisons and subtractions.

A SACK message MAY contain a tx_timestamp field if the frame had a TSETH header present (BTH.tsh). If the sender does not insert a transmit timestamp in data packets, then it will instead have stored the transmit time of the packet in a local database; such a sender MUST ignore the tx_timestamp in SACK messages. The sender computes a round-trip time estimate by subtracting the arrival time of the SACK from either the ts_timestamp contained within the SACK or retrieves the time the packet was sent from its local database.

rtt_sample = sack arrival time – (pkt.tx_timestamp or local packet sent time)

If the tx_timestamp field is being used, then rtt_valid = True.

If the tx_timestamp field is not being used, a sender SHOULD store whether it has retransmitted a packet zero times, once, or more than once. If the packet has not been retransmitted, then rtt_valid = True. If the packet has been retransmitted once and the SACK packet has the RTX flag set, then rtt_valid = True. Otherwise rtt_valid = False. This check prevents confusing SACKs for the original data packet and retransmissions.

The entropy, rcv_cwnd_pen, M and rtx fields are extracted from the SACK.

The SACK packet may acknowledge one or more packets that had previously been marked for retransmission. These packets no longer need retransmitting, so:

```
      foreach newly acked packet:
            unmark_packet_for_retransmission(acked_psn)
```

Finally, the OnACK() action is performed.

8.5. Requestor NACK preprocessing for Congestion Control

A NACK indicates a packet did not arrive at the receiver. Typically, it was sent because the packet was trimmed in the network, but it can also happen because the receiver dropped the packet. When preprocessing a NACK, the following actions are performed:

The sender looks up the packet, finds the original nominal_pktsize. If the packet is not found, typically because it was ACKed in a previous SACK, then nominal_pktsize is zero.

rtt_sample is calculated in the same way it is for SACK messages:

rtt_sample = nack arrival time – (pkt.tx_timestamp or local packet sent time)

rtt_valid is also set in the same way as for SACK messages.

The entropy value, NACK reason_code is extracted from the NACK packet. The OnNACK() action is performed.

8.6. Congestion Control Behavior on New Data Arrival

When new data is requested to be sent on the QP, the number of packets required to send the new data is calculated and delta_backlog is calculated. When performing this calculation, delta_backlog should be the sum of the nominal_pktsize of all the data packets required to send the new data. The OnNewData() action is then performed.

8.7. Congestion Control Behavior when Data is Sent

When the scheduler sends a packet from the QP, it notifies the QPCC via the OnSend() action and must immediately increase inflight by the packet's nominal_pktsize. The QPCC then determines any state changes required. Note the packet does not actually need to be sent on the wire before OnSend() is invoked, but the sender MUST have irrevocably decided this is the packet to be sent as soon as possible, modulo any egress pipeline latency. Congestion control will consider the packet to have been sent.

9. Entropy Generation and Encoding

A key feature of MRC is supporting multiple paths from source to target and distributing or load balancing (i.e., spraying) packets across these paths. The two primary functions are path selection or EV generation and encoding the path into the into the packet.

There are three supported forwarding models which define how the path is encoded into the packet

- ECMP Hash-based Forwarding.
- Source routing using Structured EV
- Segment Routing over IPv6 (SRv6)

9.1. Structured EV

Structured EV is an MRC specific EV format which requires endpoint and switch support for parsing EVs.

Structured Entropy Values (EVs) provide a deterministic mechanism for multipath routing by encoding path selection instructions directly into L3 packet headers. In IPv6 networks, a full 32-bit Structured EV can be represented by concatenating the lower 16 bits of the IPv6 Flow Label with the 16-bit UDP Source Port, forming a composite field that enables fine-grained path control. IPv4 networks lack the Flow Label field, so entropy encoding is limited to the UDP Source Port, reducing the available entropy space.

Table 9-1 Structured EV Logical Format

Byte 0	Byte 1	Byte 2	Byte 3
UDP.src_port[15:0]		IPv6.flow_lbl[15:0]	

During configuration, a Structured Entropy Value (EV) is partitioned into multiple hop-specific subfields. Each subfield encodes routing metadata for a distinct network stage, enabling deterministic forwarding decisions at successive switches. The number of fields, widths and valid value ranges are configuration specified and known to all endpoints and switches in the fabric. Every switch in the fabric is configured to examine a particular hop field in the structured EV and uses the value in the field to pick an appropriate egress port to forward the packet.

An example Structured EV with 3 hops of widths 10b, 8b and 4b is shown below. In this example, the first switch that receives the packet uses the value of hop0 to make its forwarding decision, the second switch uses hop1 and the third switch uses hop2.

Table 9-2 Example Structured EV with 3 hops of 10b, 8b and 4b

Byte 0	Byte 1	Byte 2	Byte 3
hop0	hop1	hop2	

Responders reflect a request packet's Structured EV, unmodified, in any corresponding Reliability ACK/NACK generated for the packet.

