STORM: Enabling Traffic Scheduling for RDMA

Jichun Wu
jichun.wu@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Ran Shu
ran.shu@microsoft.com
Microsoft Research
Beijing, China

Gianni Antichi
gianni.antichi@polimi.it
Politecnico di Milano
Milan, Italy
Queen Mary University of London
London, UK

Yongqiang Xiong
yqx@microsoft.com
Microsoft Research
Beijing, China

Jon Crowcroft
jon.crowcroft@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Andrew W. Moore
andrew.moore@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Abstract

Remote Direct Memory Access (RDMA) is increasingly used as a shared communication substrate across datacenter workloads with very different scheduling needs, from request-response services and storage fan-out to AI training collectives. Proper request scheduling can reduce communication time, but in practice, no RDMA flow scheduling is enabled in datacenters, leaving traffic to simple fair sharing. We present STORM, a NIC-level scheduler for all types of RDMA workloads using NIC-only information: the known RDMA request size, and per-queue-pair backlog. STORM converts these signals into a small number of extra priority levels on the wire and prioritizes requests that are either near completion or blocking queued dependent work. STORM requires no application hints and works with both in-order RoCEv2 and newer RDMA stacks that tolerate reordering. We prototype STORM on an FPGA NIC with negligible overhead. Across representative cloud and LLM training workloads, STORM reduces training iteration time by up to 12% and reduces average and P99 flow completion slowdown by up to 90% compared to fair scheduling.

CCS Concepts

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

Keywords

RDMA, Flow Scheduling

ACM Reference Format:
Jichun Wu, Ran Shu, Gianni Antichi, Yongqiang Xiong, Jon Crowcroft, and Andrew W. Moore. 2026. STORM: Enabling Traffic Scheduling for RDMA. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3789240.3829117

1 Introduction

Remote Direct Memory Access (RDMA) has become a default building block for datacenter communication because it delivers microsecond-scale latency and high throughput while largely removing the CPU from the fast path. It supports not only traditional high-performance computing (HPC) workloads [6, 57, 64], but also production services at scale, including key-value systems [24, 28, 52, 63], cloud storage [4], and distributed AI training [15, 19, 23, 48]. This shift matters because deployments increasingly standardize on the same RDMA NICs (RNICs) and accelerators across clusters. The RDMA fabric is therefore expected to deliver consistently high performance across distinct application traffic patterns.

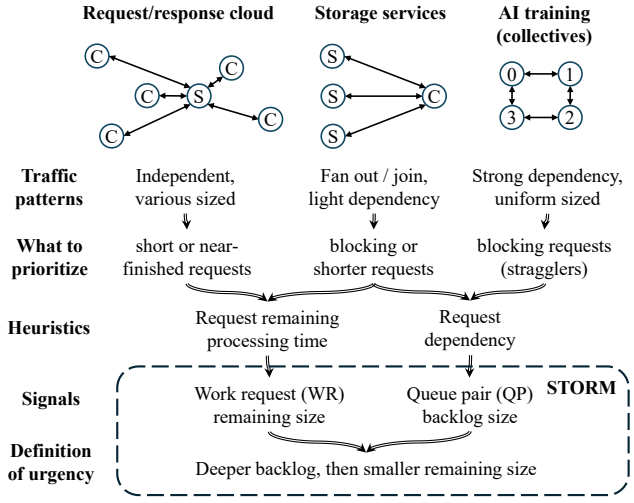


Figure 1: RDMA workload spectrum and the resulting scheduling objective.

This expectation makes RDMA request scheduling a first-order problem. RDMA application performance is strongly affected by how requests are scheduled into limited network resources. Once the host posts work requests, the RDMA Network Interface Card (RNIC) decides when packets are emitted and which priority class they are tagged with, and the fabric resolves contention accordingly. As a result, changing this arbitration and tagging policy can directly change requests' queueing delay in NIC and network. With an appropriate scheduling policy that matches workload urgency, the RNIC can reduce queueing delay under contention and improve application performance. However, in production, a service is often allocated a single priority class, and all traffic within that class is left

SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3829117

to naive arbitration that approximates fair sharing. This default is attractive operationally for its simplicity, but it leaves little room to differentiate urgency within the service. If a service can be allocated more than one priority level, it becomes possible to prioritize its own traffic under contention instead of relying on fair sharing. Doing so requires a clear definition of urgency: when multiple RDMA requests contend for the same links and queues, which ones should be served first?

The right definition for urgency is workload-dependent as RDMA workloads span a spectrum of dependency structures that directly change what the network should prioritize, as illustrated in Figure 1. At the low-dependency end are request-response services such as key-value databases, where operations are largely independent and performance is defined by per-operation latency. Storage sits in the middle: a single I/O expands into multiple rounds of coordination and data movement, with small control messages orchestrating subsequent transfers and completions; Azure's disaggregated storage stack exemplifies this structure, where frontend requests trigger coordinated transfers to multiple storage nodes and replica acknowledgments before completion [4]. At the high-dependency end is AI training, where communication is staged by an explicit dependency graph and iteration time is dictated by stragglers rather than average per-message completion. A practical RDMA scheduler must therefore support the whole spectrum of workloads.

Existing work does not provide this capability. Classic datacenter schedulers largely target TCP/IP and approximate size-aware ordering, often as an approximation of Shortest-Remaining-Processing-Time (SRPT) scheduling, inferring size signals from attained service, bytes sent, sampling, or aging [5, 8, 29, 59]. Even with perfect size knowledge like pHost [16] and Homa [37], they do not capture whether the head-of-line (HoL) work is blocking additional queued work, which is exactly what matters under dependency-driven workloads. For collective-heavy AI training, most techniques instead reshape the workload above the network by changing parallelization patterns or interleaving different jobs [7, 23, 38, 39, 43, 44, 48, 53]. Coflow schedulers similarly operate above the transport and exploit framework-visible structure to schedule communication [11–13, 27, 41, 60, 66]. These methods treat the network fabric as best effort once communication is issued, leaving the key remaining lever unchanged: the RNIC and fabric still arbitrate contending RDMA requests using naive fair scheduling.

To address this gap, we propose STORM, a NIC-resident RDMA request scheduler that turns RNIC-visible metadata into an enforceable notion of urgency. RDMA exposes two signals at exactly the point where arbitration is enforced: each work request (WR) has an explicit length, and WRs queue inside the RNIC on a per-queue-pair (per-QP) basis. STORM leverages these signals, exact request size and per-QP backlog, and assigns priorities directly at the point where packets are arbitrated, without prediction, approximation, or application annotations.

The mechanism starts from a principled baseline: SRPT is the canonical policy for minimizing flow completion time (FCT) in systems with variable transfer sizes. Prioritizing smaller remaining transfers is well known to reduce queueing under bursty contention and improves both mean and tail completion time [1, 3, 5, 8, 16, 20, 29, 32, 37, 42, 65]. Its limitation is equally well understood: SRPT optimizes per-transfer completion and is agnostic to higher-level dependency structure. In dependency-heavy executions, a transfer that is small or near completion is not necessarily the one that unblocks global progress; worse, aggressively prioritizing them can repeatedly preempt the transfers that lie on the critical path, increasing completion time even if many individual transfers finish quickly.

STORM resolves this mismatch by augmenting size-based urgency with backlog-based urgency. A deeper QP backlog indicates that the head-of-line request is gating more ready work behind it, so accelerating it can unlock a larger amount of pending progress under contention. This converts prioritization from "finish the smallest" to "finish the request that blocks the most queued work," which is the right primitive for dependency chains. Operationally, STORM combines remaining size and backlog and maps requests into a small number of priority classes. It groups fragmented WR postings so that both remaining size and backlog reflect *logical* transactions rather than artifacts of how libraries slice a transfer.

In summary, this paper makes the following contributions:

- We identify RDMA-specific signals: request size and QP backlog size, which enable practical, NIC-friendly scheduling without application changes. (§2)
- We present STORM, to the best of our knowledge, the first NIC-resident RDMA traffic scheduler. STORM is deployable using existing priority primitives and is transport-agnostic, supporting both in-order RoCEv2 and reordering-tolerant RDMA stacks. (§3)
- We build an FPGA prototype of STORM and demonstrate that it incurs minimal hardware overhead and no throughput penalty on the fast path. (§4)
- We evaluate STORM end-to-end both in large-scale simulation and on a small testbed, showing reduced average and tail completion times for cloud workloads and faster LLM training iterations. (§5)

2 Background and Motivation

2.1 Scheduling gap in RDMA

Commodity RDMA deployments expose a small set of discrete network priorities (e.g., via PCP or DSCP), but operators typically reserve them for coarse service separation. Within a single service or job, traffic is therefore commonly left to simple, naive scheduling that approximates fair sharing across active QPs. This is often a poor fit for RDMA. Contention is ultimately resolved by the priority decisions made at the RNIC and enforced by fabric queues, and RDMA exposes native, RNIC-visible signals that make finer-grained scheduling both feasible and enforceable within an existing priority budget. We illustrate this gap using three minimal scenarios spanning representative RDMA workloads.

