Falcon: A Reliable, Low Latency Hardware Transport

Arjun Singhvi, Nandita Dukkipati, Prashant Chandra, Hassan M. G. Wassel, Naveen Kr. Sharma, Anthony Rebello, Henry Schuh, Praveen Kumar, Behnam Montazeri, Neelesh Bansod, Sarin Thomas, Inho Cho, Hyojeong Lee Seibert, Baijun Wu, Rui Yang, Yuliang Li, Kai Huang, Qianwen Yin, Abhishek Agarwal, Srinivas Vaduvatha*, Weihuang Wang*, Masoud Moshref*, Tao Ji*, David Wetherall, and Amin Vahdat
Google LLC

ABSTRACT

Hardware transports such as RoCE deliver high performance with minimal host CPU, but are best suited to special-purpose deployments that limit their use, e.g., backend networks or Ethernet with Priority Flow Control (PFC). We introduce *Falcon*, the first hardware transport that supports multiple Upper Layer Protocols (ULPs) and heterogeneous application workloads in general-purpose Ethernet datacenter environments (with losses and without special switch support). Key design elements include: delay-based congestion control with multipath load balancing; a layered design with a simple request-response transaction interface for multi-ULP support; hardware-based retransmissions and error-handling for scalability; and a programmable engine for flexibility. The first Falcon hardware implementation delivers a peak performance of 200 Gbps, 120 Mops/sec, with near-optimal operation completion times that are up to 8× lower than CX-7 RoCE under network congestion, and up to 65% higher goodput under lossy conditions.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Data center networks**.

KEYWORDS

Hardware Transport, Datacenter Networks, Remote Direct Memory Access, Network Interface Card

ACM Reference Format:
Arjun Singhvi, Nandita Dukkipati, Prashant Chandra, Hassan M. G. Wassel, Naveen Kr. Sharma, Anthony Rebello, Henry Schuh, Praveen Kumar, Behnam Montazeri, Neelesh Bansod, Sarin Thomas, Inho Cho, Hyojeong Lee Seibert, Baijun Wu, Rui Yang, Yuliang Li, Kai Huang, Qianwen Yin, Abhishek Agarwal, Srinivas Vaduvatha, Weihuang Wang, Masoud Moshref, Tao Ji, David Wetherall, and Amin Vahdat. 2025. Falcon: A Reliable, Low Latency Hardware Transport. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3718958.3754353

1 INTRODUCTION

Modern datacenter workloads demand excellent networking performance. Applications such as high-performance computing, scaleout AI/ML, and real-time analytics need high bandwidth and low latency transfers. Modern storage systems need high Read/Write operation rates to fully utilize high-performance SSD devices. And the networking stack must deliver this performance while consuming minimal host CPU, so that the CPU may be used by applications.

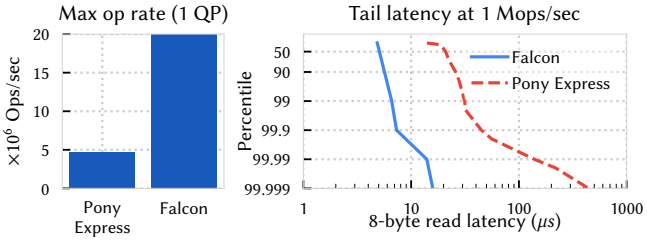


Figure 1: Comparing the limits of SW-based stacks

Hardware-accelerated transports, where the transport datapath is implemented in hardware, can meet these stringent requirements in ways that transports implemented in software cannot. By taking advantage of dedicated hardware resources, these transports can simultaneously provide high bandwidth and operation rates, and keep latency low without incurring significant CPU core usage. The Falcon transport we present in this paper achieves a 5× higher operation rate and 10× lower tail-latency than Pony Express [33], a highly-optimized software transport (Figure 1).

Our goal is to develop a transport that combines hardware-accelerated performance with the generality of software transports that support heterogeneous workloads over diverse network conditions. As a usage setting, we imagine a datacenter network based on commodity Ethernet that connects any mix of compute servers, storage, and accelerator slices, and which supports mixed workloads over industry-standard interfaces such as IB Verbs and NVMe.

Current hardware transports like RDMA over Converged Ethernet (RoCE) are better suited for specialized use cases than general-purpose datacenter use. Derived from RDMA on Infiniband networks, RoCE assumes infrequent packet loss. It is typically used with Priority Flow Control (PFC), a complex feature with deployment challenges [24]. Without PFC, RoCE is typically deployed in a fully-provisioned network with specialized switches (e.g., Spectrum-X Adaptive Routing [10]), limiting its use to dedicated backend networks. Technologies like NVLink [21] and Inter-Chip Interconnect [27] provide dedicated hardware transports for GPU and TPU traffic. They provide islands of high performance that are difficult to scale and unsuitable for general-purpose workloads. Furthermore, hardware transports also lag software-based transports in terms of innovations for reliability, congestion control and multipathing.

In this paper, we present *Falcon*, a hardware-accelerated transport that we have implemented with RDMA and NVMe in a 200G NIC for use in datacenters. Falcon leverages proven transport techniques to achieve high performance and efficient operation in lossy Ethernet networks, including Swift [29], RACK-TLP [17], Snap [33], Protective Load Balancing [41], Protective Reroute [53], Carousel [45] and multipathing.

*Work done while at Google.

SIGCOMM '25, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/2025/09
https://doi.org/10.1145/3718958.3754353

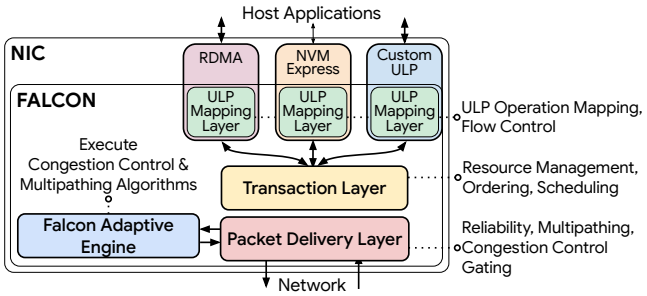


Figure 2: Falcon hardware transport layers

Our contribution is the first hardware transport that supports multiple Upper Layer Protocols (ULPs) and heterogeneous application workloads in general-purpose Ethernet datacenter environments. Falcon delivers excellent performance under diverse network conditions (including loss, reordering and path diversity) with a datapath implemented entirely in hardware. It does not need specialized switch features like PFC or Adaptive Routing. And it excels across both specialized workloads (e.g., ML traffic) and general-purpose workloads (e.g., server incast). We achieve this result with the design choices seen in Figure 2:

(1) Layered design with a simple transaction interface that supports IB Verbs, NVMe, and other ULPs. The ULP operations are mapped to transactions with flexibility for variable-length operations, ordering semantics, and ULP-specific error-handling. This layering reduces feature redundancy across ULPs and NIC hardware complexity.

(2) A lower layer of *reliable packet delivery with delay-based congestion control and multipath load balancing*. Both work well together and have hardware support for fine-grained traffic pacing, retransmissions using Selective Acknowledgements, and round-trip time measurements.

(3) NIC hardware resource admission with backpressure for retransmissions, operation ordering, and transaction error handling. This enables line-rate operation in a resource-constrained environment without performance cliffs under O(100K) connections.

(4) Transport feature programmability via the Falcon Adaptive Engine (FAE). It strategically partitions functionality between HW for line-rate performance and SW for flexible event processing.

We demonstrate that Falcon delivers peak performance of 120 Mops/sec for RDMA and NVMe applications with *near-optimal* operation completion times, even under network congestion. Compared to CX-7 RoCE, Falcon achieves up to 8× lower completion times and up to 65% higher goodput under lossy conditions.

2 REQUIREMENTS AND WHY EXISTING HW TRANSPORTS FALL SHORT

To meet the demands of diverse applications and real-world network conditions, we find that a hardware transport must fulfill the following five key requirements:

(R1) Predictable Performance: Ensure near-optimal completion times for network transfers. This is crucial for datacenter applications.

(R2) Adaptability to Diverse Networks: Maintain predictable performance across network deployments, including those with oversubscription, multi-tenancy, and heterogeneous topologies.

(R3) Robustness under stress: Handle high bandwidth, operation rates, and low latency without performance cliffs. Do so even when under stress from factors such as many connections, out-of-order network packets, and retransmissions.

(R4) Support for Standard Interfaces: Support IB Verbs and NVMe interfaces without requiring application modifications. Maintaining compatibility with the existing ecosystem of applications optimized for these interfaces is essential.

(R5) Operational Ease: Allow for fixing performance issues and evolution of the transport components without necessitating new hardware.

The dominant hardware transport, RoCE [24], struggles to meet these requirements. RoCE supports the IB Verbs interface (though not NVMe in *R*4), but it is ineffective at handling lost and reordered packets for high application performance (*R*1-*R*3). Fundamentally, RoCE inherits three key limitations from the lossless nature of RDMA-over-Infiniband that do not scale well to lossy networks.

First, RoCE does not support modern loss recovery. RoCE initially relied on Go-Back-N style loss recovery, even for single packet losses. Proprietary extensions now enable Selective Repeat (SR), but with significant restrictions. We find that SR is supported for RDMA Writes and Read Responses, but other operations remain limited to Go-Back-N recovery. The SR mechanism, which sends a Negative Acknowledgment for each out-of-order packet, can lead to slow, imprecise recovery and high tail latency (see §6.1.1). And when using SR, loss typically leads to out-of-order packet delivery, which violates IB Verbs semantics.

Adding state-of-the-art loss recovery like Selective Acknowledgments (SACK) to RoCE is non-trivial while also satisfying IB Verbs ordering semantics. RoCE NICs lack resources like packet buffers. To maintain ordering, a RoCE receiver drops out-of-order packets following a loss. This limits its ability to precisely signal missing packets to the sender. To implement precise loss signaling like SACK, RoCE NICs would require substantial changes, such as adding on-NIC packet buffers on the receive side or modifying the ordering semantics. Thus, for many use cases, Priority Flow Control remains necessary to avoid losses.

Second, RoCE does not have protocol support for multipathing. This causes complications when it is used with Adaptive Routing at switches to utilize all network paths (as is common in ML backend networks). We find that tolerating the natural reordering of multipath adversely impacts loss recovery. This leads to poor performance (see §6.1.1) so PFC is often used to avoid loss. Reordered packets are also delivered out-of-order, which is problematic for general-purpose use because it violates IB Verbs semantics.

Third, RoCE does not integrate congestion control with the datapath. Congestion control is implemented as an add-on, relying on out-of-band probes [6] to gather congestion signals. This separation makes its congestion response sluggish.

Additionally, RoCE NICs handle timeouts, errors and exceptions in firmware. This makes it susceptible to performance cliffs under stress, falling short of *R*3. We present RoCE's performance under losses, reordering, and network congestion in §6.

3 OVERVIEW & DESIGN PRINCIPLES

We first provide an overview of Falcon, then describe its built-in protocol specializations and the design principles that enable its hardware implementation. Requirements *R*1-*R*2 ensure good end-to-end application performance, while *R*3-*R*5 guide the hardware implementation for peak operation **(op)** rates, low latency and compatibility with existing interfaces. Table 1 summarizes how the

requirements defined in §2 are met with the design principles we describe below.