9.2. SRv6

The following SRv6 configurations are supported:

- SRv6 uSID
- SRv6 uSID + SRH

A maximum of 1 SRH is supported. If present, the SRH extends the uSID stack to encode additional uSID segments beyond those in the base SRv6 address. The SRH MAY alternatively be used to duplicate the base SRv6 address. This capability allows the original path to be identified from packet captures without altering forwarding behavior.

9.3. EV Selection

Packet spraying is accomplished by varying the entropy, EV, in each packet. The EV selects a path within the network, either by hashing into an ECMP set in each switch or being mapped by the switch to a specific static path. The EV is communicated using the packet fields or dedicated fields in the Reliability control messages where appropriate.

9.3.1. Path-aware Multipath EV Selection (Spraying)

The sender MUST rotate through a set of EVs configured on a per-QP basis, with the aim of spreading traffic over many paths. EVs are logically grouped into an EV Profile; every QP is associated with one EV Profile. The number of EVs within an EV profile (EV Universe) are fixed for the lifetime of the EV Profile. Each EV in the EV Universe is either eligible for use in transmission (Active EV) or ineligible (Inactive EV). The relationship between The EV Universe, Active EVs and Inactive EVs is: EV Universe = Active EVs ∪ Inactive EVs

EV selection SHOULD be pseudo-random to avoid synchronization of paths between different QPs using the same set of Active EVs when source-routing is used.

SACKs and NACKs echo the EV from the data packet back to the sender together with the M field (SACK only) which indicates whether that path experienced congestion. This allows the sender to detect transient congestion on specific paths and perform active load balancing. It also allows the sender to detect paths that have failed and have not yet been routed around or are otherwise suffering from link speed or bit error problems.

Switches in the network MUST be configured to set ECN. The recommended values are to start setting ECN when the queue size exceeds min_thresh and to set ECN on all packets when the queue size exceeds max_thresh as specified in the UltraEthernet Specification [UESPEC], Section 3.6.17.

Senders maintain per-EV state such that each EV is in one of the following states:

Table 9-3 EV States

EV State	Set Membership	Comments
GOOD	Active EVs	EV in this state SHOULD be available for selection
DENIED	Inactive EVs	Used by control-plane to disable EVs
SKIP	Inactive EVs	EV has been recently marked in a SACK or ACK message and SHOULD be skipped in EV selection. The implementation SHOULD transition SKIP EVs back to GOOD in an implementation-defined time
ASSUMED_BAD	Inactive EVs	Implementation has detected destination is unreachable using this EV (e.g. packet timeout). Recovery is implementation-defined.

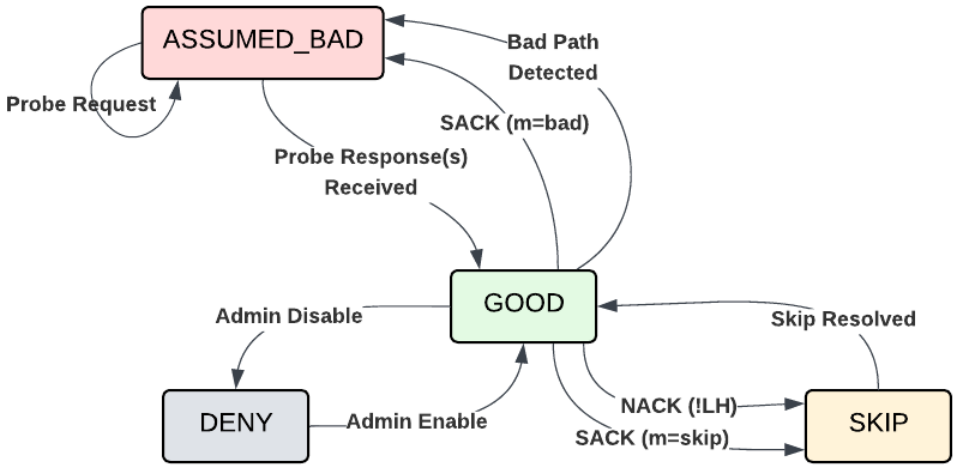

Figure 5 EV State Diagram

Each EV SHOULD default to GOOD state on QP startup, unless configured otherwise. For example, the controller may know one or more paths are bad and configure them into the DENIED state.

When a sender receives a SACK packet with the M flag set to SKIP_ONCE, it indicates a data packet arrived at the destination with ECN set and the path experienced congestion. In addition, if the sender receives a NACK packet with the TRIMMED (not TRIMMED_LASTHOP) as the nack_reason code, it also indicates congestion. In these cases, the EV transitions to the SKIP state.

The responder MAY also indicate that a path is known bad by setting the M flag to ALWAYS_SKIP. In this case, the EV is transitioned to ASSUMED_BAD state.

The sender MUST select EVs from the Active Set. The order in which the sender rotates through the set is implementation defined. An EV in the skipped state SHOULD be skipped using implementation defined mechanisms to improve load balancing.

A sender MUST provide a mechanism to detect bath paths. The mechanism MAY be a packet acknowledgment timeout, EV probing, etc. Once a bad path is detected, the EV is placed into the ASSUMED_BAD state and a reliability probe SHOULD be sent periodically to determine when the path is good.