Scenario 1, independent cloud requests: size alone is decisive. Consider a random-arrival cloud setting where requests are largely independent and completion time is dominated by per-request latency. If a small transfer overlaps with a large transfer at a bottleneck and both are served by naive fair sharing, the small transfer can easily see a 2× completion-time inflation, while the large transfer gains essentially nothing. This is the classic rationale behind SRPT-like scheduling: prioritizing the smaller request reduces mean and tail completion time because it drains quickly, frees capacity, and does not materially delay the large transfer. In

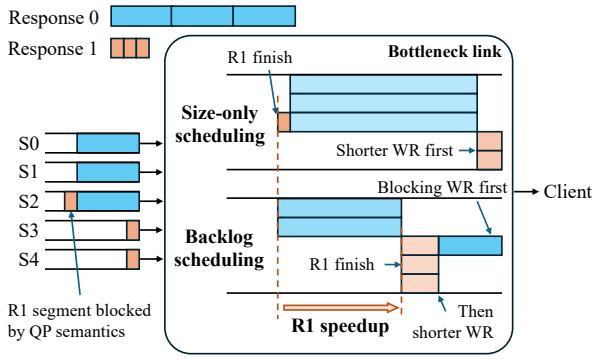


Figure 2: Storage fan-out/join at a shared bottleneck: size-only prioritization fails under RDMA QP head-of-line blocking at the gatekeeper (S2), while backlog-aware unblock-first scheduling drains the blocking WR and accelerates completion of the dependent reply.

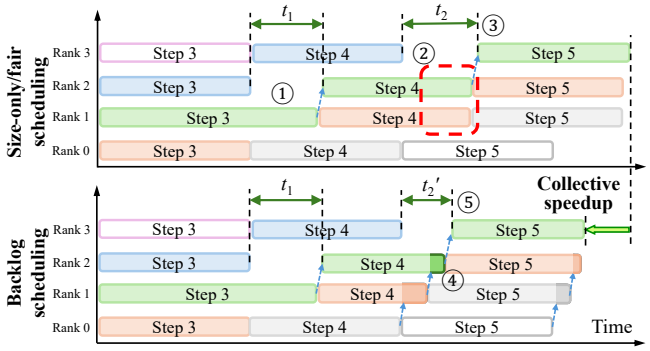


Figure 3: Ring-based collective: a straggling step creates a bubble (①) that propagates under cross-step contention ($t_2 \approx t_1$, ②–③); backlog accumulation promotes the head-of-line earlier-step WR (④), shrinking the bubble ($t_2' < t_2$) and speeding up the collective (⑤).

this regime, the only information needed is remaining size, and RDMA provides it exactly at the point of arbitration via explicit WR lengths.

Scenario 2, storage workloads: size is insufficient, blocking dominates. Storage I/O commonly follows a fan-out then join pattern: a client operation triggers multiple backend responses, and completion is gated by the last missing fragment rather than by average throughput. Figure 2 illustrates a minimal instance with five backend servers S0-S4 replying to the same client across a shared bottleneck link. Two client operations overlap, where R0 requires replies from S0, S1, S2 and R1 requires replies from S2, S3, S4. Server S2 is the gatekeeper because it must send both replies, so its QP contains [R0, R1]. Under RDMA QP semantics, S2 cannot transmit the R1 reply until the head-of-line R0 reply is drained, even if R1 is smaller. As a result, size-aware scheduling achieves essentially the same R1 completion as naive fair sharing across servers in this instance: both policies are bottlenecked by the same head-of-line constraint at S2, so prioritizing short replies elsewhere cannot deliver the missing R1 fragment any earlier as the network spends service on non-gating traffic. A backlog-aware policy resolves this by prioritizing the gatekeeper's head-of-line reply first, draining the only QP with backlog, and then allowing the short R1 reply to proceed. This "unblock-first" behavior targets the true critical path in lightly dependent storage pipelines, without requiring application hints or global coordination.

Scenario 3, collectives: backlog reveals the critical-path straggler. Figure 3 shows how a transient imbalance in a collective propagates across iterations when different steps overlap in the network. One rank is slowed (e.g., by asymmetric compute, GPU load, or link bandwidth) (①), creating a bubble before the next communication step. When faster ranks advance, traffic from a later step overlaps with the straggling traffic from an earlier step at shared bottlenecks, so an earlier-step transfer on the critical path contends with later-step communication (②). Under naive scheduling, the bottleneck allocates bandwidth without regard to step urgency, so the straggler receives only its fair share rather than the extra share needed to catch up, and the bubble persists with $t_2 \approx t_1$ (③). An omniscient solution would tag each message with its step index and prioritize earlier steps, but this requires upper-layer coordination and a global mapping of potentially unlimited steps into a handful of priority classes. Backlog provides an implementable alternative. Once later-step work becomes ready while the rank is still sending an earlier step, the corresponding WRs queue behind the head-of-line WR in the same QP, increasing backlog. STORM treats this backlog as a local critical-path signal and promotes the head-of-line earlier-step WR (④), allowing it to cut ahead of later-step traffic at the bottleneck. This targeted boost lets the straggler drain the earlier step faster and shrink the bubble from t_2 to t_2' (⑤), improving collective completion time without step tags or global scheduling state.

2.2 RNIC-observable scheduling signals

RDMA exposes the two scheduling signals STORM relies on as first-class RNIC metadata: exact message size and per-QP backlog. Message size is explicit at posting time because every send work request specifies its payload through a scatter-gather list whose entries carry an explicit byte length (e.g., `ibv_sge.length`), and one-sided READ/WRITE similarly specify the transfer length as part of the posted request [40, 50]. Two-sided SEND/RECV preserves the same property: the sender declares the message length when posting, and the receiver posts receive buffers with explicit bounds, so the RNIC can track remaining bytes directly from posted lengths and observed progress without any size inference [40, 50].

Backlog is equally explicit because RDMA work submission is mediated by doorbells and drained by completions. Applications enqueue WQEs in the QP work queue and ring a doorbell that advances the producer state visible to the RNIC; the RNIC then fetches and executes queued WQEs asynchronously and reports completion via CQEs, making outstanding work a well-defined queueing quantity at the RNIC boundary [40]. Doorbells are implemented as lightweight MMIO notifications and are a known RNIC-side resource, which makes them a natural low-latency event hook for tracking queue depth without relying on packet-level observation [45].

The only subtlety is that application libraries may fragment one logical transfer into many WQEs, which would artificially inflate

backlog size and deflate sizes if treated naively. STORM therefore defines backlog over *logical transactions* rather than raw WQEs, and later introduces a lightweight grouping mechanism to recover this logical unit so that size and backlog retain their intended scheduling semantics (§3.5).

2.3 Commodity RDMA constraints

Commodity RoCE deployments impose a tight coupling between what can be *measured* and what can be *enforced*: RNICs expose rich, continuous RDMA-native signals (e.g., WR sizes and per-QP backlog), yet the fabric typically offers only a small number of discrete priority classes (e.g., the eight 802.1Q PCP values) that map to switch egress queues and RNIC priority tagging. This section summarizes the resulting deployment model and the core design constraints it induces for any practical size-/urgency-aware scheduler.

Limited, discrete priorities are the only ubiquitous scheduling primitive. In production networks, the priority budget is typically already consumed by coarse service-level QoS using standard policies such as strict priority and weighted round-robin (WRR); common switches support combining them, for example, one strict-priority queue for latency-critical traffic, with WRR among the rest. A deployable mechanism therefore cannot assume control over all priorities, and must coexist with the existing QoS policy rather than replace it. As a result, the most viable design point is a minimal refinement that consumes only one or two additional priority levels, effectively splitting an existing class into a small number of priorities while preserving the original QoS policy.

Scheduling must run at datapath timescales, hence it must be RNIC-resident and hardware-friendly. At 100 Gbps and beyond, packet timescales are sub-microsecond; for example, a RoCE packet takes at most ≈360 ns on the wire (4000B at 100Gbps), and host-side/PCIe interactions are also on the order of hundreds of nanoseconds to microseconds [21]. This rules out closed-loop designs that repeatedly export fine-grained state to software and feed back per-packet decisions. Scheduling must be enforced where packets are emitted and queued: in the RNIC datapath and via the fabric's discrete priority queues. This constraint is not academic: commodity RNICs already implement latency-critical control loops such as DCQCN [68] and HPCC [30] in hardware, and Google [56] demonstrates that hardware NIC-resident mechanisms are the only reliable way to sustain high operation rate while controlling tail latency. Conversely, switch-side schedulers are limited by pipeline and queueing primitives in merchant silicon [9, 51, 54]. As a result, deployable scheduling logic must be simple and bounded, relying on operations such as lightweight grouping, counters, and comparisons that can be implemented at line rate.

Sender-local immediacy versus receiver-global visibility. Where decisions are made determines what information is available in time to matter. A sender has immediate access to its own exact WR sizes and per-QP backlog, and can tag packets with a priority with zero feedback delay; however, it cannot see competing senders converging on the same receiver or downstream bottleneck. This limitation is visible in incast: each sender may have only one ready request in its local queue, so no sender can tell that its request is part of a larger fan-in event. If all senders independently transmit based on this local view, the receiver-facing link and queues can become the bottleneck. A receiver, in contrast, can observe the aggregate fan-in after traffic arrives and coordinate which senders may continue transmitting. This visibility comes with at least one RTT of feedback delay, which is significant at RDMA's high data rates. Consequently, receiver coordination is naturally suited for RTT-scale arbitration under aggregate contention, while sub-RTT scheduling for short transfers must rely on sender-local signals. We later validate this distinction with STORM-soft, a software-only variant that performs sender-local priority marking at WR posting time but removes receiver-side active-set control (§5).