3.1 Falcon Overview

Falcon is a connection-oriented transport protocol providing end-to-end reliability for various upper-layer protocols **(ULPs)** like RDMA and NVMe, as illustrated in Figure 2. Falcon consists of four components. The *ULP mapping layer* translates operations from upper-layer protocols (like RDMA Reads and Writes) into Falcon connections, and handles flow control. The *Transaction layer* **(TL)** provides a request-response interface to upper-layer protocols, managing on-NIC resources, scheduling, and ordering. The *Packet Delivery layer* **(PDL)** enforces delay-based congestion control **(CC)**, paces traffic with the Timing Wheel, and ensures rapid loss recovery with SACK, all with multipathed connections. A programmable *Falcon Adaptive Engine* **(FAE)** works in conjunction with the PDL to implement CC and multipath load balancing. This programmability enables adaptation in deployments while preserving high-speed performance. Falcon can operate with inline encryption for end-to-end security. For this purpose, Falcon can utilize protocols such as the Paddywhack Security Protocol [2] (PSP) or IP-SEC [28] for authentication and encryption of data transferred over a connection.

Applications do not directly interact with Falcon but instead use standard APIs provided by the ULPs like RDMA and NVMe. These ULPs communicate with Falcon through a request-response transaction interface, with necessary adaptations (see the ULP mapping layer in Figure 2) to map their operations to Falcon transactions.

Falcon's layered design mirrors how Pony Express [33] manages interactions with applications, internal resources, and network behaviors. Falcon's Transaction and Packet Delivery layers are analogous to Pony Express Op and Reliability layers. A key difference is that Pony Express relies on ULPs for ordering and performs segmentation in its Op layer, while Falcon's Transaction layer manages ULP operation ordering, with segmentation handled by the ULPs.

3.2 Predictable Performance in Diverse Networks

Many approaches in the literature aim to provide predictable network performance (Table 5 provides a summary). Most do not meet our requirements as they rely on assumptions that do not hold in real networks. Receiver-driven schemes, like Homa [37], NDP [25], EQDS [39] and pHost [22], assume no over-subscription within the network, which is often untrue. Load balancing schemes like packet spraying require homogeneity in network paths and traffic use, including WAN traffic, which doesn't hold in real networks. Our first design tenet is:

D1. *Delay-based congestion control* and *Multipath load balancing*: these come from SW transports (Pony Express [33]/TCP) whose design and production experience informed Falcon. Delay-based CC (Swift [29]) utilizes hardware-assisted per-packet round-trip time (RTT) measurements to modulate the transmission rate. Multiple HW/SW timestamps decompose latency across the host, NIC, and network fabric. This enables a fast and precise congestion response. Delay-based CC also alleviates congestion in end-hosts/NICs, like PCIe and Memory Bus congestion [12]. Additionally, fine-grained traffic pacing via the Timing Wheel (Carousel [45]) improves fairness across tenants, and shortens network queues under high-degree congestion (when the number of flows exceeds the bandwidth-delay product in packets).

Multipath load balancing is used by our SW transports to leverage the many paths in datacenter topologies. Multipathing uses multiple flows within each connection. Each flow may traverse a different network path. Each flow rate dynamically adapts to path congestion, so that less congested paths draw more traffic from the connection. This *just-in-time scheduling*, combined with flow repathing (Protective Load Balancing [41]) to shift traffic away from congested paths, minimizes end-to-end latency. Individual flows use Protective Reroute [53] to mitigate the impact of network outages. They change the IPv6 flow label on their packets when an outage is detected, causing switches to choose a different network path. Figure 3 demonstrates the improvements in op latency of transport-level multipathing in Pony Express for an ML workload, compared to the application naively creating multiple connections.

For optimal application performance, loss recovery needs to work efficiently despite the packet reordering inherent to multipath load balancing. Falcon leverages Selective Acknowledgements (SACK) [35] for efficient retransmission of only lost packets. We further adapted RACK-TLP [17] (Recent Acknowledgment - Tail Loss Probe) for Falcon's hardware because time-based differentiation between packet loss and reordering speeds recovery, especially for losses at the end of data bursts.

Requirement	Design Principle
R1,**R2**: Predictable Performance under Diverse Networks	**D1**: Delay-based CC and Multipath load balancing (§6.1)
R3: Robustness and no performance cliffs	**D4**: Hardware resource admission, Backpressure, Error-handling (§6.2)
R4: Standard ULPs, multiple ULP support	**D2**: Layered Design **D3**: Simple Interface to ULPs (§6.3)
R5: Rapid Iterations and Operational Ease	**D5**: HW-assisted Programmable Engine (§6.2)

Table 1: Mapping Requirements to Design Principles

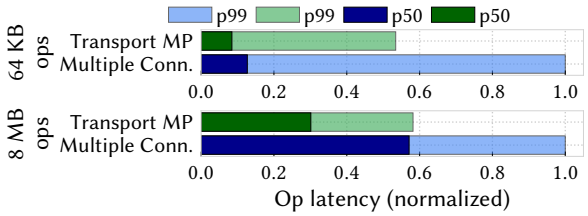


Figure 3: Multipathing benefits ML Workloads

3.3 Hardware Design: Challenges and Principles

Building a hardware transport like Falcon presents several challenges. We had to simultaneously deliver *high peak performance* for HPC and AI/ML applications and *predictable sustained performance* in warehouse scale deployments. Achieving peak op rates, bandwidth, and low latency is key – not just in the fast path when packets arrive perfectly in order, but under network conditions that create out-of-order packets, retransmissions, timeouts and application error conditions. These complications are common in production networks. At the same time, sustained performance that avoids cliffs with O(100K) connections is also crucial. These challenges necessitate careful consideration of hardware constraints of on-die space, buffers, power, and involve addressing bottlenecks like connection cache misses and handling exceptions/errors efficiently in hardware. Unlike software stacks, which can elastically use CPU cores on the machines, hardware implementations must optimize resource utilization and performance within the NIC's

pre-determined resources. We list the design principles that enabled Falcon's high-performance hardware implementation.

D2. *Layered Design:* We made a clear division of responsibilities between Falcon and ULPs with the aim of reducing feature duplication across ULPs. This approach reduces NIC hardware complexity by centralizing core transport functions within Falcon. Falcon's Transaction and Packet Delivery layer split further divides responsibilities for independent implementation and evolution.

D3. *Simple interface supporting multiple ULPs:* Falcon's multi-ULP support is enabled by a simple transaction interface, wherein each transaction consists of a request and corresponding response. It aligns well with common ULP operations. This design facilitates adding new ULPs, including those with custom ordering modes and error notification requirements.

D4. *NIC hardware resource admission with backpressure, and error-handling:* A guiding principle was to eliminate performance cliffs caused by out-of-order network packets and error conditions. To do so, Falcon utilizes dedicated *on-chip memory* and hardware-based retransmissions that support IB Verbs ordering semantics. A Falcon receiver absorbs out-of-order packets using its on-chip packet buffers, which are sized based on bandwidth-delay product. It signals missing packets to the sender, which performs precise retransmissions – Falcon is able to do so for *all* IB Verbs operations without violating their semantics. Per-connection backpressure provides isolation across connections and keeps on-NIC resource occupancy low. Falcon further avoids performance cliffs caused by various error scenarios by implementing exception handling in hardware. The cost of integrating the Falcon IP into the NIC has been minimal: 5-6% in terms of die size and 3-4% in terms of power.

D5. *Hardware-Assisted Programmable Engine:* One of our guiding principles was to enable rapid iteration of congestion control and other transport features in software by decoupling the algorithms from the datapath. To avoid HW complexity and inflexibility, our design separates the *management* of transport features (implemented as a programmable engine running as software on a general purpose CPU) from the *mechanism* (implemented in the HW path). Table 3 shows the split between hardware, software, and interface signals for various transport features.

4 DETAILED DESIGN

We now detail the components of Falcon (Figure 2). PDL and FAE together ensure reliability (§4.1), congestion control (§4.2), and multipathing (§4.3). TL supports multiple ULPs with flexible ordering and error semantics (§4.4), manages NIC resources (§4.5), and isolates connections (§4.6).

4.1 Reliability

Falcon ensures end-to-end reliability from ULP payload transmission to ULP reception. Its primary goal is rapid recovery from packet drops without excessive retransmissions, regardless of network conditions like reordering or loss. Similar to Pony Express, Falcon's receivers use bitmaps to precisely communicate received and missing packets to the sender. We chose bitmaps over TCP's variable-length Selective Acknowledgment (SACK) blocks for easier hardware processing. Like TCP, Falcon employs RACK-TLP [17] to differentiate packet loss from reordering and quickly recover from losses at the end of transmission bursts. We detail the roles of the receiver and sender PDL in achieving reliable delivery.

Receiver. The receiver's PDL tracks received packets by Packet Sequence Numbers (PSNs) with a bitmap-based Rx window. This bitmap is piggybacked on outgoing Falcon ACKs. It helps the sender identify lost packets and trigger retransmissions (as we see next). The bitmap size balances hardware implementation feasibility and ACK overhead; 128-bit bitmaps worked well for our use cases. The receiver accepts out-of-order packets, even those outside the Rx window, provided sufficient on-NIC resources are available.

Sender. On receiving an ACK, the sender employs the Recent Acknowledgement (RACK) heuristic to identify packets for retransmission. It analyzes the ACK's timestamp and the accompanying Rx bitmap as follows: Packets marked as received are ignored. Outstanding packets transmitted after time $xmit\_ts$ are also ignored to ensure RACK doesn't trigger retransmissions until sufficient time (~RTT) has elapsed. Packets sent before $xmit\_ts$ are retransmitted if their elapsed time exceeds the $rack\_rto$ duration. This heuristic applies only to packets within the Rx bitmap's range, as the reception status of packets beyond it is unknown to the sender.

To handle tail losses, the sender also uses the Tail Loss Probe (TLP) heuristic. After a period of inactivity (e.g., due to tail losses or lost ACKs), the sender retransmits a probe packet, specifically the lowest unacknowledged PSN. This probe prompts the receiver to generate an ACK. On receiving this ACK, the sender uses the RACK heuristic to trigger early retransmissions if warranted.

FAE Role. It calculates loss recovery parameters (e.g., retransmission and TLP timeouts) based on network conditions. This allows Falcon to adapt the aggressiveness of recovery to a wide range of network scenarios.

4.2 Programmable Congestion Control

Falcon implements congestion control for each connection by splitting responsibility across the PDL and FAE. FAE implements the CC algorithms that compute congestion windows based on the CC signals, while the PDL is responsible for measuring CC signals and enforcing the computed windows via the connection scheduler and Timing Wheel. Such a division of labor allows the CC algorithm to evolve via the programmable FAE without changing the PDL, which implements key per-packet operations efficiently. Falcon's CC algorithm is a variant of Swift [29] adapted to handle both fabric and NIC congestion via two congestion windows—*fcwnd* (fabric congestion window) and *ncwnd* (NIC congestion window)—with the effective window being the minimum of the two.

Handling Fabric Congestion. We use fabric delay as the congestion signal. To accurately estimate fabric delay, Falcon relies on precise hardware timestamps. t_1 is the time at which the packet is sent on the wire, and is carried in the packet header. t_2 is when the packet arrives at the remote NIC. The remote NIC stores t_1 and t_2 of the latest packet for the connection. t_3 is the time when the remote NIC sends an ACK, and t_4 is when it arrives at the local NIC. The ACK header carries t_3 along with the latest t_1 and t_2 values, so the local PDL has all the timestamps needed to compute the delay as $(t_4 - t_1) - (t_3 - t_2)$ without synchronized clocks.