Recovering an EV value from ASSUMED_BAD state is implementation-defined and can be achieved by sending reliability or EV probes, but the details of when to send a probe are implementation specific. If the probe elicits a SACK with M set to NONE or SKIP_ONCE, the entropy value should be moved to GOOD or SKIP states respectively.

9.3.2. Multi-plane EV Selection

When a NIC is configured so that multiple ports are connected to separate planes, the aim is for a single QP to be sprayed across all the planes. In this mode, the EV space is split across those planes, so that each EV maps to a unique persistent plane.

When the NIC scheduler is ready to send a packet from a QP that is in READY state, it invokes GetSendParams(), passing the QPCC a bitmap, free_ports, indicating which ports currently have space for transmission. As the QPCC scans through the set of EVs looking for one that is in GOOD state, it will not only skip EVs that are in SKIP or ASSUMED_BAD state, but it will also skip any EV that is not marked as free in the free_ports bitmap.

It is recommended that the order of scanning through the EVs is randomized periodically to avoid any aliasing between scheduler ports becoming available and EVs being used.

GetSendParams() will return both the EV to be set in the packet and the port to send it on.

In multi-plane mode, control packets (e.g., SACK/NACK) may egress on a different plane than the request packet's ingress plane. Therefore, the requestor MUST be able to identify the plane used for each transmitted request when processing the corresponding control response. This capability is implementation-defined. Common approaches include:

- Assigning EVs that are unique per plane
- Recording the transmit plane in the outstanding packet's metadata

9.3.2.1. Denylisting EVs and Ports

EVs MAY be configured to not be used if the operator knows in advance that an EV will map to a bad path. When this happens, the EV is set to the DENIED state and will be skipped when considering which EV to use for a packet. Similarly, a NIC port may also be configured as DENIED. Packets are then sprayed across the remaining non-DENIED ports.

9.3.3. Networking Model using ECMP

ECMP (equal cost multiple path) is a traditional way for networks to support multiple paths to a destination. Fields from the packet, including entropy value, are used in a hash function to select one of the paths. This is typically deployed as a CLOS network where ECMP is used to spray traffic to a root switch and traditional routing is used to forward from the root to the destination. The sender will rotate EVs (entropy values) to spray packets across the paths. The sender monitors feedback from the receiver in the form of SACK, NACKs, ECN, and actively adjusts its load balancing to reduce load on any paths that it discovers are congested and completely avoids any failed paths. Without detailed knowledge of the hash function and switch configuration, the sender does not know precisely which path an EV takes, but in the absence of failures, it expects that path to be stable on much longer timescales than message transfers.

9.3.4. Network Model using Structured EVs

Structured EV are like ECMP typologies except instead of relying on ECMP hashing function in the switches, the path to the root switch is selected by the NIC and encoded into the packet entropy value. Switches are pre-configured to map parts of the entropy field to a specific static route or port. Different switch necessarily use different fields within the entropy value until the root is reached. From the root, traditional IP routing is used to forward the packet. Consider the following 3 tier network.