Ordering and loss semantics vary across fabrics, but priority-based scheduling should not. RDMA traditionally requires in-order, lossless packet delivery, but modern stacks increasingly tolerate packet reordering and drops. Practically, RDMA fabrics span both lossless configurations (e.g., RoCEv2 with PFC) and lossy configurations enabled by transports that reduce sensitivity to loss and reordering, including IRN and selective-repeat variants, as well as redesigned hardware transports such as 1RMA, SRD, Falcon, and UEC [33, 36, 49, 55, 56, 62]. STORM is intentionally orthogonal to this diversity: it operates by assigning priorities (compatible with existing RNIC tagging and switch egress queues), and it does not assume or modify a particular reliability, loss, or congestion-control mechanism.

3 Design

3.1 Overview

STORM schedules RDMA messages by mapping an urgency signal into a small number of network priority classes. The urgency signal is derived from two RDMA-native inputs: remaining message size (for SRPT-style efficiency) and per-QP backlog depth (for blocking-aware criticality). STORM is designed to operate under a constrained priority budget: the default configuration uses three priorities (P1–P3) (lower index denotes higher priority), it remains functional with two, and it can be extended by allocating additional levels when spare priorities exist.

STORM enforces this urgency using a two-phase transmission rule with receiver-coordinated active sets for long messages. If a message fits within one BDP (sub-BDP), the sender transmits it entirely at P1. If a message exceeds one BDP (super-BDP), the sender transmits the first BDP immediately, using P2 when the QP has positive backlog and P3 otherwise. Beyond the first BDP, the receiver advances a coarse sliding window and keeps at most two super-BDP QPs (that is, QPs with HoL WR size exceeding BDP) active at any time, one in P2 and one in P3; within each slot, it selects the most urgent QP by prioritizing deeper backlog and breaking ties by smaller remaining size. This receiver control is coarse-grained eligibility, not per-packet pacing.

3.2 Sender-side scheduling

Priority primitives. STORM assumes a small priority range within an existing service class. We describe the default three-level configuration, which uses priorities P1, P2, and P3. P1 is reserved for sub-BDP messages. P2 is reserved for super-BDP QPs with positive backlog. P3 is used for all other super-BDP QPs that are not admitted to P2. This separation is intentional: it ensures that backlog-carrying,

gating traffic is never forced to compete in the same queue with non-gating large transfers.

Sub-BDP messages. For a message whose size is at most one BDP, the sender transmits the entire message at priority P1. This realizes the core benefit of SRPT in the regime where it is most decisive: small messages complete quickly, reduce queueing for subsequent traffic, and do not materially delay large transfers.

Super-BDP messages: first-BDP burst. For a message larger than one BDP, the sender transmits the first BDP immediately to avoid startup stalls and to keep the pipe full. The sender selects the priority for this initial burst based on whether the QP is currently gating queued follow-on work: (i) if the QP backlog is positive, the first BDP is transmitted at priority P2; (ii) otherwise it is transmitted at priority P3. This rule makes backlog visible to the network at the earliest point where it matters, before any RTT-scale receiver coordination is possible.

Backlog reporting. Backlog is a per-QP signal that indicates how many *logical* transactions are queued behind the head-of-line transaction. Since WR fragmentation can distort both size and backlog, backlog is computed after the grouping mechanism in §3.5: consecutive WRs that belong to a single logical transfer are treated as one transaction. The sender reports backlog to the receiver by piggybacking the updated backlog value on outgoing data whenever it changes. To avoid stale receiver state when backlog increases but the sender has exhausted its granted window, STORM permits the sender to transmit one additional small backlog-update packet for that QP, even when no further data is currently allowed. This exception is strictly for control metadata; it does not carry payload data, and it does not advance the data window.

Sender-side arbitration across QPs. A sender may have multiple QPs eligible to transmit, either sub-BDP messages tagged at P1 or super-BDP QPs with receiver-granted windows at P2 or P3. The sender always serves the highest-priority eligible QP first. Within the same priority, it breaks ties using the same urgency rule as the receiver: prefer the QP with deeper backlog, and if backlog ties, prefer the QP with smaller remaining bytes for the head-of-line transaction. This preserves a consistent ordering within a sender and avoids diluting priority decisions when multiple QPs share the same class.

Beyond the first BDP. After the first BDP, transmission eligibility and priority are controlled by the receiver through coarse-grained grants. The sender maintains a per-QP "allowed" window (in bytes or sequence numbers) and transmits freely within the granted budget at the priority indicated by the receiver. The sender does not request permission per packet; it merely respects the current grant, ensuring the mechanism remains coarse and hardware-friendly.

3.3 Receiver-side active-set scheduling

The receiver coordinates super-BDP transmissions because it is the only endpoint that can arbitrate among multiple senders contending for the same destination bottleneck. Crucially, STORM uses the receiver only where RTT-scale coordination is acceptable: it does not attempt to control the first BDP, and it does not micro-schedule packets. Instead, the receiver maintains a small set of active QPs and advances their coarse sliding windows, which determines which QPs may continue transmitting and at what priority.

State and events. For each QP with an outstanding super-BDP message, the receiver maintains: (i) remaining bytes, (ii) most recent backlog value (piggybacked from the sender), (iii) whether the QP is active, and (iv) the current grant boundary. The receiver updates scheduling decisions only on discrete events: a new QP becomes eligible (arrival), the receiver observes a backlog update that changes a QP's rank, or an active QP finishes (remaining becomes zero). Between events, the active set remains unchanged, yielding stable behavior in steady state.

Active-set constraint and priority assignment. To prevent contention within a priority class from diluting desired scheduling, the receiver admits at most one active QP per super-BDP priority. In the default configuration, the receiver maintains two slots: a P2 slot for the most urgent backlog-positive QP and a P3 slot for the most urgent of the remaining eligible super-BDP QPs. Thus, priority assignment is also an admission-control rule: at any time, each receiver allows at most one active super-BDP QP to transmit at P2 and at most one to transmit at P3, while other super-BDP QPs wait for grants. If there is no backlog-positive QP, only the P3 slot is used by default, avoiding promotion of non-blocking traffic into the backlog-reserved priority. Implementations may also choose to let traffic without backlog use the P2 slot when idle to improve utilization. When a slot is free, the receiver admits the most urgent eligible QP of that class. When a slot is occupied, the receiver swaps the occupant only if a different QP in the same class becomes strictly more urgent. This restriction avoids frequent priority changes, prevents many super-BDP QPs from simultaneously entering the same priority, and bounds sustained long-flow contention within each priority class.

Granting rule. For each active QP, the receiver advances its window in coarse increments (e.g., two BDP per grant) upon receiving acknowledgments for a configurable fraction of the current window (e.g., half a BDP). Each grant carries the updated boundary and the priority (P2 or P3) the sender should use. The sender is then free to transmit within the new boundary, allowing the underlying congestion control to regulate fine-grained pacing. This grant size is deliberately generous: it keeps the pipe full and avoids turning receiver coordination into micro-pacing, while still ensuring that a demoted QP quickly stops once its granted budget is exhausted.

In-order delivery constraint. RoCEv2 requires in-order delivery, so changing a QP's priority while earlier packets are still in flight can induce reordering and trigger costly recovery. STORM therefore promotes a QP only after its in-flight packets at the previous priority have drained, and it applies a conservative promotion guard to avoid frequent switches: when backlog ties it promotes only if the remaining-size advantage exceeds one BDP, and it avoids promoting near-completion transfers whose remaining size is at most one BDP. This constraint is optional. In RDMA stacks that support out-of-order delivery and selective retransmission, such as Falcon and UEC [33, 56], the guard can be removed entirely. In §5.2.2 and §5.3 we evaluate the impact of this constraint and find that it only modestly affects collective performance while preserving correctness under in-order delivery.

3.4 Priority budget and variants

STORM is designed to operate under a constrained priority budget while retaining clear upgrade paths when additional priorities are available. The default configuration uses three priorities (P1–P3), while noting these are semantic classes rather than workload-tuned numerical thresholds. P1 separates sub-BDP messages, for which sender-local scheduling is the only timely option, from super-BDP traffic where receiver feedback can act. P2 and P3 then separate blocking from non-blocking super-BDP traffic using backlog: P2 is reserved for backlog-positive traffic, while P3 is used for other eligible super-BDP traffic. This separation enables both size-aware completion for short messages and backlog-aware unblocking for dependent pipelines. We further evaluate sensitivity to the priority budget in §6.2.

Two-priority minimal configuration. STORM remains functional with only two priorities by removing the dedicated sub-BDP class. Concretely, we collapse P1 into the super-BDP scheme and treat all messages using only P2 and P3: messages (including sub-BDP) use P2 when the QP has positive backlog and P3 otherwise, while the receiver continues to maintain at most two active super-BDP QPs (one per priority) beyond the first BDP. This reduction sacrifices some protection for small, independent transfers, but preserves the core unblock-first behavior and the receiver's active-set control.

More priorities for sub-BDP dominated regimes. When small messages dominate a service's traffic mix and additional priorities are available, STORM can refine the sub-BDP class by splitting P1 into multiple sub-BDP priorities. The sender assigns these priorities using message-size thresholds: these thresholds can be configured offline from expected workload distributions, or adjusted dynamically from observed traffic so that each priority carries a similar amount of traffic, following the approach used by Homa [37]. Smaller sub-BDP messages then receive higher priority. This extension improves tail completion time for latency-sensitive control and short transfers without changing receiver logic.

More priorities for super-BDP dominated regimes. When large transfers dominate and spare priorities exist, STORM can increase parallelism by allocating additional super-BDP priorities and maintaining one active QP per super-BDP priority. Operationally, the receiver selects the top-K candidates by the same urgency ordering (backlog first, then remaining size) and assigns them to the available active slots. This increases utilization under heavy super-BDP load while preserving the key property that each active QP has an exclusive priority class, avoiding dilution from within-class contention.