Handling Rx NIC Congestion. Falcon CC explicitly handles NIC pipeline congestion, particularly for Rx buffers, which can become exhausted under two scenarios. First, ordered connections that experience losses buffer OOO packets. This leads to buffer buildup while awaiting retransmissions of lost packets. Second, end-host bottlenecks (PCIe, IOMMU or memory [12]) at the receiver can cause the NIC's host interface (see Figure 7) to backpressure the ULP, which then backpressures Falcon. This leads to buffer buildup until

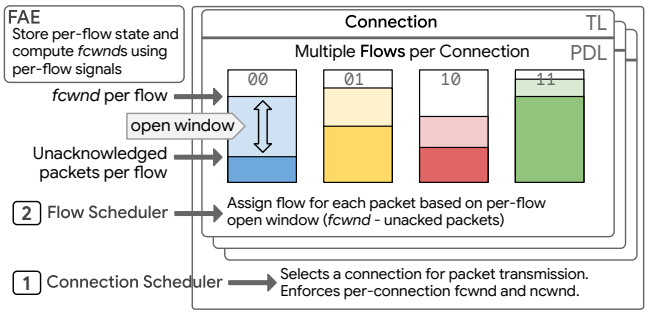


Figure 4: Congestion-aware Multipathing in Falcon.

packets received from the network can be forwarded to the ULP. Falcon uses NIC Rx buffer occupancy, a readily available hardware counter, as the NIC congestion signal. ACKs carry this signal back to the sender which uses it for *ncwnd* modulation. This prevents excessive packet drops due to the lack of NIC Rx resources.

4.3 Multipathing

Multipathing is a core design element that enables Falcon to achieve high per-connection performance by allowing a single connection to use multiple paths. Our key insight is to use these paths in a congestion-aware manner: favoring less-congested paths lowers op latencies and sustains higher throughput. Falcon aims to transmit each packet on the best path to the destination. This goal requires two decisions: selecting the set of paths for a connection, and assigning a specific path to each packet. Both decisions are a natural fit for the PDL, letting it isolate the multipathing mechanics from the upper layers.

As shown in Figure 4, Falcon maintains an indexed list of *flows* per connection. Each flow is assigned a unique IPv6 Flow Label that is set on outgoing packets of the flow. This Flow Label also includes the flow's index to facilitate tracking per-flow state. Switches hash on the flow label in addition to the ECMP/WCMP 4-tuple, so that each flow maps to a potentially different network path [41].

Although flows share the connection's PSN space (§4.1), Falcon tracks congestion state for each individual flow and computes a dedicated *fcwnd* (§4.2). The PDL: [1] enforces a connection-level *fcwnd* computed by aggregating the per-flow *fcwnd* values, and [2] maps packets to flows based on the flow-level *fcwnd* values. This design embodies a classic accuracy-complexity trade-off. At one extreme, PDL could simply spray packets across flows oblivious of per-flow congestion, requiring simpler HW. The other extreme, enforcing *fcwnd* per flow with separate PSN spaces, would introduce additional HW complexity. We find that performing CC by combining connection-level *fcwnd* gating and flow-level *fcwnd* scheduling offers a good balance. We now describe these mechanisms in detail by tracing events during a network round trip.

Congestion Control. The PDL transmits a packet when the connection congestion window (minimum of *ncwnd* and *fcwnd*) opens up. When ready, the PDL selects the flow with the largest open window, defined as the difference between the flow *fcwnd* and the number of unacknowledged packets for the flow, and applies the corresponding Flow Label. The packet then travels along the network path dictated by its Flow Label to reach the receiver. The receiver extracts the flow index from the Flow Label and updates per-flow congestion metadata, such as timestamps. It also implements per-flow ACK coalescing. When an ACK is generated for a flow, the PDL populates it with the congestion metadata specific to that flow. Upon receiving the ACK packet, the PDL identifies the flow using the flow index carried in the ACK. It then extracts per-flow congestion metadata from the ACK, with the exception of the number of acknowledged packets for the flow. This is because a single ACK may acknowledge PSNs from multiple flows for the connection. To accurately calculate acknowledged packets per flow, the PDL uses its internal state which tracks the flow index associated with each unacknowledged packet. PDL sends these per-flow congestion signals to FAE.

ULP	Operation	Falcon Transaction
RDMA	WRITE, SEND/RECV	Push
RDMA	READ, ATOMICS	Pull
NVMe	Read	Pull
NVMe	Write	Push and Pull

Table 2: RDMA and NVMe ops mapped to Falcon transactions.

FAE Role. Leveraging the per-flow congestion signals provided by PDL, FAE computes the flow's new *fcwnd* by executing the CC algorithm. The updated *fcwnd* is sent back to the PDL for enforcement, as previously described. Beyond CC, FAE also implements PLB [41] and PRR [53] to determine the Flow Label to use for each flow based on path state. To facilitate these computations, FAE maintains essential algorithm state per flow.

Reliability. Packets within a single connection share a common PSN space, regardless of the flow they traverse. An implication of this design choice is that packets across flows may arrive out of PSN order. Falcon's reliability mechanisms (§4.1) handle these conditions well by robustly disambiguating reordering from losses. The Rx bitmap in an ACK allows tracking acknowledged packets across flows, i.e., if a packet or an ACK is lost for one flow, the Rx bitmap from ACKs for other flows will still inform the sender of the aggregate state at the receiver and trigger SACK-based loss recovery appropriately. Moreover, Falcon implements RACK-TLP per flow to avoid spurious retransmissions due to reordering.

4.4 Support for Multiple Upper Layer Protocols

Falcon's TL interfaces with multiple ULPs, which in turn expose well-defined APIs (e.g., IB Verbs in case of RDMA) that are used by applications. We discuss this interface and how it provides flexible ordering and error semantics.

Interface to ULPs. Falcon supports multiple ULPs through a well-defined request-response *transaction interface*. Each *Push* or *Pull* transaction comprises a request and its corresponding response. In both kinds of transactions, the requester Falcon sends a request packet to the responder Falcon, which returns a response packet. The key difference is the direction of data flow: Push is used for operations (ops) where the requester is sending data to the responder (e.g., writes). Pull is used for ops where the requester is receiving data from the responder (e.g., reads). Table 2 summarizes the mapping of ULP ops to Push/Pull transactions.

Two questions need to be addressed by the design of the transaction interface: (1) Should the interface be oblivious to the type of transaction being performed by the ULP? and (2) What is the granularity of the interface?

Transaction Type Awareness. Unlike TCP, which employs a byte stream-oriented model, we designed the Falcon interface to be aware of the transaction type to avoid deadlocks due to finite NIC

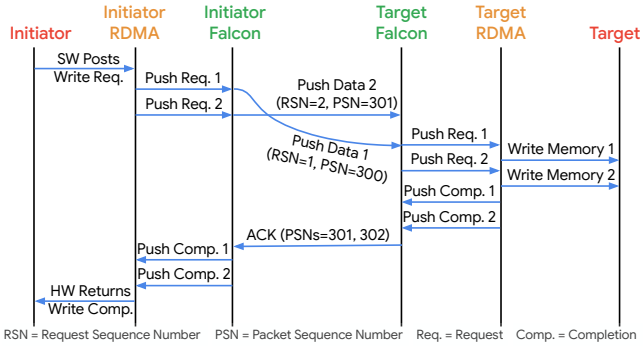


Figure 5: RDMA Write Operation. To initiate an RDMA Write (larger than one MTU), the software posts a request, prompting the NIC's RDMA ULP to issue two Falcon Push Requests, each within the MTU limit. The initiator transmits two push data packets (RSN=1 and 2) to the target. The PDL assigns them PSN=300 and 301. These packets arrive out of order due to network reordering. The target's Transaction layer correctly orders the two packets based on their RSN before delivering them to the NIC's RDMA ULP. The ULP executes the memory writes and returns Push Completions to Falcon, which then triggers a single ACK packet acknowledging both push data packets. Upon receiving the ACK, the initiating Falcon connection passes two completions to the RDMA block, which then generates a write completion and posts it to the completion queue.

resources (§4.5). Being type-aware enables the TL to assign requests and responses from the ULP to separate PSN spaces to avoid a request-response deadlock (§A.1). Additionally, it helps with performance – it enables Falcon to (a) accurately enforce congestion control based on packet types (as Pull Responses are not subject to *ncnwd* because the requester Falcon has reserved resources to land the requested responses; see §4.5); and (b) schedule requests flowing in one direction (or responses in reverse) in order by assigning them to the appropriate request-response scheduler queue.

MTU Granularity. With Falcon, we decided that transactions can be no more than one MTU in length (e.g., a large READ op is broken into multiple MTU-size Pull transactions). This choice simplifies the hardware as each request maps to a single response. Also, MTU-sized transactions enable fine grained management of hardware resources, allowing arbitration based on application priorities.

Flexible Ordering Semantics. Each Falcon connection can be configured to be either *ordered* or *unordered*. The ordered mode reflects the IB Verbs specification ordering, which requires in-order data placement and in-order completions. This mode enables legacy RDMA applications to run atop Falcon. Unordered mode, as the name suggests, corresponds to out-of-order (OOO) data placement with OOO completions. It caters to modern datacenter applications that issue independent ops.

To enforce ordering when required, the TL relies on each transaction being uniquely identified by a request sequence number (RSN), and orders packets per the RSN. (The PSN alone is not sufficient to ensure ordering; see §A.2). Crucially, the on-NIC resources described in §4.5 enable Falcon to accept OOO packets, buffer them while awaiting earlier packets, and then deliver them in order to the ULP. With unordered connections, the TL sends packets to the ULP as they arrive. Additionally, *weakly ordered* semantics (OOO data placement with in-order completions), which reflects the iWARP model [43], can be achieved by the ULP doing completion ordering atop an unordered connection. Tying it together, Figure 5 illustrates the flow of an RDMA Write op over an ordered Falcon connection.

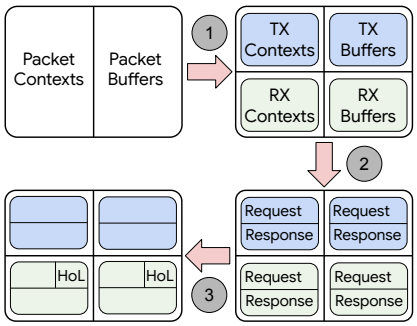


Figure 6: Falcon Resource Carving to avoid Deadlocks

Enhanced Error Notifications. Falcon supports existing ULP error handling semantics, including the Receiver Not Ready (RNR) indication that allows a target ULP to specify a retry time for a transaction. Falcon handles RNR retries transparent to the ULPs. Falcon also supports an enhanced fine-grained error notification, *Complete in Error and Continue (CIE)*. It allows more graceful handling of errors like RDMA memory protection errors that do not need to be fatal. The target ULP returns a CIE indication with an error code, and Falcon transmits a CIE NACK to the initiator, which completes the initiator's transaction with an error. Subsequent transactions continue to finish successfully.

4.5 Managing Hardware Resources

Falcon introduces on-NIC resources to natively support OOO packet arrivals and ensure reliability for all its ULPs. Falcon uses two kinds of hardware resources: *contexts* and *buffers*. Contexts hold fixed-size metadata (e.g., sequence numbers for ordering and reliability, resources held, packet headers, etc.). Buffers hold variable-size metadata (e.g., Scatter Gather Lists) and the actual packet payload.

These resources are finite and sized to meet the op-rate and bandwidth targets of a given Falcon generation, while allowing slack for RTT jitter or variation. The TL needs to carefully manage Falcon resources to avoid deadlocks that can occur in request-response protocols. We now describe resource carving and the reserve-use-release lifecycle which are essential to protocol forward progress.

Resource Carving. Resources in Falcon are carved into multiple sub-pools based on three intuitive principles to avoid deadlocks (Figure 6). First, the resources are carved between packets to be transmitted (Tx) and received (Rx) so that they can be allocated independently. Not doing so can lead to scenarios where all the resources are used by outgoing packets, preventing incoming packets from being accepted, or vice versa, leading to no forward progress. Second, the Tx and Rx resources are further divided between requests and responses so that both of them can make independent progress. Not having this separation can lead to scenarios wherein outstanding requests prevent responses from being sent as they use up all the resources or vice versa. Lastly, the Rx request resource pool allows only head-of-line (HoL) requests to be admitted beyond a resource occupancy threshold. In case of ordered connections, a request is HoL when its RSN is one more than the RSN of the last in-order request received; recall that order is determined by RSN (§4.4). On unordered connections, all requests are considered to be HoL. Not admitting HoL requests can lead to a situation where all the resources are occupied by non-HoL requests and the protocol makes no forward progress for ordered connections.