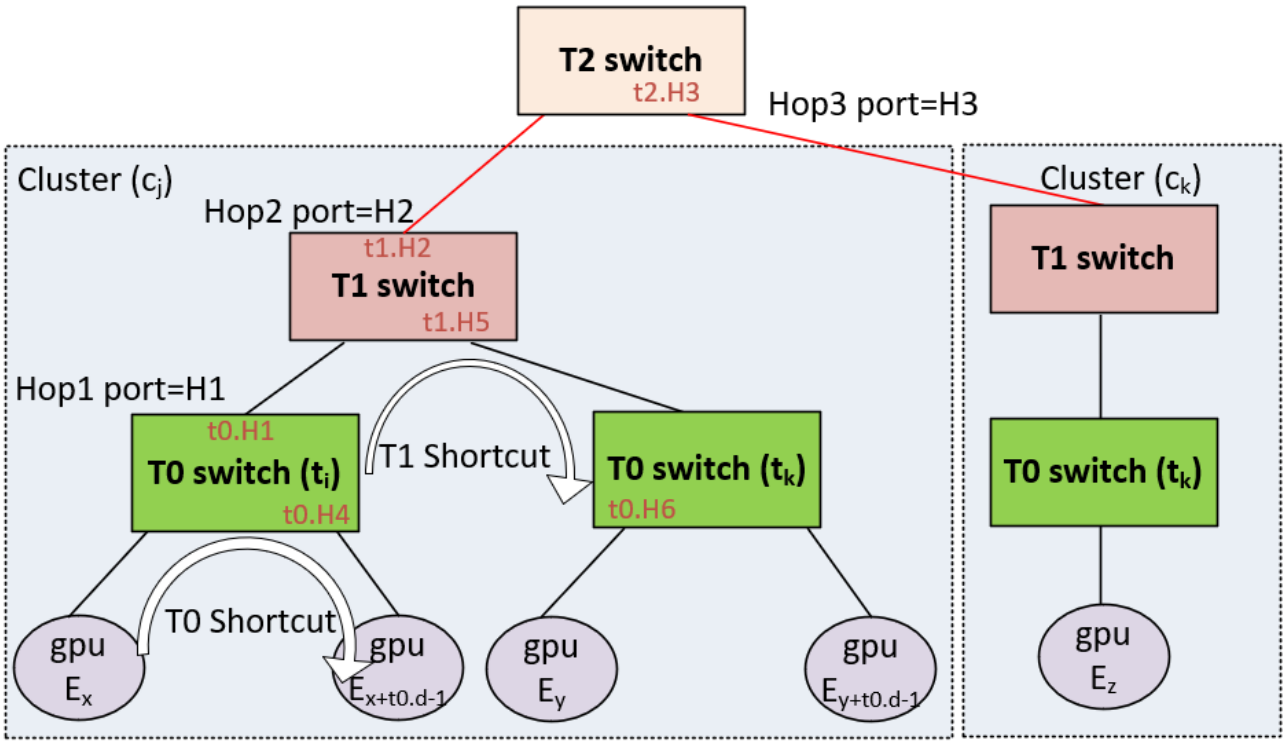

Figure 6 Structured EV Network Example

Three hops (H1, H2 and H3) are used to identify output ports of the tier0, tier1 and tier2 switches respectively. The hops are encoded into the entropy field of the packet carried in the IPv6 flow label and UDP source port. For example, the path from GPU Ex to GPU Ez would use an entropy of t1.H2||t0.H1 to reach the T2 root switch. Switch T2 may use Hop3 for traditional ECMP routing if multiple paths from the root to destination are desired. For completeness, two short cut paths are shown. These paths use optimized connections which don't go to the T2 root (Ex to Ey and Ex to Ex+1 in the example). Switch Forwarding in the T0 and T1 switch MAY be based on ACLs (Access Control List) or other static forwarding tables.

The source-routed forwarding model relies on changing the H1, H2 and H3 fields to distribute traffic across the paths. The process of selecting the paths is implementation define but MAY include a hash-based distribution function.

9.3.5. Network Model using SRv6

MRC supports source routing using SRv6 [RFC8986]. SRv6 explicitly defines a path from source to destination using micro-segments (uSID) to identify each hop in the path. SRv6 implies an IP-IP encapsulation where the outer header is used to carry the segment routing information. MRC uses uSID [SRV6USID] containers to encode segment values.

A single optional SRH MAY be used to encode additional segments or to carry a copy of the uSID container for debug and telemetry purposes. If a SRH is used it MUST be encoded using Compressed Segment List Encoding [RC9800].

The LID is a common value used across a set uSIDs in a container. The individual uSID values are used by the switches to determine which the next hop for the packet. uSIDs are processed by each switch via a pop-and-left-shift operation. The forwarding behavior is based on if uN (Micro-Node SID) or uA (Micro-Adjacency SID) mode is used. In uN mode, after shifting the revealed uSID points to the next hop router. The remaining uSIDs in the SRv6 address are shifted left and will be processed by the next hop switch. When an SRH header exists, and there are no more uSIDs in the SRv6 address, the segment in the SRH header is promoted to the SRv6 address thereby supplying the next set of uSIDs. The switch will execute this promotion and proceed with processing the packet.

For example, consider the following 3 tier network which uses the uN behavior to forward to the adjacent router.