3.5 Signal semantics and grouping

STORM schedules *logical transactions*, not raw work requests. Both remaining size and per-QP backlog are defined at the granularity of a logical transfer that the application intends as one message, because many systems fragment a single transfer into multiple WRs for pipelining. For example, NCCL slices a chunk (e.g., 16 MB) into fixed-size slices (e.g., 512 KB) and posts a sequence of slice WRs to the same QP in each step [22]. Treating these slices independently would artificially inflate backlog and bias size-based prioritization.

We therefore group related WRs into a single transaction before computing remaining size and backlog. Backlog counts the number of grouped transactions queued behind the head-of-line transaction, and remaining size tracks the aggregate bytes of the head-of-line transaction as it drains. Grouping relies on a simple tagging mechanism: WRs that belong to the same logical transaction are tagged with a shared `wr_id`; the RNIC aggregates their sizes and treats them as one backlog unit, closing the group when the final WR completes (CQE) and then moving to the next group. There are two practical ways to provide this identifier: (i) at the RDMA stack, e.g., by extending `rdma-core` to attach the posting thread/process identifier (PID) as the `wr_id` for each posted request, or (ii) at a higher layer, e.g., by having the collective library assign a logical transaction ID per chunk/step and reuse it across the slice WRs that implement that chunk.

If an explicit identifier is unavailable, STORM conservatively groups temporally adjacent WRs on the same QP within a short timeout window. This timeout-based rule is only a fallback. A false split under the fallback makes the backlog estimate less precise, and may therefore slightly affect priority accuracy and performance, but it does not affect correctness, create additional traffic, or bypass any previously described rules.

4 Implementation

STORM is designed to be a thin, NIC-resident scheduling layer. To demonstrate practicality, we built a prototype on a Xilinx Alveo U250 FPGA by extending Coyote, an open-source FPGA RoCEv2 NIC [26]. Our changes preserve Coyote's transport data path and add only the minimum mechanisms required by STORM: (i) sender-side priority tagging and per-sender arbitration, and (ii) receiver-driven coarse grants that maintain a small active set of QPs beyond the first BDP. The prototype demonstrates that STORM's scheduling logic can be integrated into an RNIC datapath without disrupting line-rate packet processing. We therefore focus this section on the datapath/control-path split, the required per-QP state, and the additional FPGA resource cost.

4.1 Event-driven control

The implementation follows a logical disaggregation between a *data path* that enforces decisions at line rate and a *control path* that reacts to discrete events. The data path runs on the TX and RX pipelines and performs only constant-time actions per packet: it looks up per-QP state, checks whether transmission is permitted by the current grant boundary, tags the packet's priority, and emits lightweight events. The control path runs only on event boundaries (QP arrival, backlog update, completion, and grant-threshold milestones), and updates a small set of per-QP parameters that the data path consumes (active-slot membership, granted boundary, and current priority). This division keeps the fast path fully pipelined and avoids per-packet scheduling work when the system is in steady state.

Concretely, we implement the receiver policy in §3.3 as a two-slot controller. It maintains at most one active QP in P2 and one active QP in P3, then advances their windows by issuing GRANTs. On the sender, the data path arbitrates among eligible QPs by priority first, then by backlog depth, and finally by remaining size, matching the tie-break logic used at the receiver.

Resource	Used		Available	STORM
	STORM	Coyote		Utilization
LUT	2 692	311 190	1 728 000	0.16%
Register	1 767	565 809	3 456 000	0.05%
BRAM	4	427.5	2 688	0.15%
URAM	0	9	1 280	0%

Table 1: FPGA resource utilization for the STORM prototype on Alveo U250.

4.2 Implementing size and backlog signals

We implement STORM's two scheduling signals, remaining size and per-QP backlog, as small per-QP fields updated directly in the RNIC pipeline. On the send side, RDMA message length is explicit in the posted WQE: the WQE's data segments carry a byte count that the RNIC already consumes for DMA and packetization [40]. In our prototype, when a head-of-line transaction becomes active we latch its total length into the QP context and maintain a *remaining-bytes* counter, decrementing it as payload bytes are transmitted (TX) and, symmetrically, as progress is observed on RX. The scheduler only needs the head-of-line remaining bytes, so this counter is sufficient and avoids any dependence on verb-specific headers.

Backlog is implemented as a per-QP counter updated on the same RNIC events that define work arrival and completion. Doorbell updates make newly posted WQEs visible to the NIC, and CQEs mark completed work [40, 45]. To match STORM's semantics, we apply grouping (§3.5) and count *logical transactions* rather than raw WQEs: on a doorbell event we increment backlog only when a group head is posted, and on completion we decrement backlog only when the corresponding group tail completes. Backlog changes are piggybacked to the receiver on outgoing packets; if backlog increases while the sender is window-limited, we allow one additional backlog-update packet that carries only control metadata and does not advance the data window.

4.3 State extensions and FPGA cost

We extend Coyote's QP context table with a small amount of scheduling state. Per QP we add fields to store the current grant boundary, the currently assigned priority, remaining bytes of the head-of-line message, and the current backlog depth (after grouping). These fields are sufficient to implement both the sender datapath checks (grant compliance, tagging, sender-local arbitration) and the receiver control logic (slot selection and grant generation).

As Table 1 shows, the additional logic is negligible relative to the baseline NIC. The scheduler is implemented as a small number of counters and comparisons plus a two-slot controller, and the datapath remains fully pipelined. In our prototype, we observe no throughput penalty relative to the Coyote baseline, indicating that STORM can be integrated without degrading the critical path.

5 Evaluation

In this section, we demonstrate the effectiveness of STORM in cloud and AI training workloads.

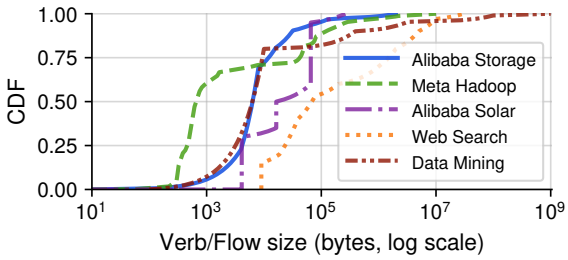


Figure 4: Size distributions for the cloud workloads used in this evaluation.

Model	Parameter size	Layers	Framework
GPT-3 13B	13B	40	Megatron
GPT-3 175B	175B	96	Megatron
LLaMA 65B	65.2B	80	Megatron
LLaMA 7B	6.7B	32	Deepspeed

Table 2: LLM training workloads used in this evaluation.

5.1 Experiment Setup

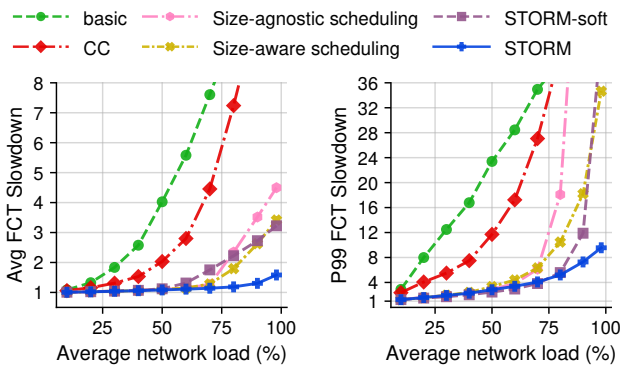
Our core STORM evaluation uses an extended version of the ns-3 simulator [46] for large-scale simulation, and we additionally validate practicality on a small hardware testbed. Building upon publicly available codebases [30, 58, 61], we implemented STORM. Our evaluation uses the following configurations.

Our evaluation focuses on scheduling within an existing service or QoS allocation. STORM is not intended to replace operator-level isolation across services; rather, it refines the ordering of RDMA requests that already share a service class. This matches reported production practice: Meta's RDMA training deployment places AI traffic on a separate back-end GPU network, while storage/RPC traffic uses front-end NIC networks with service-level QoS classes [15]. We therefore evaluate cloud and AI workloads separately, using each to stress a different dependency structure.

Network. We simulate a spine-leaf network, with 128 hosts attaching to eight top-of-rack switches, in turn connecting eight spine switches. All links run at 100 Gbps with 1 µs propagation latency. For LLM workloads, each host is equipped with 1–8 GPUs (depending on the model configuration; see below), connected by 600 GB/s NVLink to emulate A100-class intra-node bandwidth. The topology follows the configurations used in HPCC [30] and SimAI [61].

We model a PFC-enabled lossless fabric, matching common RoCEv2 deployments. Importantly, we enable dynamic buffer sharing across priorities (dynamic thresholds), so that using additional priority classes does not give STORM an inherent advantage from statically partitioned switch buffers. This ensures that any gains from STORM come from request prioritization itself, rather than from extra reserved buffering per priority.

Cloud workloads. Following established practice [2, 30, 58, 67, 68], we synthesize workload traces from published cumulative distribution functions (CDFs). As shown in Figure 4, we include three recent traffic profiles: the Alibaba Storage workload [30], which is RDMA-specific; Alibaba Solar [34], an RPC-IP workload accelerated with hardware; and Meta's Hadoop workload [47], representative



(a) Average FCT slowdown (b) 99%-tile FCT slowdown

Figure 5: Cloud performance across load. The slowdown is measured as the ratio of the observed flow completion time (FCT) to the theoretical optimum.