Feature	Hardware Mechanism	Interface Signals	Software (FAE) Tasks
Reliability, Loss Recovery	ACK Bitmap, RACK, TLP	Timestamps, Rx Buffer Level	Compute RTO, RACK timeout, TLP threshold
Congestion/Flow Control	Sliding Windows, Pacing	#hops, ECN, RTO, CSIG [42]	Compute fcwnd, ncwnd, IPG, target delay scaling
Multipathing, PLB [41], PRR [53]	Path-aware flow scheduler	Flow Labels	Flow Label (re)assignment and per-flow CC tasks
Resource Isolation	Dynamic Thresholds	Resource Occupancy	Compute DT alpha based on dynamic factors
Pure ACK Generation	Ack Request Bit in Header	Ack Request Rate	Decide when/how to set the AR bit

Table 3: Split of responsibilities between hardware and software in Falcon

Resource Lifecycle. Each transaction holds various resources during its lifetime. On receiving requests from ULP, the initiator Falcon reserves not only the Tx resources for the request, but also Rx resources for the corresponding response (completions for Push; Pull Response for Pull). This approach naturally eliminates the formation of reverse path incasts as the initiator Falcon sends out requests only if it can land the corresponding response. More importantly, doing so also ensures that incoming HoL responses always have resources reserved, and thus ensures that deadlocks due to resource unavailability for HoL responses do not occur. On the other side, the target Falcon uses Rx (Tx) resources when requests (responses) arrive from the network (ULP). Rx resources are released once received packets are delivered to ULP. Tx resources are released when the ACK corresponding to the transmitted packet is received.

When Falcon runs out of resources to handle new transactions from ULPs, it backpressures them to not send any additional transactions via an Xon/Xoff flow control interface. Resource NACKs are generated by Falcon when there are not enough resources to accept incoming network traffic.

4.6 Ensuring Isolation

Falcon provides isolation between connections. Without isolation, a connection making slow progress (and thus holding onto resources longer) could deprive other connections of resources, hindering their performance. Isolation is implemented in Falcon rather than the ULPs, since (a) it has direct visibility into resource usage and connection progress, both of which are required to ensure isolation, as we discuss below; and (b) not doing so would require ULPs to have more visibility into Falcon and unnecessarily replicate the isolation logic in each ULP.

We equipped the TL to exert *fine-grained backpressure* to ULPs via a per-connection Xon/Xoff flow control interface whenever a connection exceeds its resource usage threshold. This enables Falcon to backpressure slow connections, preventing them from sending transactions, and thus allow other connections to continue making progress. Falcon calculates the backpressure thresholds dynamically and takes a similar approach to the Dynamic Thresholds (DT) algorithm that is commonly used to manage datacenter switch buffers [18]. Similar to DT, Falcon calculates the threshold T_c for a connection c as $T_c = \alpha_c \cdot FreeResources$. The α_c parameter of a connection determines its limit relative to other connections - a larger value implies a larger share and vice-versa.

FAE Role. Unlike DT, wherein α_c is static, FAE calculates α_c for a connection dynamically as $\alpha_c = \beta_c \cdot \alpha$, where β_c is a *congestion-aware* variable proportional to *fcwnd* and *ncwnd*, but inversely proportional to fabric delay and buffer occupancy. For example, in the case of fabric (host) congestion, if the *fcwnd* (*ncwnd*) is low and/or the fabric delay (buffer occupancy) is high, it indicates the connection is making slow progress and requires fewer resources. Falcon scales down α_c accordingly to decrease the connection's backpressure threshold relative to a congestion-free connection.

5 BUILDING A SCALABLE HW TRANSPORT

We describe our experience implementing Falcon transport along with some of the notable choices we made.

5.1 Building and Integrating Falcon into NICs

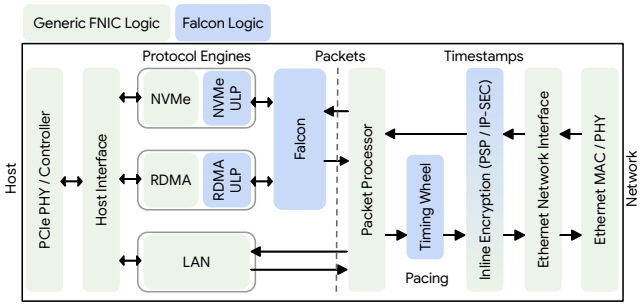


Figure 7: Falcon Implementation in a NIC

We built Falcon as an independent HW block that could be integrated into vendor foundational NICs (FNICs) or Infrastructure Processing Units (IPUs). We have integrated the Falcon block in an Intel E2100 200Gbps IPU, and have scaled the Falcon block for 400Gbps and 800Gbps speeds. Falcon is integrated as an IP block on the same die as the rest of the NIC. Using an ASIC implementation allowed Falcon to deliver on its peak op rate and low latency requirements. Alternatives such as FPGA-based implementations were not pursued due to its higher total cost of ownership (TCO) and worse performance in terms of op rate and low latency.

Figure 7 shows Falcon HW integrated into a generic FNIC, interfacing with ULPs on one side and packet processing block on the other side. Implementing Falcon in a NIC requires additional changes beyond the integration. Changes within the ULPs are done to map their ops to Falcon transactions. Given Falcon also relies on a Timing Wheel (TW) to potentially pace packets, we introduced a standalone TW block; alternatively it can be implemented within Falcon HW itself. Falcon also relies on timestamping of packets close to the Ethernet port to enable precise fabric RTT measurements. Typically this timestamping function is implemented within the inline encryption block (using the IV field of the PSP [2] or IP-SEC [28] protocols). Figure 8 shows the Falcon HW block diagram, featuring distinct transmit and receive pipelines with shared control logic—a typical structure for HW transport implementations.

5.2 Notable Implementation Choices

We highlight aspects of Falcon's hardware implementation that deviate from conventional wisdom. Our goal is to ensure that the implementation is not susceptible to performance cliffs so it can cater to warehouse scale datacenter applications that often involve O(100K) communicating processes spread across O(10K) machines; performance of such applications is very sensitive to communication tail effects [20].

On-NIC Resources. Traditionally, on-chip memories are avoided to the extent possible, given their area and power implications. However, with Falcon, we had to support packet buffering on the receive path for reordering purposes, which required O(BDP) worth of resources. To meet cluster-level requirements, the BDP for a 200Gbps IPU and 50us RTT setting (typical of intra-cluster RTTs) is 1.2MB. This grows to 2.4MB for 400Gbps and 4.8MB for 800Gbps. We find that with current process technologies (such as TSMC N7, N5 or N3) the packet buffer requires 1-2mm^2 of die area. This is <1% of the die area of an IPU SoC, making it feasible to realize. Crucially, sizing the packet buffer for a cluster-level setting translates to also supporting inter-cluster and inter-metro use-cases because the typical bandwidth required per-connection is significantly lower (by at least 5-10×) while the RTTs are correspondingly higher. For example, in case of AI/ML collectives, the demand reduces across network levels as communication is realized hierarchically. Finally, while the Falcon protocol has several mechanisms to ensure the on-NIC resources do not bottleneck performance (e.g., when connections experience network drops or host slow-downs), Falcon HW also allows packet buffers to overflow from on-chip SRAM to external on-NIC DRAM. This choice enables graceful behavior under extreme corner case conditions.

Connection State Caching. Like other connection-oriented hardware transports, Falcon employs a connection cache to keep the state of the active connections on chip. At warehouse scale, the active connection count far exceeds the cache size, resulting in a near 100% cache miss rate, which can cause a sharp degradation in throughput and latency. To limit the performance cliff in this regime, we provisioned enough bandwidth into the cache from the memory system that holds the connection state. In addition, a larger second-level cache (shared across multiple blocks in the IPU) was implemented to further improve the miss bandwidth and miss latency. Limiting this performance cliff comes at the cost of additional die area and memory bandwidth but this tradeoff is necessary to provide predictable sustained performance.

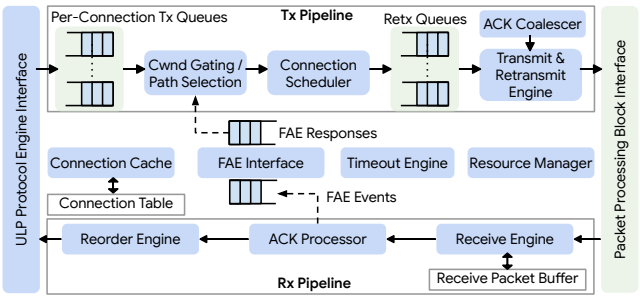


Figure 8: Falcon Block Diagram

Error Handling. Hardware transport designs typically choose to implement the fast path of the data plane in hardware, leaving the exception paths and error handling to be implemented by firmware. While this approach simplifies the hardware design, it can lead to a performance cliff caused by firmware packet processing blocking the hardware fast path due to packet ordering or data dependency requirements. At warehouse scale, errors and exceptions (triggered by packet drops, packet reordering, etc.) occur regularly and can often be bursty in nature, stalling the fast path for extended periods. To avoid this performance cliff, we implemented the error handling

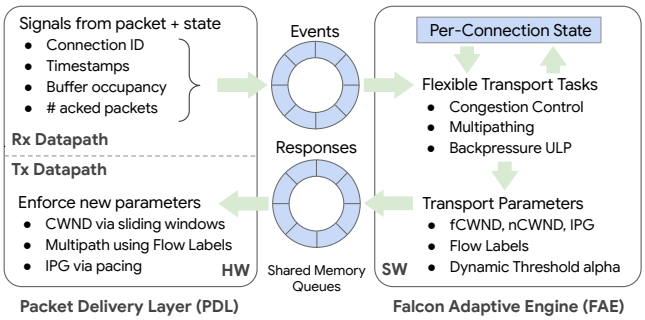


Figure 9: Falcon Adaptive Engine Implementation

in hardware. Loss recovery and error handling is implemented by maintaining per-packet contexts that store the necessary metadata (e.g. initial Tx timestamp) and retry states (e.g. RNR NACK). Retransmission timeouts are handled by efficient hardware threads that scan per-connection linked-lists of packet contexts.

5.3 Implementing and Scaling Falcon Adaptive Engine

In our IPU, FAE runs on general purpose on-NIC CPU cores; each core has private L1/L2 caches and a shared L3 cache. FAE communicates with the PDL via shared memory queues using well-defined *event-response* formats. PDL packs the relevant signals in the event, passes it to FAE, which computes various transport parameters and passes them back to PDL in the response (see Figure 9). Table 3 shows how Falcon implements various transport features by splitting the *mechanism* (implemented in hardware PDL) and the *management* (implemented in software FAE), along with the interface signals for specific features.

Apart from ensuring that FAE responds in a timely manner, it is also crucial to optimize its processing rate so that minimal cores are used by FAE. A key factor that influences this is how the FAE algorithm state is managed. Accessing per-connection algorithm state, on receiving an event, can incur cache misses, slowing down FAE. Cache misses occur as connection count increases which leads to the cumulative state exceeding cache capacity.