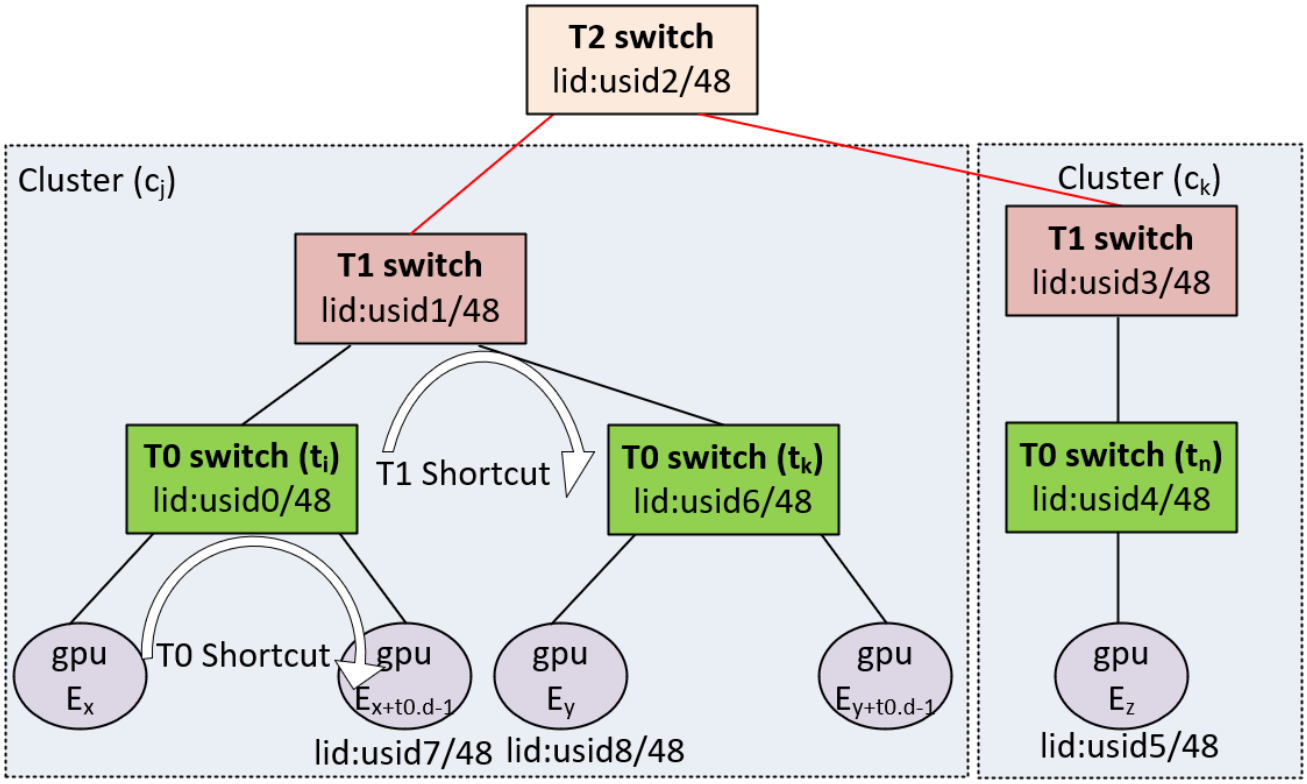

Figure 7 SRv6 Network Example

The path between GPUs is fully specified using the SRv6 segment routing uSID stack. All switches are configured as SRv6 transit switches. The NIC is responsible for the SRv6 encapsulation and decapsulation. The LID is common for destinations and up to 6 uSIDs are

used to describe the path between GPUs. The following 3 cases depicts the uSID stack between various GPUs.

1. GPU Ex to GPU Ez: lid:usid0:usid1:usid2:usid3:usid4:usid5
2. GPU Ex to GPU E x+t0.d-1: lid:usid0:usid7:00:00:00:00 – T0 Shortcut
3. GPU Ex to GPU Ey: lid:usid0:usid1:usid6:usid8:00:00 – T1 Shortcut

9.3.5.1. Received SRv6 Packet Header Stack

MRC NIC implementations MUST handle both SRv6 header stack variants:

1. Preserved outer SRv6 encapsulation — packets retain the outer IPv6 header and SRH as per End/End.X behavior.
2. Decapsulated SRv6 encapsulation — packets are processed according to End.DX/End.DT behavior, where the outer SRv6 header is removed.

9.4. EV Generation

EV generation is the process of selecting among the paths to a destination. The precise mechanism of the selection is vendor defined. There are two schemes for generating EV: Explicit and Generated.

9.4.1. Default Mode

If an EV profile is configured for AUTO mode or an EV profile isn't assigned to a QP, the set of entropy values used is vendor defined. As an example, a vendor might choose to implement ECMP style hashing for the default mode. Refer to vendor-specific documentation as required.

9.4.2. Explicit EVs

Explicit EVs are those that are explicitly programmed by the controller application. The controller knows all the available paths between the source and destination so it can compute a set of 32b EVs that can be used in a profile. Once the EV profile is allocated and configured for EXPLICIT mode, it can program each of these EVs into the profile which would then be used by all QP associated with the profile.

Support for explicit EV profiles introduces additional on-chip state and may not be supported by a vendor's NIC. As part of the MRC control APIs, there is a device capability exposed that specifies whether explicit EVs are supported.

9.4.3. Generated EVs

Generated EVs are those which are generated on the NIC itself. The controller application configures the parameters for the generation, and the NIC generates a set of EVs for the profile. Depending on the number of hops required and number of bits at each hop, the controller

application can specify a set of generation parameters that allows the NIC to generate a correct set of EVs for the network topology.

The generation parameters are defined as a list of EV Format Field structures. Each structure is logically comprised of the following fields:

- Width – Number of bits that the field requires
- Min value – The minimum value that can be generated in the field's bits
- Max value – The maximum value that can be generated in the field's bits

The sum width of the fields in the list cannot exceed:

- 16b for ECMP
- 32b for Structured EVs
- 128b for SRv6 uSID
- 256b for SRv6 uSID+SRH

Full details on EV Format Fields are available in the SW API [MRCSW].

The configuration of the field generation parameters is per MRC device. The maximum number of EV Format Fields and the bounds of these fields is implementation-defined. A QP is configured to select an EV profile. An EV profile configured for generated mode will have a set of EVs generated by the NIC that are within the bounds of these field parameters.

Support for generated EV profiles introduces additional hardware logic and may not be supported by a vendor's NIC. As part of the MRC control APIs, there is a device capability exposed that specifies whether generated EVs are supported.