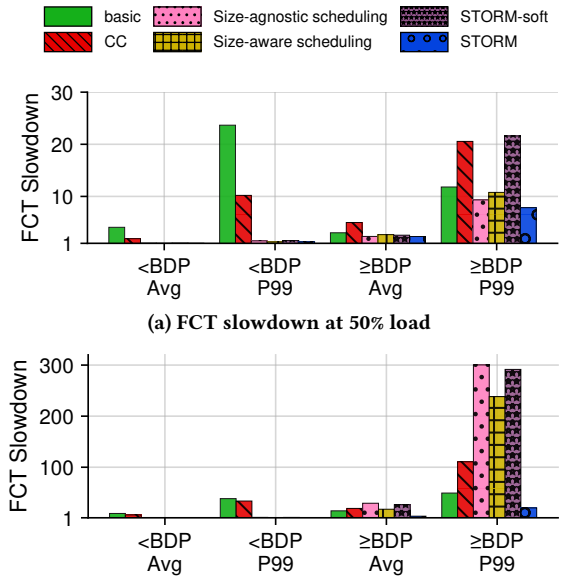
of traditional datacenter applications with a long tail. To facilitate comparisons with earlier studies, we also incorporate two classic workloads: web search [2] and data mining [17]. For each workload, flows arrive as a Poisson process; sizes are sampled from the CDF; source-destination pairs are chosen uniformly at random; and the arrival rate is tuned to reach the target offered load. This methodology matches prior work.

Across these five workloads, we observe consistent qualitative trends; unless otherwise stated, we present results for Alibaba Storage, which is both RDMA-native and representative of our target cloud setting.

AI training workloads. We evaluate STORM on AI training communication using SimAI [61], which generates NCCL collective workloads from published model and parallelization configurations. We consider both (i) standalone collective microbenchmarks to isolate communication behavior and (ii) end-to-end training-iteration traces that include the collective schedule and its dependencies. We simulate training four models: GPT-3 (13B and 175B) and LLaMA (7B and 65B). We follow SimAI's hyperparameter settings and summarize the key model parameters in Table 2. All collectives use NVIDIA NCCL, which implements ring- and tree-based algorithms [22, 61].

Collective communication differs from the cloud workloads above in two ways that matter for scheduling. First, collectives dispatch bursts of requests with synchronized start times and often identical sizes, rather than independent flows with random arrivals and a wide size distribution. Second, requests within a collective are dependency-coupled: for example, in ring AllReduce a node cannot transmit the next chunk until it has received the predecessor chunk [22, 48]. These properties make performance sensitive to stragglers and head-of-line blocking rather than average per-verb completion, as discussed in §2.1.

Testbed. We use four servers, each equipped with a Mellanox ConnectX-6 RNIC, connected to an Arista 7060CX-32S switch. Three links run at 100 Gbps and one runs at 50 Gbps, creating a controlled bottleneck to emulate contention.



(a) FCT slowdown at 50% load

(b) FCT slowdown at 80% load

Figure 6: Cloud performance by request size.

Performance metrics. For cloud workloads, we report flow completion time (FCT), since each flow corresponds to an RPC or a similar request. Following prior work [30, 37, 58], we normalize by the theoretical minimum FCT, defined as transmission time plus propagation delay, and report slowdown relative to this baseline. For AI training, we measure collective completion time in standalone collective microbenchmarks, and we measure end-to-end training iteration time in SimAI traces. These metrics directly capture the throughput of the training job.

Schemes compared. We compare STORM with a range of datacenter congestion-control schemes: DCTCP [2], DCQCN [68], TIMELY [35], HPCC [30], ExpressPass [10], RCC [67], pHost [16], Homa [37], and PIAS [5]. We have included pHost, Homa, and PIAS, as there is, to the best of our knowledge, no established RDMA-native SRPT design. These systems are the strongest available references, while noting the protocol mismatches that prevent a direct RDMA deployment.

We use the ns-3 implementations from Li et al. [30] for DCTCP, DCQCN, TIMELY, and HPCC, scaling parameters to 100 Gbps and disabling DCTCP's slow-start phase for fairness. We have verified that our parameter settings align with prior work [58, 61]. We also include a "basic" configuration with no congestion control and naive scheduling, aligning with reports of Meta RDMA deployments [15]. For clarity in plots, we report one representative from each baseline category based on the best overall performance in our experiments: DCQCN for congestion control, PIAS for size-agnostic scheduling, and Homa for size-aware scheduling.

We also include STORM-soft, a pure software variant that assigns priorities before WRs enter the RNIC. STORM-soft does not implement receiver-side grants or active-set control; it only uses the WR size available at posting time to assign a fixed priority to each request. This baseline tests whether software-only priority

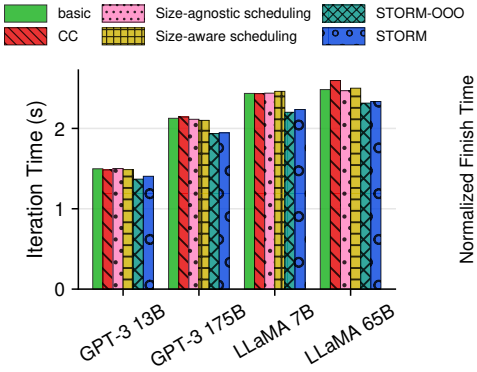


Figure 7: End-to-end training performance for different LLM models

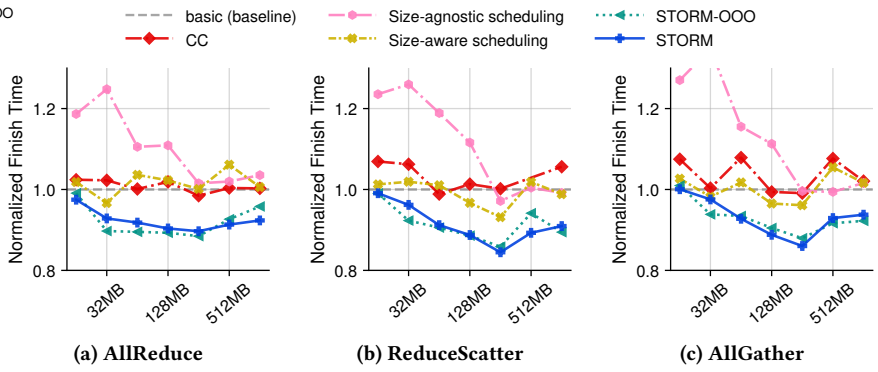


Figure 8: Large-scale simulation collective performance. Normalized against basic configuration (no CC, naive scheduling).

marking through existing QoS mechanisms can capture the benefit of STORM. Compared with PIAS, STORM-soft has exact size information at posting time but cannot adjust a request's priority as it receives service; PIAS can demote flows at runtime based on attained service, but it does not know the exact request size in advance.

5.2 Experiment Results

5.2.1 Cloud applications. Figure 5 shows the performance of STORM and competing schemes under the Alibaba Storage workload. STORM consistently delivers the lowest average slowdown across all offered loads, and its advantage becomes most pronounced at high load. At low and moderate loads, several scheduling schemes are competitive because queueing is limited and most short transfers already complete near their ideal time. As load increases, however, the differences between the schemes widen. Size-agnostic scheduling improves over congestion control by using runtime demotion, while size-aware scheduling further benefits from exact request-size information. STORM-soft isolates the software-only case: it knows the WR size at posting time and can assign an initial priority, but the priority is fixed once the request enters the RNIC and there is no receiver-side active-set control. As a result, the average and tail slowdowns of STORM-soft rise substantially at high load. STORM remains the most robust scheme because it combines exact size information with backlog awareness and receiver-side control over which super-BDP QPs may continue transmitting.

Figure 6 breaks results down by verb size and illustrates where this difference comes from. For sub-BDP verbs, all scheduling schemes with prioritization are close to optimal, leaving little room for further improvement; STORM preserves this small-verb performance rather than improving it at the expense of other traffic. The main separation appears for super-BDP verbs, especially at high load. Pure size-based or software-only prioritization can keep large transfers deprioritized for too long, even when they are blocking queued work behind the same QP. STORM avoids this failure mode by using backlog size as an additional urgency signal: a large head-of-line request with queued dependent work is no longer treated as just another long transfer. Receiver-side active-set control then limits how many super-BDP QPs can keep transmitting at each priority, reducing sustained within-class contention. As a result, STORM improves small-verb latency while substantially reducing large-verb average and tail slowdown under heavy contention.

5.2.2 AI training. STORM also improves collective-dominated AI communication, as shown in Figure 7. Across all four SimAI training workloads, STORM reduces end-to-end iteration time by roughly 10% relative to the basic baseline, while none of the compared baselines consistently beat the basic setting. This highlights that, for collectives, there is still meaningful headroom in how the NIC arbitrates requests within a single job: better prioritization can shorten the transient "bubbles" that form when a few delayed transfers stall the dependency chain (§2.1).

To understand what drives the gain, we compare against both size-agnostic and size-aware scheduling. In our evaluation, they track the basic configuration closely on collective-heavy workloads: they primarily optimize per-transfer completion, but they do not reliably target the transfers that gate collective progress under dependencies. In contrast, STORM consistently improves iteration time by prioritizing QPs that are either near completion or blocking queued dependent work, which directly targets the straggler-driven critical path. Across the evaluated LLM workloads, around 30% of traffic enters P2, indicating that a substantial fraction of communication is backlog-positive and benefits from backlog-aware scheduling.