To avoid the state cache miss penalty, we initially designed FAE to be *stateless*, offloading state management to the PDL, which maintains the per-connection algorithm state and embeds it in the FAE event. This ensures that state is readily available along with the interface signals in the event for FAE algorithm consumption. Finally, FAE returns the state back to PDL in the response. However, stateless FAE is limited by the amount of per-connection algorithm state that can be managed by the PDL. To accommodate additional use-cases, we moved to *stateful* FAE, in which FAE manages additional state in software, which is retrieved while processing an event. To prevent cache miss overheads, prior to processing an event, we look further in the event queue and prefetch state for an upcoming event. This approach ensures that the later event does not incur the state retrieval penalty.

6 EVALUATION

We evaluate Falcon's reliability, congestion control, multipathing (§6.1), NIC hardware performance (§6.2), and its impact on MPI, HPC and Live Migration workloads (§6.3). We use a testbed of 32 machines equipped with 200 Gbps Falcon NICs and 400Gbps NVIDIA CX-7 NICs (with RoCE; RTTCC [6] used as the CC algorithm). When evaluating features present in higher-speed NICs

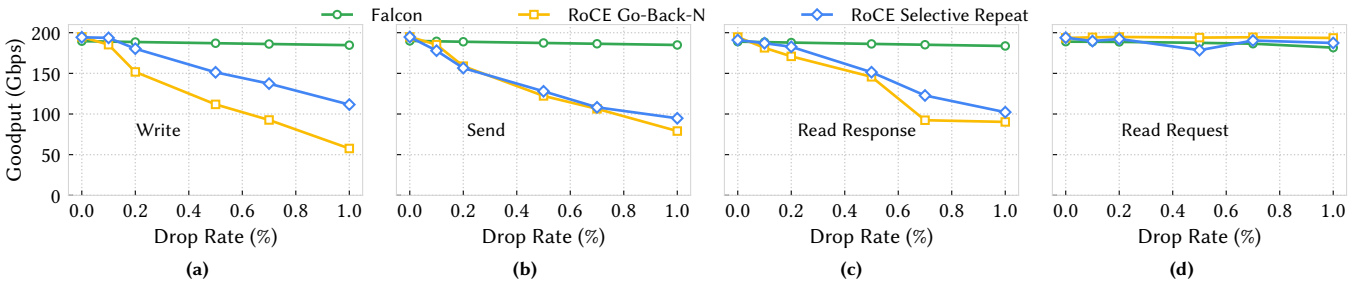


Figure 10: Falcon and RoCE goodput under losses for different ops (a) Write (b) Send (c) Read Responses and (d) Read Requests

such as RACK-TLP and multipathing (unavailable in our testbed), we use a simulator that has been validated against hardware emulation, ensuring that it accurately reflects the behavior of the actual hardware. The simulated topology is a 3-stage network like [48].

6.1 Performance of Falcon Protocol

We compare Falcon and RoCE using Reliable Connected (RC) Queue Pairs (QPs).

6.1.1 Loss Recovery

Impact of losses. To study the efficacy of reliability, we perform a point-to-point experiment wherein the first host initiates multiple 8KB RDMA ops (to ensure link is fully utilized) on a single QP and the switch between the two hosts is configured to randomly drop packets in the forward direction; we configure both Falcon and RoCE NICs to 200Gbps for a fair comparison. Figure 10 plots the achieved goodput as we vary the switch drop percentage for Write, Send, Read Response, and Read Request ops respectively.

Falcon is able to maintain stable goodput, even under losses, courtesy of its precise and timely loss recovery heuristic at the sender and ability to accept packets OOO at the receiver. In contrast, RoCE reliability schemes observe a drastic drop – up to 3.4× – in goodput under losses. As expected, RoCE's traditional Go-Back-N (GBN) achieves the lowest goodput due to it not accepting OOO packets and spuriously retransmitting all outstanding packets as a response to a single drop. We see RoCE's enhanced loss recovery scheme, known as Selective Repeat (SR), improves RoCE-GBN performance by 5%-95% in case of Writes and Read Responses. However, we see that when Sends and Read Requests are dropped, the performance of RoCE-SR and RoCE-GBN are same – revealing RoCE-SR is not available to these IB Verbs ops. Due to lack of on-NIC resources, RoCE is forced to drop incoming Sends and Read Requests. By contrast, Falcon uses on-NIC resources to enable accepting OOO ops while still meeting ordering requirements by reordering OOO arrivals. Moreover, even in scenarios where RoCE-SR makes an impact, Falcon outperforms RoCE by 5%-65%. This is due to slow RoCE-SR, while Falcon can recover multiple lost packets (as communicated by the bitmap in Falcon ACK) in an RTT. We note that dropping Read Requests doesn't lead to significant loss of goodput. This is because the bottleneck is the reverse direction and even if some requests are dropped, they are reliably retransmitted before they become head-of-line for processing.

Impact of reordering. A critical aspect of any loss recovery scheme is to distinguish between losses and reordering. To study Falcon and RoCE under packet reordering, we repeat the same 1:1 experiment as above with the switch reordering packets instead of dropping them. Figure 11a plots the goodput achieved by Write

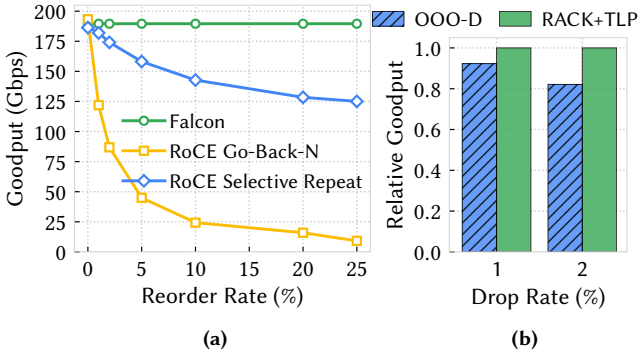


Figure 11: (a) Falcon and RoCE goodput when writes are reordered; (b) role of RACK-TLP under losses

ops as we vary the reordering extent. Similar to performance under loss, we see Falcon achieve stable goodput even under reordering while RoCE alternatives have lower goodput – 5%-52% lower in case of RoCE-SR and up to 20× lower with RoCE-GBN. Falcon's design choice of accepting OOO packets and including a reordering-tolerant factor in its loss recovery scheme enables it to distinguish between losses and reordering. By contrast, RoCE suffers due to it not being able to do the said distinction, leading to spurious retransmissions of reordered packets. RoCE-GBN goodput is worse than RoCE-SR because it spuriously retransmits the entire window and also cannot accept OOO packets.

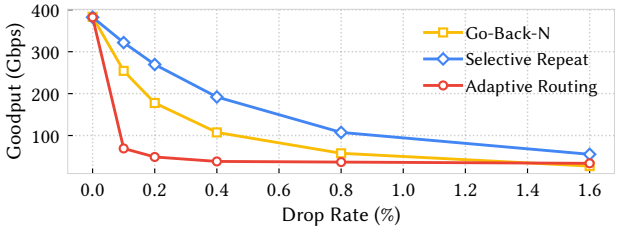
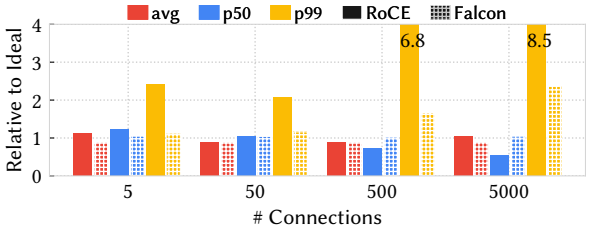
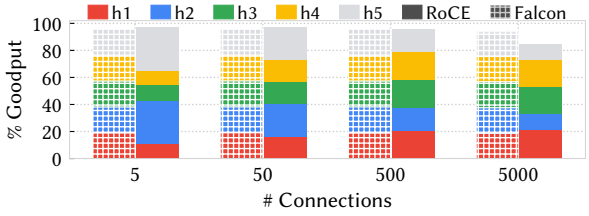


Figure 12: RoCE goodput under losses, in three different modes

Comparing different RoCE Modes. In addition to RoCE-SR and RoCE-GBN, we compare against Adaptive Routing mode (RoCE-AR) on 400G NICs, using the same loss recovery experiment described previously, but with 16KB RDMA Write ops for higher (~380G) goodput. As observed in Figure 12, RoCE-AR performs the slowest. Packet capture traces show no signal from the target for immediate retransmission (e.g. NACK packets), indicating a slower retransmission mechanism. This performance profile is presumably because



(a) Op latency relative to ideal with varying incast size



(b) Total goodput and fairness achieved under incast

Figure 13: Falcon and RoCE behavior under fabric congestion

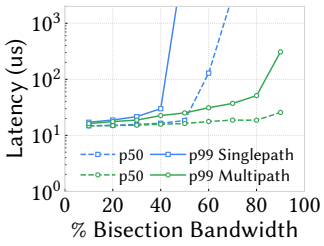


Figure 15: Multipath Latency

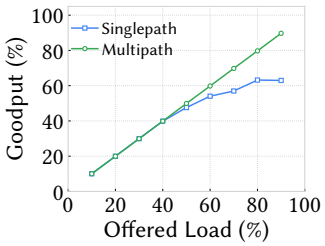


Figure 16: Multipath Goodput

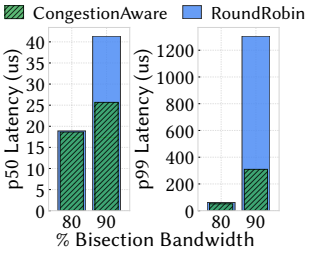


Figure 17: Multipath Policy

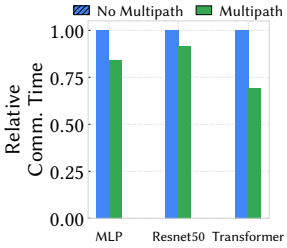


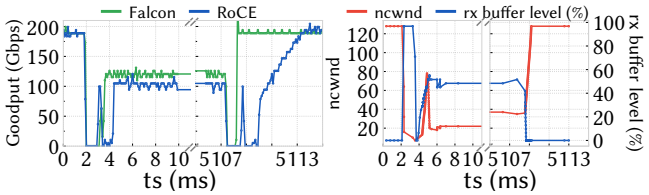
Figure 18: ASTRA-sim Training

AR mode is designed for lossless fabrics equipped with Adaptive Routing capability and not lossy fabrics.

Role of RACK-TLP. We simulate a sender generating 128KB Write ops with a Poisson arrival distribution, and the switch randomly dropping packets at varying rates. Figure 11b compares RACK-TLP and Out-of-Order Distance (OOO-D) heuristic (the initial loss recovery scheme in 200G Falcon) where the sender considers a packet eligible for retransmission if the distance between a non-received packet and a received higher packet sequence number (PSN) exceeds the OOO-threshold, similar to TCP FACK recovery [34]. OOO-D has 7-18% lower goodput than RACK-TLP. Under higher drop rates where tail losses increase, RACK-TLP uses tail loss probes for faster recovery, while OOO-D relies on retransmission timeouts.

6.1.2 Congestion Control

Fabric Congestion. We delve into Falcon's behavior in the presence of incasts where 5 client machines generate 1MB RDMA Writes (on a per-QP basis) to a single server; we vary QP count on each host to stress CC. Figure 13 shows that Falcon achieves close to optimal mean and median latency even with 5000:1 incast, and p99 is only ~2× from ideal – significantly better than RoCE. Within each host, the per-QP goodput distribution is tight, with <1% variance and the total goodput is close to line rate whereas RoCE loses 13% bandwidth during very large incasts.