10. Software API

With the new Multipath RC (MRC) QP type behavior being introduced by this spec, software changes are necessary. Additional functionality must be exposed to the application to allow the application to learn about the underlying device's MRC capabilities, configuring the device, create and configure MRC type QPs, and driving operations over those MRC QPs.

10.1. MRC Library

MRC functionality is exposed to an application via a dynamically loaded (or static) library. The MRC library is mostly independent from libibverbs. The library will provide APIs for querying MRC capabilities on a device, creating/modifying/destroying MRC type QPs, posting work requests to these QPs, polling for completion events, and configuring advanced MRC features. Like libibverbs, each vendor will implement their own MRC provider driver that glues the MRC library application-level APIs to the device's kernel driver.

An MRC capable application is required to continue to use libibverbs for device discovery and IBV context management when opening or closing a device. When a device is opened via libibverbs, the returned ibv_context is used for the device binding and configuring MRC.

10.1.1. MRC APIs

MRC APIs are defined at the following location [MRCSW]:

Defined by the libmrc project are two distinct header files:

- mrc.h – Used by an application for MRC device discovery, context management, QP/CQ management, and sending work requests
- mrc_ctl.h – Used by a privileged controller application for MRC device management, CC profile management, EV profile management, EV state event processing, and EV probes

Note: Further learning of the details and specifics of the programming interface as mentioned in this specification must be directed to the libmrc project on GitHub[MRCSW].

10.1.2. MRC QP Connection Setup

10.1.2.1. QP1 Support

RDMA-CM support is not provided for MRC.

10.1.2.2. Connection Attributes

Per-QP attribute exchange MUST occur using an out-of-band connection setup scheme. The connection attributes exchanged between peers are:

Table 10-1 QP Connection Attributes

Attribute	Directionality	Description
MRC_QP_MAX_WIMM_DEST	Unidirectional	Maximum number of in-flight WriteIMM requests supported by the responder
MRC_QP_MAX_MPR_DEST	Unidirectional	Max MPR supported by the responder
MRC_QP_DYNAMIC_MPR	Bidirectional	Support for Dynamic MPR. If either peer does not support this capability, it is disabled in both directions
MRC_DEVICE_CAP_TRIM_NACK	Unidirectional	Responder capability to generate TRIM NACKs
MRC_DEVICE_CAP_SVC_TIME	Unidirectional	Responder capability to report local service time

10.2. Writing Applications for MRC

MRC software is broken up into two different entities. The first entity is the MRC application itself (i.e., NCCL or similar) which is a typical RDMA application using the MRC application APIs defined in mrc.h. These APIs parallel libibverbs and allows the application to create/modify/destroy MRC QPs and CQs. Additional MRC specific configuration is included in the QP setup process using mrc_modify_qp(). The second entity is the MRC controller application using mrc_ctl.h which is responsible for configuring an MRC device with special configuration to be used by MRC application QPs. An MRC controller application is a privileged entity and must be running with the CAP_NET_ADMIN capability.

10.2.1. MRC Applications

An MRC application is a typical RDMA application that is interfacing with the set of MRC APIs instead of Verbs. The MRC application APIs are modeled directly after the Verbs create/modify/destroy design wherever possible. There may be multiple processes running simultaneously, each with their own instance of libmrc and allocated MRC QPs.
At a high level, the MRC application can:

- Determine MRC device capabilities:
    - Each vendor exposes different limits on various capabilities. The application must learn what the MRC device is capable of which results in proper MRC QP configuration.
- Carry out MRC resource management (i.e., like libibverbs RDMA resources):
    - MRC CQs and completion channels – The application is required to create MRC CQs to be assigned to QPs. Additionally, a completion channel may be created and CQs associated to it. This allows the application to poll/select for completions using a file descriptor.
    - MRC QP resources – The application allocates MRC QPs using the familiar create, modify, and destroy APIs. The mrc_modify_qp() call contains a large set of MRC specific attributes used to tune MRC features on a QP.

    - Post Send/Receive work requests – The application can post work requests to an MRC QP's SQ or RQ. Note that a receive work requests are only used for completions of WRITE w/ Immediate operations.
    - MRC asynchronous events – Various asynchronous events can be posted by the provider driver when errors occur.
- Restrictions:
    - No support for READ, SEND, or ATOMIC operations. If an invalid opcode is received by the Responder on an MRC QP, the QP will transition to the ERROR state.
    - The maximum number of outstanding WRITE w/ Immediate operations must be limited by the value advertised by the Responder.
    - No support for RNR.

10.2.2. MRC Controller

There is expected to be a single MRC controller application running per node (i.e., Linux server) in a cluster. The controller implementation may be designed as a long running daemon process or can be ephemeral and implemented as a command line application tool. Regardless of the design, the resource state configured by the controller is persistent in the device kernel driver and/or firmware.