We also include STORM-OOO, which removes the in-order promotion constraint by allowing out-of-order delivery in the simulator: packets may arrive reordered, and a reorder buffer reassembles them before delivering to the RDMA layer. STORM-OOO is only slightly better than STORM in end-to-end training, indicating that the core benefit comes from backlog-aware prioritization, and that the in-order constraint is not the dominant limiter for these workloads.

The per-collective microbenchmarks in Figure 8 show the same pattern. The size-agnostic baseline is particularly weak for smaller collectives, consistent with demotion-by-attained-service behavior that can deprioritize transfers even when they are close to completion. As collective size increases, STORM's gains grow, reaching up to about 10% for AllReduce and up to about 15% for ReduceScatter in our settings. We again find STORM-OOO only occasionally

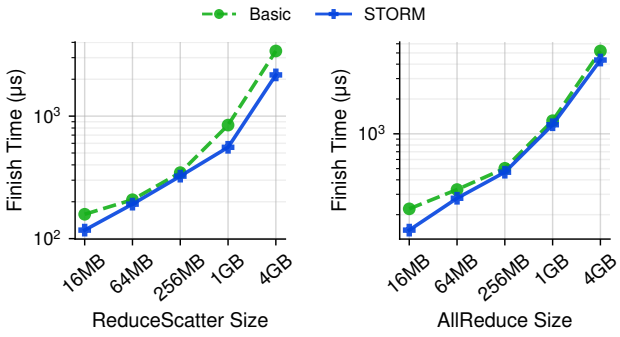
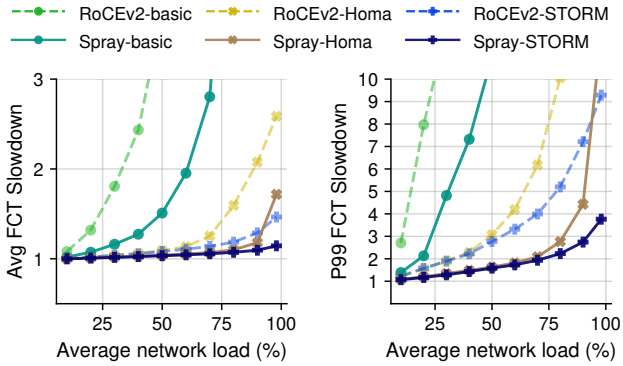


Figure 9: Testbed collective performance



(a) Average FCT slowdown **(b) 99%-tile FCT slowdown**

Figure 10: Performance of STORM with RoCEv2 and custom RDMA systems under different network load

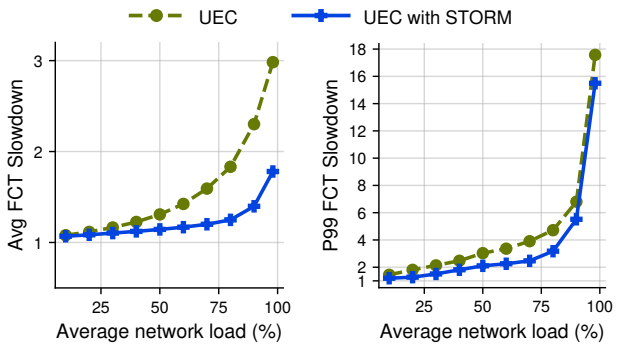
better than STORM, suggesting that enforcing in-order delivery modestly affects performance for collective traffic.

Finally, STORM outperforms DCQCN by up to 12% and improves over the basic baseline by up to 10%. Notably, CC methods are not consistently beneficial for these AI workloads, aligning with prior observations that conventional RDMA congestion control can provide limited value for collective-dominated training traffic [15].

5.2.3 Testbed. In our testbed, we emulate STORM's prioritization on ConnectX-6 RNICs without enforcing receiver window control. We enable strict-priority queueing on both the RNIC and the switch so that tagged priorities are enforced on the wire. We compare STORM against a basic baseline with naive scheduling. As shown in Figure 9, STORM achieves 8%-53% speedup in completion time across our test cases, demonstrating that priority-based RDMA request scheduling translates into tangible gains on commodity hardware.

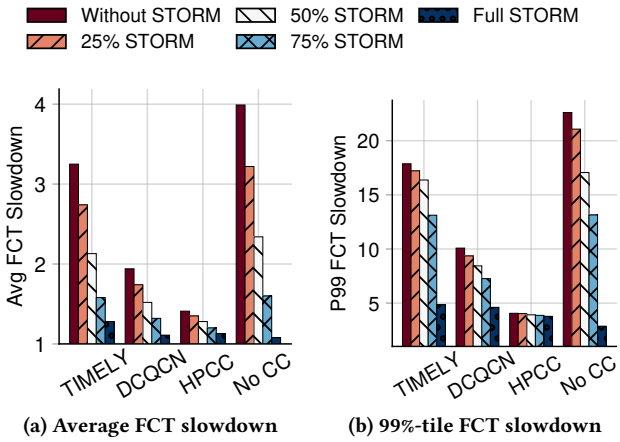
5.3 Applicability to custom RDMA systems

As discussed in §2.3, STORM is protocol-agnostic: it relies only on NIC-visible signals and commodity priority mechanisms, not on RoCEv2-specific header formats or loss semantics. This matters in practice because hyperscale deployments increasingly use custom RDMA transports that diverge from RoCEv2, often adding out-of-order delivery, alternative retransmission mechanisms, and different routing choices. Examples include Amazon SRD [49], Google



(a) Average slowdown **(b) 99%-tile slowdown**

Figure 11: Performance of STORM with UEC



(a) Average FCT slowdown **(b) 99%-tile FCT slowdown**

Figure 12: Incremental deployment and congestion-control interaction. Average and 99%-tile FCT slowdown under several congestion-control baselines as 0%, 25%, 50%, 75%, or 100% of hosts deploy STORM.

1RMA and Falcon [55, 56], Alibaba Stellar [31], and the Ultra Ethernet Consortium (UEC) proposals [33]. Many of these systems also rely on packet spraying or other multipath schemes rather than static ECMP hashing.

We evaluate STORM under this broader design point in two steps. First, we implement a simplified sprayed transport that permits reordering and therefore removes the in-order promotion constraint. Figure 10 shows that STORM continues to outperform the size-aware baseline on this transport, indicating that its prioritization and coarse receiver gating remain effective even when packets are sprayed and reordered. Second, we integrate STORM into the htsim UEC simulator [33]. As shown in Figure 11, STORM again improves performance over baseline UEC, demonstrating that its core mechanisms carry over to emerging RDMA stacks without requiring protocol-specific tuning.

6 Discussion and Future Work

6.1 Incremental deployment

STORM can be deployed incrementally and works with existing congestion-control mechanisms. We evaluate this by upgrading

Configuration	Avg Slowdown	P99 Slowdown
basic	9.560	43.571
CC (DCQCN)	6.128	31.452
Size-agnostic (PIAS)	2.345	18.086
Size-aware (Homa)	1.794	10.471
STORM-2	1.315	8.628
STORM-3 (default)	1.230	6.355
STORM-4	1.197	5.104
STORM-7	1.189	5.012

Table 3: Sensitivity analysis of STORM with various priority configurations (STORM-x denotes a configuration using x priority levels.) STORM-3 is used in all previous evaluations.

only a fraction of hosts and running the same workload under several congestion-control baselines. Figure 12 reports average and 99%-tile FCT slowdown for the non-STORM baseline and deployment fractions from 25% to 100%.

Incremental rollout. In a mixed cluster, STORM operates in full mode only when both endpoints are upgraded. For mixed pairs, we use a compatibility mode that preserves correctness without receiver-driven grants: sub-BDP requests follow the sender-side priority rule, while larger requests fall back to a fixed priority without receiver windowing. Despite this conservative fallback, Figure 12 shows that even 25% deployment reduces both average and tail slowdown, with monotonic improvement as more hosts are upgraded.

Interaction with congestion control. STORM controls which requests are eligible to transmit and at what priority, while congestion control regulates sending rate based on feedback. Figure 12 shows that STORM's gains are consistent across congestion-control choices: adding STORM on top of a baseline improves both mean and tail slowdown relative to that baseline alone. The figure also suggests diminishing returns from layering additional rate control on top of STORM in this setting, because STORM's receiver gating already limits persistent queue buildup and reduces head-of-line blocking. We do not use this figure to rank CC schemes, whose relative ordering varies with workload and tuning; instead, we compare each CC baseline against the same CC augmented with STORM. For AI training, we find that DCQCN+STORM provides a small additional benefit (about 1%) over STORM without congestion control.

6.2 How many priority levels are needed?

Our default configuration uses three total priority levels, corresponding to P1–P3 in §3.4. In production, the available priority budget can be smaller or larger because priorities are often pre-partitioned for tenant isolation or coarse service-level functions (e.g., management traffic). We therefore quantify STORM's sensitivity to the number of additional priorities, as described in §3.4.

We find that AI collective workloads are largely insensitive to the priority budget, and we omit them for brevity. Likewise, under moderate and low offered loads in the cloud workloads, the differences between configurations are small, indicating that a single extra priority is often sufficient outside of heavy contention. We also tested whether additional sub-BDP priorities improve performance, as suggested in §3.4. In our evaluated workloads, however, sub-BDP requests already achieve near-optimal slowdown under the default P1/P2/P3 split, leaving little room for further improvement from finer sub-BDP bins. We therefore focus the sensitivity analysis on adding priorities to super-BDP traffic, where heavy-load queueing and tail slowdown remain more pronounced.