(a) Goodput Timeline (b) Falcon Signals

Figure 14: Falcon and RoCE behavior under end-host congestion

End-Host Congestion. To demonstrate how Falcon deals with end-host congestion, we carry out a point-to-point experiment in which a client issues 64KB RDMA writes to a server, where we induce a slowdown by downgrading the PCIe link from 200Gbps to ~100Gbps. Figure 14a plots the goodput achieved over time. We can see the moment PCIe is downgraded, Falcon is able to quickly converge to a rate of 100 Gbps to match the bottleneck, while RoCE takes twice as long. Figure 14b shows that the *ncwnd* of Falcon adjusts itself to a lower value within ~5 RTTs. This is courtesy of Falcon ACKs carrying the RX buffer occupancy signal, which Falcon uses for ncwnd modulation and ensures the RX buffer occupancy converges to the desired target. We remove the PCIe downgrade at ~5108 ms, and Falcon quickly ramps up back to the original rate; RoCE takes 6× longer.

6.1.3 Multipath Load Balancing

To examine Falcon's native multipathing support, we run rack-level simulations in which 24 hosts in one-rack communicate with 24 hosts in another rack; host-1 in rack-1 communicates with host-1 in rack-2 and so on. We compare Falcon multipath performance against Falcon connections configured to use single-path.

Latency and Goodput. Figure 15 plots the observed op latency as we vary the offered load. We observe multipath-enabled connections deliver up to 180× lower latency compared to single-path connections. With multipathing, connections are able to use multiple paths in a congestion-aware manner. As shown in Figure 16, this enables sustaining offered loads of up to 90% of fabric capacity; in contrast, single-path connections only sustain loads up to 60%.

Scheduling Policy. Falcon's congestion-aware path selection policy assigns packets to the flow with the largest open congestion window, favoring less congested paths (§4.3). Figure 17 shows that this policy achieves 2-4× lower op latency compared to round-robin policy at high offered loads.

Benefit to ML Workloads. We use ASTRA-sim [3] to simulate ML workloads, to study the impact of multipathing. Figure 18 shows that multipathing, for a 64-node testbed across two racks, reduces communication time by up to 30% for larger models. The corresponding run time improves by up to 5%.

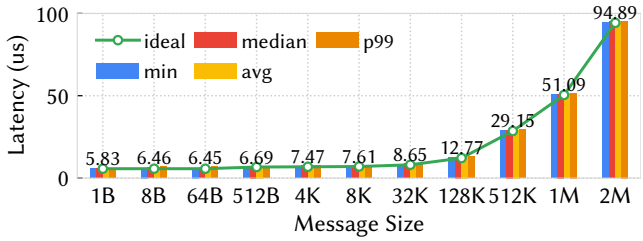


Figure 19: Message Size Scaling

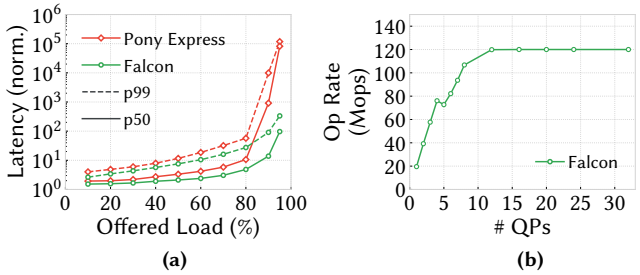


Figure 20: (a) BW Scaling (b) Op-rate Scaling

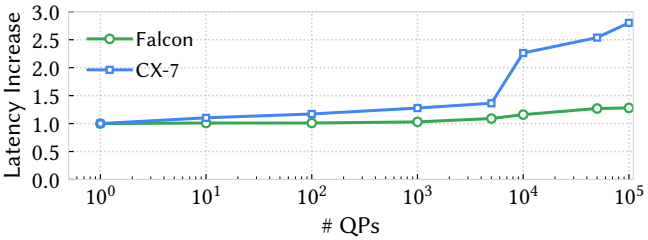


Figure 21: Connection Cliff

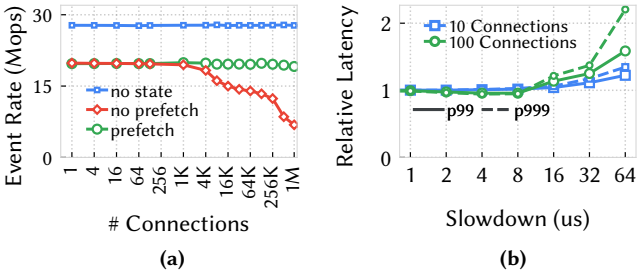


Figure 22: (a) FAE State Scaling (b) Impact of Slow FAE

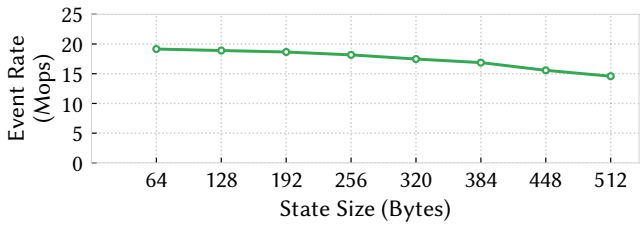


Figure 23: FAE State Sensitivity

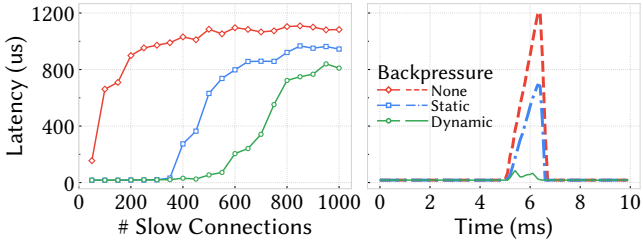


Figure 24: Isolation via fine-grained backpressure

6.2 Hardware Scalability

Message Size Scaling. We measure RDMA Write completion latency between two hosts in an unloaded network with varying message sizes. Figure 19 demonstrates that Falcon achieves tight tail latencies – p99 and median latencies within 1% of ideal.

Bandwidth Scaling. We perform a 500:1 RDMA Read incast where a single client requests data from 500 connections spread over five servers at varying bandwidths. Figure 20a shows that Falcon achieves two orders of magnitude lower latency than state-of-the-art SW transport when close to saturating the link bandwidth.

Op-Rate Scaling. Figure 20b shows the maximum op rate achieved between two hosts using 8B RDMA Writes: a single QP drives 20Mops and we hit 120Mops with as few as 12 QPs.

Connection Cliff. To understand the implications of the connection state caching design, we run a ping-pong (closed-loop, single-outstanding, 8B RDMA Reads) workload between two hosts, selecting connections randomly to exert cache pressure. Figure 21 plots the increase in the software-measured RTT as a function of #connections. RTT is highly sensitive to cache misses, as these misses are on the critical path due to interaction with on-NIC DRAM (in case of Falcon NIC) or host memory (in case of CX-7). NIC designs with ample concurrency can "hide" such misses when measuring throughput, but the latency profile is always revealing – unlike Falcon NICs, CX-7s incur a ~3× latency increase for high connection counts above 100K.

FAE Scaling. FAE's scalability depends on #connections and their associated state. Figure 22a shows the maximum event rate (using one Neoverse-N1 ARM core [8]) for varying connection counts with FAE variants (stateless, stateful, and stateful with prefetching). While a naive stateful implementation degrades event rate, prefetching maintains a stable 20 Mops. Prefetching reduces state fetch overhead in the critical path, especially at high connection counts where state spills out of L1/L2/L3 caches progressively.

To demonstrate the impact of a slow FAE, we limit its event processing rate by introducing delays in event turn around times. We run an incast experiment with two senders creating upto 100 QPs each, issuing 1MB RDMA Writes. Figure 22b plots the fabric RTT relative to an unrestricted FAE. Falcon's performance remains robust, with an impact of 1.5-2× increase in fabric delay due to delayed FAE responses only when added delays are beyond 32μs.

FAE State Sensitivity. We tested sensitivity to state size by measuring the random access event rate for 128K connections. Even with an eight-fold increase in state size from 64B, performance only dropped to 15M Events/s (from 20M Events/s) due to cache misses (Figure 23). Given that our real-world applications with Falcon (§6.3) peaked at 8M Events/s, FAE has substantial capacity to support larger state for future use cases.

Isolation. To demonstrate resource isolation (§4.6), we start two sets of flows from the same host: *fast* flows within the rack and *slow* flows across the rack, which are periodically slowed down by an incast. Slow flows can occupy most of Falcon's resources,

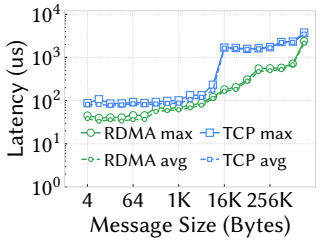


Figure 25: MPI AllReduce

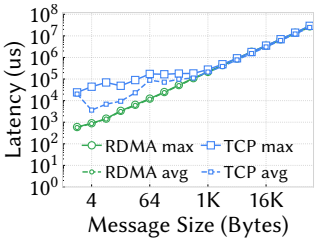


Figure 26: MPI AllToAll

Metric	NLF	SSD
Read Bandwidth	97.1%	100%
Write Bandwidth	91.0%	100%
IOPS (w/ 1MBps limit)	100%	100%

Table 4: Performance of NLF over Falcon and Local SSD

and impact fast flows. Figure 24 shows that fast flows experience a 63× slowdown (with 500 slow flows) without backpressure, improving to 37× under static backpressure, and to ~3× with dynamic backpressure.

6.3 Real-World Applications

MPI Collectives. We use Intel MPI Benchmarks for MPI collective performance evaluation. Figure 25 and 26 plot *AllReduce* and *AllToAll* collective completion time as a function of message size, for RDMA-Falcon and TCP (legacy MPI SW stack). We show 32-node scale with 192 processes per node, using all cores. The benefits of Falcon extend to both small and large operations. RDMA-Falcon improves AllToAll performance with small 4B messages by 4.3× relative to TCP, demonstrating the latency benefits of the hardware transport. Large, compute-bound ops also benefit from the offloaded Falcon transport; e.g., 64KB AllReduce is 5.5× faster with RDMA-Falcon. §B.1 shows the performance of other collectives.

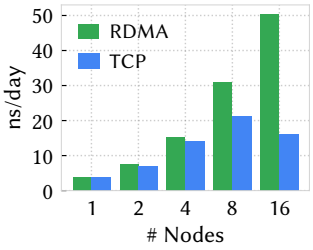


Figure 27: Gromacs

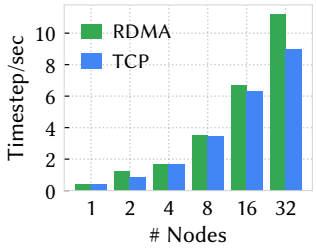


Figure 28: WRF

GROMACS and WRF. We evaluate GROMACS [4], a popular molecular dynamics simulation tool, and WRF [7], a weather forecasting model, over RDMA-Falcon and TCP, both using all 192 CPU cores per node. Figure 27 shows that with Falcon, simulation performance for the GROMACS *benchpep* workload scales to 16 nodes, while overall performance decreases above 8 nodes with TCP. RDMA-Falcon achieves 2.4× higher performance over the best-case TCP scale. Figure 28 shows WRF simulation rate for the *CONUS 2p5km* model. With Falcon, WRF scales to 32 nodes, with a 32% improvement in simulation time relative to TCP.

Near Local Flash (NLF). NLF uses NVMe-over-Falcon to disaggregate SSDs from compute nodes, making them accessible over a network as if they were locally attached. We evaluate NLF performance with storage access patterns using random reads (4KB to 16KB) and writes (1MB). NLF (using Falcon) incurs network overheads compared to local SSDs with DRAM caching. Table 4 shows that despite the network overheads, NLF's bandwidth and op rate are within 10% of local SSDs.