At a high level, the MRC controller application is responsible for:

- Determine MRC device capabilities
    - Various capabilities are optional, and each vendor's NIC may provide a different set of features. The controller must learn the capabilities of an MRC device which will guide how EV and CC profiles may be configured.
- EV Management
    - EV Profile configuration – EV profiles are used to define a set of EVs that can be used by an application's MRC QP to spray packets to the destination over multiple paths in the network. An EV profile can be configured as one of a couple different available modes. Modes include explicitly defined EVs, NIC generated EVs, and SRv6.
    - EV value/state management – Within an EV profile each individual EV has a value and state attribute. For an explicit EV profile, a single EV's value can be modified so a packet using that value will travel a different path in the network. For all EV profile types, an EV's state can be modified to the DENY or GOOD states. A denied EV will not be used by the NIC and is instead skipped over.
    - EV Events – As EVs within an EV profile are used by the NIC, it could be determined that a network path is bad as packets using an EV value associated with that path aren't getting to the destination. In such a situation the NIC will transition the EV to the BAD state. When this occurs, the controller can receive an asynchronous EV event that includes the EV value and state. This allows the controller to react to the information immediately in determining if there is a problem in the network or not.

- EV Probes – The controller can send out-of-band EV probes. These probes are connectionless and not tied to a specific MRC QP. Probes allow the controller to test various network paths through the network to a specific destination. EV Probes MAY be sent on any traffic class/DSCP.
- CC Management
    - CC Profile configuration – CC profiles are used to define a congestion control algorithm that can be used by an application's MRC QP when sending packets to the destination. The default algorithm is NSCC and the controller APIs allow a vendor to define and expose their own custom algorithm(s).

Note: For EV and CC profile management, it is expected that the MRC user application and MRC controller application have a means to communicate between each other. The user application is required to learn what EV/CC profiles are available and to what destinations they can be used. With this information the application is then able to match these resources to the proper QPs during initialization. The design and implementation of the application/controller interaction is outside the scope of this specification and the libmrc APIs.

11. Implementation Guidance

The recommendations in this section are informative and should not be considered normative.

11.1. MPR and WriteIMM Dimensions

Implementations SHOULD support the following per-QP MRC property ranges:

Table 11-1 MRC Device Parameters

Property	Min	Max
max_psn_range (mpr)	1	32
max_wimm_inflight	0	32

Implementations SHOULD also support configurations where at least 16 QPs can be configured for:

- Min MPR = 8
- Min max_wimm_inflight = 32

11.2. Load Balancing

11.2.1. EVs per EV Profile

Feedback from the SACK and NACK messages needs to be timely, the size of the actively used EV set should not be significantly greater than the number of packets to be sent during a congestion window, though this is a soft bound. A goal should be to keep the Active EV set size roughly between one and two congestion windows. As an upper bound, generally performance is better the larger the Active EV set, but so long as active load balancing is performed, there are rapidly diminishing returns once the Active EV set size exceeds 100. Implementations may safely cap the size of the Active EV set to values in this general range.

11.2.2. EV Use

The order in which the sender rotates through the EV set is implementation-defined. An example algorithm is as follows:

The sender generally rotates through all the EVs in the active EV set before reusing an EV. When the sender chooses an EV for transmission, if that EV's state is GOOD, the EV will be used. However, if that EV's state is set to SKIP, the sender will reset that state to GOOD and try to find the next good EV. Please note that one send-packet can only reset one EV from SKIP to GOOD state. Ideally a sender would keep skipping EVs in this manner until it finds an EV is GOOD, but to limit implementation complexity, it MAY skip a fixed number of EVs and then send using an EV in SKIP state or use any reasonable algorithm to find an EV in GOOD state.

This algorithm provides active load balancing that significantly improves the load that the network can handle beyond what would be achieved with oblivious spraying.

11.3. Source-Routing Model

Expected deployment configurations for Structured EV and SRv6 source-routing modes:

- Source routing directs packets from the source to an upper-tier switch based on the Entropy Value (EV) encoded in the packet.
- From that point, standard IP forwarding delivers the packet to the destination

11.4. SRv6 uSID Container Formats

Expected deployment uSID Container formats are:

- F3216
- F1616

11.5. Device and Addressing Model

The following outlines a set of general expectations for a NIC that supports MRC. These provide a guideline to how an MRC capable NIC is integrated into the Linux environment and what an application developer can expect. Vendors are free to deviate from these guidelines in their own implementations if needed.