Table 3 reports the most challenging regime: the cloud workload at 80% offered load. Average slowdown improves modestly as priorities increase, whereas tail slowdown is more sensitive, reflecting that additional priority resolution primarily helps isolate and drain the worst-case queueing episodes. With one more priority beyond the default, STORM-4 reduces P99 slowdown by roughly 20% relative to STORM-3. We do not use STORM-4 in our main evaluation, as our goal is to keep the default configuration as deployable as possible under tight priority budgets. Adding further priorities provides diminishing returns: STORM-7 improves only marginally over STORM-4, while consuming three additional priority levels. Overall, these results show that STORM remains effective under tight priority budgets, and that most of the benefit is captured with a small number of extra priorities, with additional levels mainly improving the extreme tail under heavy load.

6.3 Limitations and future work

STORM assumes strict-priority queues, which are widely available but provide weaker fairness isolation than WRR under adversarial traffic mixes: under sustained high-priority load, low-priority traffic can be delayed. We did not observe starvation in our workloads, but pathological mixes remain possible; a simple guard such as a bandwidth floor or a wait-time trigger can mitigate this while keeping the design NIC-friendly. Second, STORM currently treats backlog and remaining size as the primary urgency signals. This is largely orthogonal to coflow and framework-level schedulers that infer higher-level structure above the transport; future variants could incorporate such semantics as an optional input once coflow definitions and scheduling objectives are better understood, while still keeping the RNIC decision logic event-driven and NIC-friendly. More broadly, STORM is constrained by a small, discrete priority budget; if future switches expose finer-grained queues, an interesting direction is to map the same NIC-visible signals into weights or per-packet ordering for higher precision and robustness.

7 Related Work

Datacenter traffic scheduling has long used priorities to reduce mean and tail completion time by steering scarce bandwidth toward the most urgent transfers. In-network designs such as pFabric [3] push fine-grained ordering into switches, while end-host schemes such as PIAS [5] and QCLIMB [29] approximate size-aware prioritization. Receiver-driven transports such as pHost [16] and Homa [37] gather exact flow size and shift arbitration to receivers via grants or credits, and QJump [18] provides predictable latency with static priorities plus rate limiting of higher-priority traffic. STORM is closest in spirit to these priority-based flow schedulers, but it targets RDMA requests and exploits an RDMA-specific opportunity that these systems do not have: the NIC sees both the exact size at posting time and the depth of pending work queued behind a QP's head-of-line request.

A complementary line of work schedules groups of related flows using application-level structure. Coflow schedulers either assume prior knowledge of coflows (e.g., Varys [12], Baraat [13]) or infer coflows online without explicit annotations (e.g., Aalo [11], CODA [66]). For training, systems such as ByteScheduler [41] and Lina [27] similarly exploit framework-visible structure to schedule communication, and recent work extends coflow-style scheduling to modern training workloads, including LLM training [60]. These approaches typically operate above the transport (identifying, grouping, and prioritizing higher-level units), whereas STORM operates below (prioritizing the resulting RDMA verbs), without application hints, so they are largely complementary. Congestion control and alternative RDMA transports are similarly orthogonal: CC regulates sending rate with feedback, and custom RDMA stacks change loss and/or reordering semantics, while STORM focuses on request ordering and arbitration at the RNIC and priority enforcement using commodity queues.

Programmable NICs and SmartNIC resource-management systems are relevant as other possible implementation substrates. Platforms such as Corundum provide open FPGA NIC datapaths for high-speed packet processing [14], while systems such as OSMOSIS explore resource multiplexing and QoS management inside SmartNICs [25]. If a programmable or commodity SmartNIC exposes an RDMA-capable datapath, priority-tagging hooks, and sufficient per-QP state, then STORM's sender-side classification and receiver-side active-set logic could be implemented on top of that substrate.

8 Conclusion

This paper presents STORM, a NIC-resident RDMA request scheduler that converts RDMA-native signals already visible to the RNIC, namely request size and per-QP backlog, into a practical urgency signal. STORM uses only one or two additional priority levels, together with a lightweight receiver gate for long transfers. It requires no application hints, is compatible with both in-order RoCEv2 and reordering-tolerant RDMA stacks, and incurs negligible hardware overhead. Across cloud and collective-dominated workloads, STORM reduces training iteration time by up to 12% and cuts average and P99 FCT slowdown by up to 90% relative to fair scheduling, demonstrating substantial performance headroom in NIC-level scheduling.

Ethical issues. This work does not raise any ethical issues.

Acknowledgments

We thank our shepherd and the anonymous reviewers for their constructive feedback and guidance. We are grateful to Rolf Neugebauer, Michael Schapira, and Noa Zilberman for their helpful comments and discussions on earlier versions of this work. We also thank Wei Bai for helpful discussions that improved our baseline implementation.

References

[1] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. 2018. Sincronia: near-optimal network design for coflows. 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, 16–29. doi:10.1145/3230543.3230569

[2] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP DCTCP. In *Proceedings of the ACM SIGCOMM 2010 Conference* (New Delhi, India) *(SIGCOMM '10)*. Association for Computing Machinery, New York, NY, USA, 63–74. doi:10.1145/1851182.1851192

[3] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (Hong Kong, China) *(SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 435–446. doi:10.1145/2486001.2486031

[4] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. 2023. Empowering Azure Storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 49–67. https://www.usenix.org/conference/nsdi23/presentation/bai

[5] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Weicheng Sun. 2014. PIAS: Practical Information-Agnostic Flow Scheduling for Data Center Networks. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks* (Los Angeles, CA, USA) *(HotNets-XIII)*. Association for Computing Machinery, New York, NY, USA, 1–7. doi:10.1145/2670518.2673871

[6] A. Bazavov, D. Toussaint, C. Bernard, J. Laiho, C. DeTar, L. Levkova, M. B. Oktay, Steven Gottlieb, U. M. Heller, J. E. Hetrick, P. B. Mackenzie, R. Sugar, and R. S. Van de Water. 2010. Nonperturbative QCD simulations with 2 + 1 flavors of improved staggered quarks. *Rev. Mod. Phys.* 82 (5 2010), 1349–1417. Issue 2. doi:10.1103/RevModPhys.82.1349

[7] Jiamin Cao, Yu Guan, Kun Qian, Jiaqi Gao, Wencong Xiao, Jianbo Dong, Binzhang Fu, Dennis Cai, and Ennan Zhai. 2024. Crux: GPU-Efficient Communication Scheduling for Deep Learning Training. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia) *(ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3651890.3672239

[8] Li Chen, Kai Chen, Wei Bai, and Mohammad Alizadeh. 2016. Scheduling Mix-flows in Commodity Datacenters with Karuna. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 174–187. doi:10.1145/2934872.2934888

[9] Marco Chiesa, Roshan Sedar, Gianni Antichi, Michael Borokhovich, Andrzej Kamisiński, Georgios Nikolaidis, and Stefan Schmid. 2021. Fast ReRoute on Programmable Switches. *IEEE/ACM Transactions on Networking* 29, 2 (2021), 637–650. doi:10.1109/TNET.2020.3045293

[10] Inho Cho, Keon Jang, and Dongsu Han. 2017. Credit-Scheduled Delay-Bounded Congestion Control for Datacenters. 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, 239–252. doi:10.1145/3098822.3098840

[11] Mosharaf Chowdhury and Ion Stoica. 2015. Efficient Coflow Scheduling Without Prior Knowledge. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 393–406. doi:10.1145/2785956.2787480

[12] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient coflow scheduling with Varys. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 443–454. doi:10.1145/2619239.2626315

[13] Fahad R. Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony Rowstron. 2014. Decentralized task-aware scheduling for data center networks. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 431–442. doi:10.1145/2619239.2626322

[14] Alex Forencich, Alex C. Snoeren, George Porter, and George Papen. 2020. Corundum: An Open-Source 100-Gbps Nic. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 38–46. doi:10.1109/FCCM48280.2020.00015

[15] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia)

(ACM SIGCOMM '24). Association for Computing Machinery, New York, NY, USA, 57–70. doi:10.1145/3651890.3672233

[16] 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)*. Association for Computing Machinery, New York, NY, USA, Article 1, 12 pages. doi:10.1145/2716281.2836086

[17] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.* 39, 4 (Aug. 2009), 51–62. doi:10.1145/1594977.1592576

[18] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft. 2015. Queues Don't Matter When You Can JUMP Them!. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 1–14. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/grosvenor

[19] Chuanxiong Guo. 2017. RDMA in data centers: Looking back and looking forward. *Keynote at APNet* (2017). https://conferences.sigcomm.org/events/apnet2017/slides/cx.pdf

[20] Chi-Yao Hong, Matthew Caesar, and P. Brighten Godfrey. 2012. Finishing flows quickly with preemptive scheduling. *SIGCOMM Comput. Commun. Rev.* 42, 4 (Aug. 2012), 127–138. doi:10.1145/2377677.2377710

[21] Wentao Hou, Jie Zhang, Zeke Wang, and Ming Liu. 2024. Understanding Routable PCIe Performance for Composable Infrastructures. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 297–312. https://www.usenix.org/conference/nsdi24/presentation/hou

[22] Zhiyi Hu, Siyuan Shen, Tommaso Bonato, Sylvain Jeaugey, Cedell Alexander, Eric Spada, James Dinan, Jeff Hammond, and Torsten Hoefler. 2025. Demystifying NCCL: An In-depth Analysis of GPU Communication Protocols and Algorithms. arXiv:2507.04786 [cs.DC] https://arxiv.org/abs/2507.04786