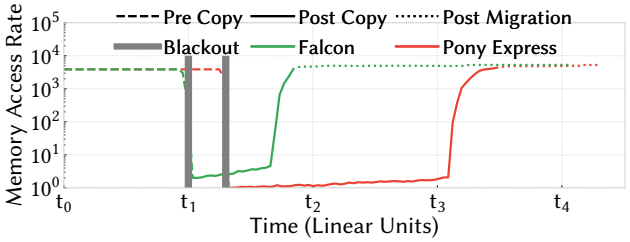


Figure 29: Live migration timeline with RDMA-Falcon and Pony Express

VM Live Migration. We compare Falcon with Pony Express for live migration of guest memory using a synthetic benchmark. To stress the migration stack, the guest VM continuously accesses and dirties its memory throughout the migration. Memory migration occurs in two phases [44]. In the pre-copy phase, the guest VM's memory is transferred to a target host while the VM remains active on the source. After a brief blackout period, the VM resumes on the target and enters the post-copy phase, fetching remaining memory in the background. If a guest's vCPU tries to access a page that is not yet fetched, that page is fetched on-demand, which can stall the corresponding vCPU until the requested page arrives.

As shown in Figure 29, Falcon significantly improves this process over Pony Express, accelerating the pre-copy and post-copy phases by 1.17× and 2.5×, respectively. Crucially, the faster post-copy yields a 3× higher guest access rate and a 6× reduction in vCPU wait time.

7 RELATED WORK

Hardware Transports. Amazon's *Scalable Reliable Datagram* (SRD) [46], designed for multi-tenant cloud environments, provides a libfabric interface for HPC workloads and NCCL for ML collectives. SRD employs congestion control similar to BBR [16] and uses multipathing for network load balancing. While SRD offloads ordering to the application, Falcon provides multipathing while maintaining IB Verbs ordering semantics. *TTPoE* [40], Tesla's RDMA hardware transport, uses on-NIC SRAM to improve performance and tolerate large RTTs. However, it is designed for single-tenancy and P2P transmission model, doesn't support IP networks, and lacks CC and multipathing support, which makes it unsuitable for large-scale, multi-tenant environments. *1RMA* [49] supports only a simplified non-verbs RMA API, onloads CC to the host SW, and relies on a central control plane to manage NIC resources. *ZeroNIC* [50] is a co-designed NIC that combines a zero-copy HW datapath and SW based control-path. This allows flexibility to implement different transport stacks, at the cost of reduced scalability and performance compared to Falcon.

While both *Ultra Ethernet Transport* (UET) from UEC [19] and Falcon target AI/ML and HPC networking, they differ in key areas, including Falcon's additional support for shared networks. For *application support*, Falcon maintains compatibility with existing applications using the IB Verbs standard, whereas UET is layered under libfabric, requiring non-libfabric applications to be adapted. To address *connection scalability*, Falcon employs micro-architectural

		RoCE	SRD	1RMA	Pony Express	EQDS/NDP	Falcon
Mechanism	**Cong. Ctrl (CC)**	DCQCN/RTTCC [6][1]	Delay-based	App-level	Swift[1]	Receiver-based	Swift[1]
	Multipathing	No support[3]	Yes	Yes[2]	Yes	Packet Spraying	Yes
	Programmability	Limited[4]	CC	No	SW	SW[5]	HW-assisted[6]
	Switch Support	PFC desired	Commodity Eth	Commodity Eth	Trimming	Commodity Eth	
	Loss Recovery	GBN/SR	Timeout-based	N/A[2]	SACK	Trimming	SACK/RACK-TLP
Capability	**Ordering**	Ordered	Unordered	Unordered	Unordered	Ordered	Ordered/Unordered
	Multi-ULP	IB Verbs	libfabric	Custom RDMA	Custom RMA	Yes	IBVerbs/NVMe/etc.
	Op Rate	High	High	High	SW-limited[7]	SW-limited	High

[1] Swift uses in-band hardware timestamps while RTTCC uses probe packets to measure delay.
[2] 1RMA's network Aquila [23] provided link-level reliability.
[3] Some RoCE deployments rely on switch-based adaptive routing.
[4] Some RoCE NICs support programming CC [9].
[5] EQDS [39] talks about a potential EQDS NIC. It is not clear how much programmability such NIC would have.
[6] Falcon's FAE provides programmability for CC, resource management, multi-pathing and ack modulation.
[7] Pony Express provides high CPU efficiency if compared to SW transports.

Table 5: An overview of transports capabilities and the mechanisms that enable them

solutions like on-NIC DRAM caching, while UET introduces the packet-delivery context, an ephemeral, on-demand connection state. For *security*, Falcon secures traffic on a per-connection basis, while UET offers either a group-based secure domain with shared decryption or a client-server model for private traffic. For *load balancing*, Falcon utilizes multipathing with a reorder buffer to maintain IB Verbs' strict ordering, but UET's packet spraying is limited to unordered traffic. Similar to Falcon, UET employs a SACK-based method for loss recovery.

Loss recovery. To mitigate RoCE limitations, IRN [36] introduced selective acknowledgments, and later a variant, selective repeat (SR), is adopted by RoCE NICs [5]. However, as we showed, SR's performance falls short compared to Falcon. SRNIC [52] attempts to mitigate the connection scalability problem, which is exacerbated by SR bitmaps, by carefully offloading only the frequently accessed state to the NIC.

Load-balancing. ConWeave[51] achieves network load balancing for RDMA by masking out-of-order delivery with programmable ToR switches. MP-RDMA [32] adds multi-pathing capability to RoCE but still requires PFC. However, neither fully overcomes the limitations of current RoCE.

Programmability. While Tonic [1] and HotCocoa [13] are programmable frameworks that enable custom CC algorithms, they cannot support Falcon's multipath CC, which requires multiple congestion windows and a path selection primitive for a single connection. NanoTransport [14] optimizes RPC-based applications by leveraging P4 programmable architecture and FPGAs. Similarly, nanoPU [26] uses a P4 pipeline and a RISC-V CPU scheduling codesign to reduce message latencies. However, the long-term viability of P4 NIC pipelines in production environments remains unproven.

Transport Protocols. Homa [37] and NDP [25] use receiver-driven packet scheduling, but require full-bisection bandwidth fabrics or rely on packet trimming. EQDS [39] provides transport functionality by tunneling different ULPs and while it envisions a hardware NIC implementation, it does not support standard IB Verbs. AccelTCP [38] accelerates short-lived flows by offloading the control path to the NIC. FlexTOE [47] is a TCP offload engine that achieves high performance by pipelining TCP processing. RoGUE [30] onloads CC to the CPU to support easier upgrades, potentially incurring heavy overhead during high congestion event rates. HPCC [31], PowerTCP [11], and Bolt [15] utilize In-band Network Telemetry signals and advanced switch capabilities to enhance CC, limiting their applicability in legacy networks. Falcon has no reliance on advanced switch capabilities but can potentially benefit from them due to its programmability. We summarize the related work in Table 5.

8 CONCLUSION

Falcon is a significant advancement in hardware transports: it is a converged NIC that works over standard Ethernet datacenter networks with packet losses, reordering and a diversity of paths, offering high performance, efficiency, and scalability for both specialized and general scale-out applications. Falcon is a departure from existing hardware transports through its ability to provide predictable performance and CPU efficiency without requiring complex network configurations or specialized switches, while maintaining compatibility with existing application interfaces. Its layered architecture, hardware-assisted programmable engine, native multipathing, hardware-based retransmissions and error-handling mechanisms collectively contribute to its high performance. Falcon's development embraced production experience from software transports, and as datacenter networks continue to grow in scale and complexity, Falcon's design rooted in adaptability positions it as a promising transport for existing and future applications.

Ethics. This work does not raise any ethical issues.

ACKNOWLEDGMENTS

We would like to thank Neal Cardwell, Thomas Wenisch, Arjun Singh, the anonymous SIGCOMM reviewers and our shepherd, Anirudh Sivaraman, for providing valuable feedback. Falcon is a multi-year effort at Google that benefited from an ecosystem of support and innovation. We thank the RDMA-Falcon hardware, production, and support teams at Google for their contributions to the work, including but not limited to, Jiazhen Zheng, Chandan Muddamsetty, Sean Clark, Kaiyu Shen, Alvin Wijaya, Weiwei Jiang, Gerald Schmidt, Rakesh Gautam, Ajay Venkatesan, Bala Jupudi, Matt Wycklendt, Julian Mentasti, Jessica Ramirez, Neil Van Lysel, Stefano Vazzoler, Tom Landon, Jacob Moroni, Akshat Thakkar, Huadong Liu, Anil Yelam, Gautam Kumar, Chen Zhao, Ashwin Murthy, Amrutha Sampath, Nic McDonald, Ahmad Ghalayini, Sarah Tollman, Tiancan Yu, Jiaxin Lin, Weida Huang and Jiayi Chen.

REFERENCES

[1] 2020. Enabling Programmable Transport Protocols in High-Speed NICs. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/nsdi20/presentation/arashloo

[2] 2021. PSP Security Protocol. https://github.com/google/psp. [Accessed 24-07-2025].

[3] 2024. AstraSIM Models. https://github.com/astra-sim/astra-sim/tree/ASTRA-sim-1.0/inputs/workload. [Accessed 30-01-2024].

[4] 2024. Gromacs: A free and open-source software suite for high-performance molecular dynamics and output analysis. https://www.gromacs.org/. [Accessed 30-01-2024].

[5] 2024. NVIDIA ConnectX-6 Dx Network Adapters — nvidia.com. https://www.nvidia.com/en-us/networking/ethernet/connectx-6-dx/. [Accessed 11-01-2024].

[6] 2024. Scaling Zero Touch RoCE Technology with Round Trip Time Congestion Control. https://developer.nvidia.com/blog/scaling-zero-touch-roce-technology-with-round-trip-time-congestion-control/. [Accessed 30-01-2024].

[7] 2024. Weather Research & Forecasting Model (WRF). https://www.mmm.ucar.edu/models/wrf/. [Accessed 30-01-2024].

[8] 2025. ARM Neoverse N1. https://www.arm.com/products/silicon-ip-cpu/neoverse/neoverse-n1. [Accessed 24-07-2025].

[9] 2025. DOCA PCC. https://docs.nvidia.com/doca/sdk/doca+pcc/index.html. [Accessed 28-01-2025].

[10] 2025. Spectrum-X. https://www.nvidia.com/en-us/networking/spectrumx/. [Accessed 24-07-2025].

[11] Vamsi Addanki, Oliver Michel, and Stefan Schmid. 2022. PowerTCP: Pushing the Performance Limits of Datacenter Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 51–70. https://www.usenix.org/conference/nsdi22/presentation/addanki

[12] Saksham Agarwal, Rachit Agarwal, Behnam Montazeri, Masoud Moshref, Khaled Elmeleegy, Luigi Rizzo, Marc Asher de Kruijf, Gautam Kumar, Sylvia Ratnasamy, David Culler, and Amin Vahdat. 2022. Understanding host interconnect congestion. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks* (Austin, Texas) *(HotNets '22)*. Association for Computing Machinery, New York, NY, USA, 198–204. doi:10.1145/3563766.3564110

[13] Mina Tahmasbi Arashloo, Monia Ghobadi, Jennifer Rexford, and David Walker. 2017. HotCocoa: Hardware Congestion Control Abstractions. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (Palo Alto, CA, USA) *(HotNets '17)*. Association for Computing Machinery, New York, NY, USA, 108–114. doi:10.1145/3152434.3152457

[14] Serhat Arslan, Stephen Ibanez, Alex Mallery, Changhoon Kim, and Nick McKeown. 2021. NanoTransport: A Low-Latency, Programmable Transport Layer for NICs. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)* (Virtual Event, USA) *(SOSR '21)*. Association for Computing Machinery, New York, NY, USA, 13–26. doi:10.1145/3482898.3483365

[15] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. 2023. Bolt: Sub-RTT Congestion Control for Ultra-Low Latency. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 219–236. https://www.usenix.org/conference/nsdi23/presentation/arslan

[16] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *Queue* 14, 5 (Oct. 2016), 20–53. doi:10.1145/3012426.3022184

[17] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. 2021. RFC 8985: The RACK-TLP Loss Detection Algorithm for TCP.