- Port = Plane & Plane = Port
- Every panel port is exposed as a PCIe physical function
- Every port has its own netdev interface exposed in the Linux networking stack
    - Management Address (mng_ipX): IP address assigned to a netdev interface used for management traffic (i.e., traditional L2 over that port)
    - Every interface has its own Management Address assigned (i.e., ip add addr <mng_ipX> dev <ifname>)
- The controller can learn which devices support MRC via a successful call to mrc_query_device()
    - Alternatively, a vendor can provide its own API to help the controller learn what's available
    - The vendor implementation must provide at least one netdev interface as MRC capable
        - A vendor's implementation could allow MRC on each PF, on a single PF, only on a VF, etc.
- If required by the deployment configuration, the admin/controller could assign an IP address (mrc_ipX) to an MRC capable interface that can be used by MRC QPs (useful for isolation of traffic from management using mng_ipX)
    - When an mrc_ipX IP address is assigned, the RoCE kernel driver will get a notification and create a new SGID on the interface corresponding to that address
    - When the application is launched, it learns from the admin/controller or job launcher which device should be opened and which SGID should be used for MRC communications (e.g., this is like traditional RDMA application command line args to assign the SGID, ib_write_bw -x sgid)

    - Note: The application can view all the SGIDs available on an MRC device using the standard IB Verbs APIs calling ibv_query_sgid()
    - Note: A deployment doesn't have to require an mrc_ipX address exist solely for MRC traffic, instead any IP address on an MRC interface could be used
- The vendor implementation defines which ports are usable as planes on an MRC device
    - A QP that is created on an MRC device can send and receive traffic over ANY available plane
    - The referenced EV profile defines the set of EVs available and the corresponding planes
- The vendor implementation determines how packets are directed through the NIC's Tx/Rx pipelines
    - When a packet is sent out on a plane, the srcMac is the port's MAC address and srcAddr is mrc_ipX
    - When a packet is received on a plane, the dstMac is the port's MAC address and dstAddr is mrc_ipX
    - Note: The Linux ARP Flux feature enables the networking stack to reply to ARP requests that come in on ANY interface for an IP address not assigned to the receiving interface, so every ToR will ARP for mrc_ipX and will get the connected NIC port's MAC address
    - On receive, a parser/action will be used to direct traffic to the corresponding interface and MRC context (i.e., match on dstMac, mrc_ipX, RoCE, MRC opcode, etc.), again this design is integrated into the vendor's implementation

The diagram below shows an example of a 4 port NIC where MRC is available on PF0 (as well as VF0 thru PF0). Each interface is assigned a mng_ipX address and PF0 (and/or VF0) is also assigned a second IP address mrc_ipX.

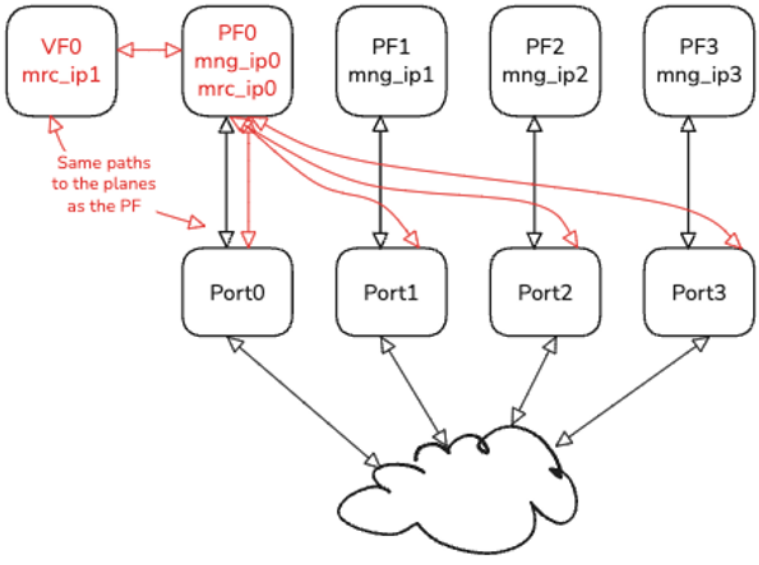

Figure 8 Multi-port NIC Example

12. Appendix

12.1. Glossary and Abbreviations

Acronym	Description
BDP	Bandwidth-Delay Product
CC	Congestion control
CWND	Congestion Window
EV	Entropy value – Multipath selector communicated via various fields in the packet
LID	SRv6 Segment Routing Locator Identifier
MPR	Maximum PSN Range
MRC	Multipath Reliable Connection (as defined in this document)
NSCC	Network-Signalled Congestion Control
OOO	Out-of-order
QPCC	QP Congestion Controller
RC	Reliable Connection transport connection as defined in [IBTAVS16]
RQ	Receive Queue
SACK	Selective ACK: Reliability message used to deliver bitmap information
SRH	SRv6 Segment Routing Header
SRv6	Segment Routing v6
SW	Software (host application and/or middleware)
TRIM	Packet which is truncated and forwarded to receiver
uSID	SRv6 Micro Segment Identifier (a LID and stack of uSIDs make up an SRv6 address)
WR	Work Request

12.2. References

Document	Details
[IBTASPEC]	InfiniBandTM Architecture Specification Volume 1 Release 1.8
[MRCSW]	https://github.com/opencomputeproject/OCP-Multipath-Reliable-Connection
[RFC2119]	Key words for use in RFCs to Indicate Requirement Levels
[RFC8986]	Segment Routing over IPv6 (SRv6) Network Programming
[RFC9800]	Compressed Srv6 Segment List Encoding
[SRV6USID]	Network Programming extension: SRv6 uSID instruction
[UESPEC]	UltraEthernet 1.01 Specification