[23] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 463–479. https://www.usenix.org/conference/osdi20/presentation/jiang

[24] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2014. Using RDMA Efficiently for Key-Value Services. *SIGCOMM Comput. Commun. Rev.* 44, 4 (8 2014), 295–306. doi:10.1145/2740070.2626299

[25] Mikhail Khalilov, Marcin Chrapek, Siyuan Shen, Alessandro Vezzu, Thomas Benz, Salvatore Di Girolamo, Timo Schneider, Daniele De Sensi, Luca Benini, and Torsten Hoefler. 2024. OSMOSIS: Enabling Multi-Tenancy in Datacenter SmartNICs. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. USENIX Association, Santa Clara, CA, 247–263. https://www.usenix.org/conference/atc24/presentation/khalilov

[26] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. 2020. Do OS abstractions make sense on FPGAs?. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 991–1010. https://www.usenix.org/conference/osdi20/presentation/roscoe

[27] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. 2023. Accelerating Distributed MoE Training and Inference with Lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 945–959. https://www.usenix.org/conference/atc23/presentation/li-jiamin

[28] Pengfei Li, Yu Hua, Pengfei Zuo, Zhangyu Chen, and Jiajie Sheng. 2023. ROLEX: A Scalable RDMA-oriented Learned Key-Value Store for Disaggregated Memory Systems. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. USENIX Association, Santa Clara, CA, 99–114. https://www.usenix.org/conference/fast23/presentation/li-pengfei

[29] Wenxin Li, Xin He, Yuan Liu, Keqiu Li, Kai Chen, Zhao Ge, Zewei Guan, Heng Qi, Song Zhang, and Guyue Liu. 2024. Flow Scheduling with Imprecise Knowledge. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 95–111. https://www.usenix.org/conference/nsdi24/presentation/li-wenxin

[30] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 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

[31] Jie Lu, Jiaqi Gao, Fei Feng, Zhiqiang He, Menglei Zheng, Kun Liu, Jun He, Binbin Liao, Suwei Xu, Ke Sun, Yongjia Mo, Qinghua Peng, Jilie Luo, Qingxu Li, Gang Lu, Zishu Wang, Jianbo Dong, Kunling He, Sheng Cheng, Jiamin Cao, Hairong Jiao, Pengcheng Zhang, Shu Ma, Lingjun Zhu, Chao Shi, Yangming Zhang, Yiquan Chen, Wei Wang, Shuhong Zhu, Xingru Li, Qiang Wang, Jiang Liu, Chao Wang, Wei Lin, Ennan Zhai, Jiesheng Wu, Qiang Liu, Binzhang Fu, and Dennis Cai. 2025. Alibaba Stellar: A New Generation RDMA Network for Cloud AI. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) *(SIGCOMM '25)*. Association for Computing Machinery, New York, NY, USA, 453–466. doi:10.1145/3718958.3750539

[32] Yuanwei Lu, Guo Chen, Larry Luo, Kun Tan, Yongqiang Xiong, Xiaoliang Wang, and Enhong Chen. 2017. One more queue is enough: Minimizing flow completion time with explicit priority notification. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 1–9. doi:10.1109/INFOCOM.2017.8056946

[33] J Metz. 2024. Empowering AI Workloads in Ultra Ethernet Consortium. In *2024 IEEE Photonics Society Summer Topicals Meeting Series (SUM)*. 1–2. doi:10.1109/SUM60964.2024.10614558

[34] Rui Miao, Lingjun Zhu, Shu Ma, Kun Qian, Shujun Zhuang, Bo Li, Shuguang Cheng, Jiaqi Gao, Yan Zhuang, Pengcheng Zhang, Rong Liu, Chao Shi, Binzhang Fu, Jiaji Zhu, Jiesheng Wu, Dennis Cai, and Hongqiang Harry Liu. 2022. From luna to solar: the evolutions of the compute-to-storage networks in Alibaba cloud. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 753–766. doi:10.1145/3544216.3544238

[35] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 537–550. doi:10.1145/2829988.2787510

[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)*. Association for Computing Machinery, New York, NY, USA, 221–235. doi:10.1145/3230543.3230564

[38] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3341301.3359646

[39] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient large-scale language model training on GPU clusters using megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 58, 15 pages. doi:10.1145/3458817.3476209

[40] NVIDIA Corporation. 2022. *Mellanox Adapters Programmer's Reference Manual (PRM)*. https://network.nvidia.com/files/doc-2020/ethernetadapters-programming-manual.pdf Accessed: 2026-02-06.

[41] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. 2019. A generic communication scheduler for distributed DNN training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 16–29. doi:10.1145/3341301.3359642

[42] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: a centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 307–318. doi:10.1145/2619239.2626309

[43] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. 2024. CASSINI: Network-Aware Job Scheduling in Machine Learning Clusters. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1403–1420. https://www.usenix.org/conference/nsdi24/presentation/rajasekaran

[44] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 3505–3506. doi:10.1145/3394486.3406703

[45] Feng Ren, Mingxing Zhang, Kang Chen, Huaxia Xia, Zuoning Chen, and Yongwei Wu. 2024. Scaling Up Memory Disaggregated Applications with SMART. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 351–367. doi:10.1145/3617232.3624857

[46] George F. Riley and Thomas R. Henderson. 2010. The ns-3 Network Simulator. In *Modeling and Tools for Network Simulation*, Klaus Wehrle, Mesut Güneş, and James Gross (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 15–34. doi:10.1007/978-3-642-12331-3_2

[47] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. *SIGCOMM Comput. Commun. Rev.* 45, 4 (8 2015), 123–137. doi:10.1145/2829988.2787472

[48] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[49] 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

[50] Tom Shanley. 2003. *InfiniBand network architecture*. Addison-Wesley Professional.

[51] Naveen Kr. Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. 2020. Programmable Calendar Queues for High-speed Packet Scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 685–699. https://www.usenix.org/conference/nsdi20/presentation/sharma

[52] Jiacheng Shen, Pengfei Zuo, Xuchuan Luo, Tianyi Yang, Yuxin Su, Yangfan Zhou, and Michael R. Lyu. 2023. FUSEE: A Fully Memory-Disaggregated Key-Value Store. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. USENIX Association, Santa Clara, CA, 81–98. https://www.usenix.org/conference/fast23/presentation/shen

[53] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL] https://arxiv.org/abs/1909.08053

[54] Vishal Shrivastav. 2019. Fast, scalable, and programmable packet scheduler in hardware. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 367–379. doi:10.1145/3341302.3342090

[55] 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

[56] 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 *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) *(SIGCOMM '25)*. Association for Computing Machinery, New York, NY, USA, 248–263. doi:10.1145/3718958.3754353

[57] William C Skamarock, Joseph B Klemp, Jimy Dudhia, David O Gill, Zhiquan Liu, Judith Berner, Wei Wang, Jordan G Powers, Michael G Duda, Dale M Barker, et al. 2019. A description of the advanced research WRF version 4. *NCAR tech. note ncar/tn-556+ str* 145 (2019).

[58] 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

[59] Vojislav Đukić, Sangeetha Abdu Jyothi, Bojan Karlas, Muhsen Owaida, Ce Zhang, and Ankit Singla. 2019. Is advance knowledge of flow sizes a plausible assumption?. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 565–580. https://www.usenix.org/conference/nsdi19/presentation/dukic

[60] Xinchen Wan, Xinyu Yang, Kaiqiang Xu, Xudong Liao, Yilun Jin, Yijun Sun, Zhenghang Ren, Han Tian, and Kai Chen. 2025. Coflow Scheduling for LLM Training. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) *(SIGCOMM '25)*. Association for Computing Machinery, New York, NY, USA, 1232–1234. doi:10.1145/3718958.3750467

[61] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, Yang Liu, Pengcheng Zhang, Kun Qian, Kunling He, Jiaqi Gao, Ennan Zhai, Dennis Cai, and Binzhang Fu. 2025. SimAI: Unifying Architecture Design and Performance Tuning for Large-Scale Large Language Model Training with Scalability and Precision. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 541–558. https://www.usenix.org/conference/nsdi25/presentation/wang-xizheng-simai

[62] 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

[63] Xingda Wei, Rong Chen, and Haibo Chen. 2020. Fast RDMA-based Ordered Key-Value Store using Remote Learned Cache. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 117–135. https://www.usenix.org/conference/osdi20/presentation/wei

[64] Henry G Weller, Gavin Tabor, Hrvoje Jasak, and Christer Fureby. 1998. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in physics* 12, 6 (1998), 620–631.

[65] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. 2011. Better never than late: meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 Conference* (Toronto, Ontario, Canada) *(SIGCOMM '11)*. Association for Computing Machinery, New York, NY, USA, 50–61. doi:10.1145/2018436.2018443

[66] Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, and Yanhui Geng. 2016. CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 160–173. doi:10.1145/2934872.2934880

[67] Jiao Zhang, Xiaolong Zhong, Zirui Wan, Yu Tian, Tian Pan, and Tao Huang. 2023. RCC: Enabling Receiver-Driven RDMA Congestion Control With Congestion Divide-and-Conquer in Datacenter Networks. *IEEE/ACM Transactions on Networking* 31, 1 (2023), 103–117. doi:10.1109/TNET.2022.3185105

[68] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. *SIGCOMM Comput. Commun. Rev.* 45, 4 (8 2015), 523–536. doi:10.1145/2829988.2787484