[18] Abhijit K. Choudhury and Ellen L. Hahne. 1998. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Trans. Netw.* 6, 2 (April 1998), 130–140. doi:10.1109/90.664262

[19] Ultra Ethernet Consortium. 2025. *Ultra Ethernet Specification v1.0*. https://www.ultraethernet.org/wp-content/uploads/sites/20/2025/06/UE-Specification-6.11.25.pdf

[20] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (Feb. 2013), 74–80. doi:10.1145/2408776.2408794

[21] Denis Foley and John Danskin. 2017. Ultra-performance Pascal GPU and NVLink interconnect. *IEEE Micro* 37, 2 (2017), 7–17.

[22] Peter X. Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. pHost: Distributed Near-optimal Datacenter Transport over Commodity Network Fabric. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies* (Heidelberg, Germany) *(CoNEXT '15)*. ACM, New York, NY, USA, Article 1, 12 pages. doi:10.1145/2716281.2836086

[23] Dan Gibson, Hema Hariharan, Eric Lance, Moray McLaren, Behnam Montazeri, Arjun Singh, Stephen Wang, Hassan M. G. Wassel, Zhehua Wu, Sunghwan Yoo, Raghuraman Balasubramanian, Prashant Chandra, Michael Cutforth, Peter Cuy, David Decotigny, Rakesh Gautam, Alex Iriza, Milo M. K. Martin, Rick Roy, Zuowei Shen, Ming Tan, Ye Tang, Monica Wong-Chan, Joe Zbiciak, and Amin Vahdat. 2022. Aquila: A unified, low-latency fabric for datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1249–1266. https://www.usenix.org/conference/nsdi22/presentation/gibson

[24] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 202–215. doi:10.1145/2934872.2934908

[25] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichik, and Marcin Mojcik. 2017. Re-architecting Datacenter Networks and Stacks for Low Latency and High Performance. In *Proceedings of the ACM SIGCOMM 2017 Conference* (Los Angeles, CA) *(SIGCOMM '17)*. ACM, New York, NY, USA, 29–42.

[26] Stephen Ibanez, Alex Mallery, Serhat Arslan, Theo Jepsen, Muhammad Shahbaz, Changhoon Kim, and Nick McKeown. 2021. The nanoPU: A Nanosecond Network Stack for Datacenters. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 239–256. https://www.usenix.org/conference/osdi21/presentation/ibanez

[27] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. 2023. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–14.

[28] Stephen Kent and Ran Atkinson. 1998. IP Encapsulating Security Payload (ESP). RFC 2406. doi:10.17487/RFC2406

[29] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 514–528. doi:10.1145/3387514.3406591

[30] Yanfang Le, Brent Stephens, Arjun Singhvi, Aditya Akella, and Michael M. Swift. 2018. RoGUE: RDMA over Generic Unconverged Ethernet. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. Association for Computing Machinery, New York, NY, USA, 225–236. doi:10.1145/3267809.3267826

[31] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and et al. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 44–58. doi:10.1145/3341302.3342085

[32] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. 2018. Multi-Path Transport for RDMA in Datacenters. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 357–371. https://www.usenix.org/conference/nsdi18/presentation/lu

[33] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, and et al. 2019. Snap: A Microkernel Approach to Host Networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 399–413. doi:10.1145/3341301.3359657

[34] Matthew Mathis and Jamshid Mahdavi. 1996. Forward acknowledgement: refining TCP congestion control. In *Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Palo Alto, California, USA) *(SIGCOMM '96)*. Association for Computing Machinery, New York, NY, USA, 281–291. doi:10.1145/248156.248181

[35] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. 1996. *TCP Selective Acknowledgment Options*. RFC 2018. RFC Editor.

[36] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting Network Support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 313–326. doi:10.1145/3230543.3230557

[37] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A Receiver-driven Low-latency Transport Protocol Using Network Priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. ACM, New York, NY, USA, 221–235. doi:10.1145/3230543.3230564

[38] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. 2020. AccelTCP: Accelerating Network Applications with Stateful TCP Offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 77–92. https://www.usenix.org/conference/nsdi20/presentation/moon

[39] Vladimir Olteanu, Haggai Eran, Dragos Dumitrescu, Adrian Popa, Cristi Baciu, Mark Silberstein, Georgios Nikolaidis, Mark Handley, and Costin Raiciu. 2022. An edge-queued datagram service for all datacenter traffic. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 761–777. https://www.usenix.org/conference/nsdi22/presentation/olteanu

[40] Eric Quinnell. 2024. Tesla Transport Protocol Over Ethernet (TTPoE): A New Lossy, Exa-Scale Fabric for the Dojo AI Supercomputer. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. 1–23. doi:10.1109/HCS61935.2024.10664947

[41] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: Congestion Signals Are Simple and Effective for Network Load Balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 207–218. doi:10.1145/3544216.3544226

[42] Abhiram Ravi, Nandita Dukkipati, Naoshad Mehta, and Jai Kumar. 2024. *Congestion Signaling (CSIG)*. Internet-Draft draft-ravi-ippm-csig-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ravi-ippm-csig/01/ Work in Progress.

[43] Renato Recio, Bernard Metzler, Paul Culley, Jeff Hilland, and Dave Garcia. 2007. A Remote Direct Memory Access Protocol Specification. RFC 5040.

[44] Adam Ruprecht, Danny Jones, Dmitry Shiraev, Greg Harmon, Maya Spivak, Michael Krebs, Miche Baker-Harvey, and Tyler Sanderson. 2018. VM Live Migration At Scale. *SIGPLAN Not.* 53, 3 (March 2018), 45–56. doi:10.1145/3296975.3186415

[45] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable Traffic Shaping at End Hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 404–417. doi:10.1145/3098822.3098852

[46] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. 2020. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro* 40, 6 (2020), 67–73. doi:10.1109/MM.2020.3016891

[47] Rajath Shashidhara, Tim Stamler, Antoine Kaufmann, and Simon Peter. 2022. FlexTOE: Flexible TCP Offload with Fine-Grained Parallelism. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 87–102. https://www.usenix.org/conference/nsdi22/presentation/shashidhara

[48] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, and et al. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 183–197. doi:10.1145/2829988.2787508

[49] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 2020. 1RMA: Re-Envisioning Remote Memory Access for Multi-Tenant Datacenters. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 708–721. doi:10.1145/3387514.3405897

[50] Athinagoras Skiadopoulos, Zhiqiang Xie, Mark Zhao, Qizhe Cai, Saksham Agarwal, Jacob Adelmann, David Ahern, Carlo Contavalli, Michael Goldflam, Vitaly Mayatskikh, Raghu Raja, Daniel Walton, Rachit Agarwal, Shrijeet Mukherjee, and Christos Kozyrakis. 2024. High-throughput and Flexible Host Networking for Accelerated Computing. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 405–423. https://www.usenix.org/conference/osdi24/presentation/skiadopoulos

[51] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. 2023. Network Load Balancing with In-Network Reordering Support for RDMA. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 816–831. doi:10.1145/3603269.3604849

[52] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, Tianhao Wang, Weicheng Ling, Kejia Huo, Pingbo An, Kui Ji, Shideng Zhang, Bin Xu, Ruiqing Feng, Tao Ding, Kai Chen, and Chuanxiong Guo. 2023. SRNIC: A Scalable Architecture for RDMA NICs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1–14. https://www.usenix.org/conference/nsdi23/presentation/wang-zilong

[53] David Wetherall, Abdul Kabbani, Van Jacobson, Jim Winget, Yuchung Cheng, Charles B. Morrey III, Uma Moravapalle, Phillipa Gill, Steven Knight, and Amin Vahdat. 2023. Improving Network Availability with Protective ReRoute. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 684–695. doi:10.1145/3603269.3604867

APPENDICES

Appendices are supporting material that has not been peer-reviewed.

A ADDITIONAL FALCON DESIGN DETAILS

A.1 Separate Request-Response PSN Space

Avoiding request-response deadlocks in protocol implementations with finite resources is a critical requirement for Falcon and existing protocols such as RoCE and iWARP.

To avoid request-response deadlocks, Pull Requests and Pull Responses flowing in the same direction need to be handled independently. Falcon achieves this by using two separate sequence number spaces for Pull Requests and Pull Responses in **each** direction.

In contrast, RoCE uses the requester's sequence number space for the responses flowing in the other direction. In doing so, requests and responses in the same direction use different sequence number spaces, resolving the request-response deadlocks. However, sharing the sequence number space between different directions, i.e., requester and responder, has other negative side effects specifically in loss recovery – e.g., even if a packet is lost from responder to requestor, the entire request needs to be retransmitted etc.

On the other hand, iWARP avoids these deadlocks by assuming that the target RDMA has enough resources which is impractical for a hardware protocol implementation.

A.2 Role of PSN and RSN

To meet the in-order reliable delivery requirements, each Falcon packet carries PSN for reliability and RSN for transaction ordering.

Unlike alternatives such as RoCE, wherein PSN ordering implies the required ordering between Pull Responses and Push Completions by virtue of using the requester's sequence number space for the responses flowing in the other direction, Falcon uses different PSN sequence number spaces for requests and responses to avoid deadlocks as discussed above (§A.1).

Thus, given separate PSN spaces for requests and responses, PSNs cannot be used to ensure ordered delivery. To do so, Falcon introduces RSNs that are used to provide ordering. Specifically, initiator Falcon uses RSNs to ensure that the responses are delivered in-order to the ULP, e.g., delivering Pull Responses and Push Completions in order to the initiator ULP. Likewise target Falcon uses RSNs to deliver transactions in order to the target ULP.

B ADDITIONAL EVALUATION

B.1 Other MPI Collectives

Continuing from §6.3, Figures 30 and 31 plot the Intel MPI Benchmark's *AllGather* and *MultiPingPong* collective completion time as a function of message size, for RDMA-Falcon and TCP. We show 32-node scale with 192 processes per node, using all cores. The benefits of Falcon extend to both small and large operations, but larger gains are seen for small operations when the transport is not bandwidth bound.

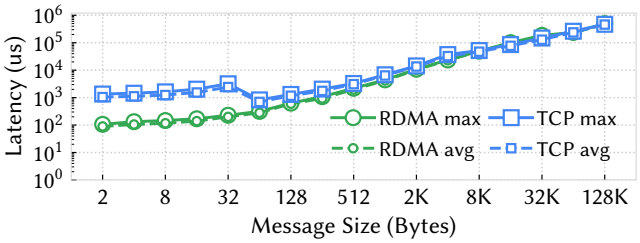


Figure 30: MPI allgather performance with 8 nodes, 192 processes per node

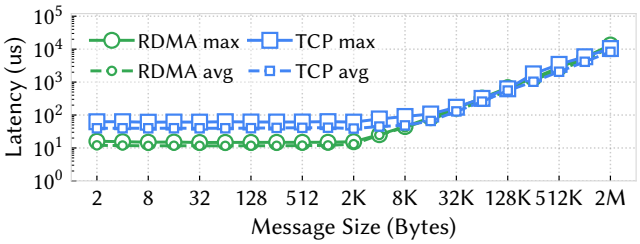


Figure 31: MPI multi-pingpong performance with 2 nodes, 192 processes per node