# Reimagining RDMA Through the Lens of ML

Ertza Warraich, Ali Imran, Annus Zulfiqar, Shay Vargaftik, Sonia Fahmy, and Muhammad Shahbaz

***Abstract*—As distributed machine learning (ML) workloads scale to thousands of GPUs connected by ultra-high-speed interconnects, tail latency in collective communication has emerged as a primary bottleneck. Prior RDMA designs, like RoCE, IRN, and SRNIC, enforce strict reliability and in-order delivery, relying on retransmissions and packet sequencing to ensure correctness. While effective for general-purpose workloads, these mechanisms introduce complexity and latency that scale poorly, where even rare packet losses or delays can consistently degrade system performance. We introduce Celeris, a domain-specific RDMA transport that revisits traditional reliability guarantees based on ML's tolerance for lost or partial data. Celeris removes retransmissions and in-order delivery from the RDMA NIC, enabling best-effort transport that exploits the robustness of ML workloads. It retains congestion control (e.g., DCQCN) and manages communication with software-level mechanisms such as adaptive timeouts and data prioritization, while shifting loss recovery to the ML pipeline (e.g., using the Hadamard Transform). Early results show that Celeris reduces 99th-percentile latency by up to 2.3×, cuts BRAM usage by 67%, and nearly doubles NIC resilience to faults—delivering a resilient, scalable transport tailored for ML at cluster scale.**

***Index Terms*—Data Centers; Hardware Accelerators; Network Transport; AI Workloads; Tail Latency; Disaggregation; SLO**

## I. Introduction

DISTRIBUTED machine learning (ML) workloads (training and inference) now span thousands of GPUs connected by ultra-high-speed 100–400G interconnects. As models grow and clusters scale, the primary bottleneck shifts from compute to communication [1], [2]. Collective operations such as AllReduce, AllGather, and All-to-All lie on the critical path of both data- and model-parallel processing. These operations introduce global synchronization points, where even minor transport delays can stall progress across the entire cluster. As a result, tail latency—not average throughput—has become the dominant limiter of model efficiency at scale [1], [2].

To mitigate communication overhead, the community has focused on optimizing collective algorithms (e.g., NCCL, RCCL, and MSCCL) and reducing traffic via sparsification, quantization, and other compression techniques [3], [4]. These advances build on a foundational insight: ML workloads processed with stochastic gradient descent (SGD) are statistically resilient; they tolerate noise, approximation, and even bounded loss without hurting convergence [1], [2].

Yet, the underlying transport stack remains overly conservative (Table I). RoCE, the de facto RDMA protocol in ML clusters, is designed for strict reliability and correctness [5]. It enforces in-order delivery, retransmissions, and lossless operation using Priority Flow Control (PFC). While effective for general-purpose workloads (e.g., key-value stores, distributed databases, and RPCs), these mechanisms increase hardware complexity and introduce latency spikes that scale poorly with cluster size. A single packet drop can trigger go-back-N retransmissions or cascade into fabric-wide PFC stalls, inflating tail latency and limiting scalability [6].

Efforts like IRN [7] eliminate PFC by handling packet loss directly in the NIC. IRN uses selective repeat with bitmap tracking and SACK-based recovery to improve scalability. However, its reliance on NIC-resident bitmaps and reordering logic increases per-QP state, placing pressure on NIC memory and constraining overall connection density. SRNIC [6] addresses this by offloading retransmission and reordering to host software, while eliminating WQE caching to further simplify NIC design. Its hybrid approach assumes that packet loss is rare and delegates recovery to the software slow path. However, at ML scale, that assumption no longer holds: what appears as rare losses at the single-node level becomes frequent when viewed across thousands of GPU nodes synchronizing in parallel. These losses accumulate at synchronization points, exacerbated by slow workers, turning infrequent delays into persistent tail latency—a classic tail-at-scale effect [8].

This paper revisits the NIC transport design from a domain-specific perspective. We ask: *if ML can tolerate partial loss and reordering, why pay the cost of enforcing strict delivery guarantees at the RDMA NIC transport layer?*

To that goal, we make a case for Celeris, a hardware-accelerated RDMA transport tailored for ML workloads. Celeris removes retransmissions and in-order delivery, forwarding best-effort, unordered packets directly to application memory. It retains congestion control (e.g., via DCQCN), while delegating timeout handling and data prioritization to software. This architecture drastically simplifies NIC logic, reducing memory footprint and fault exposure. Rather than recovering from packet loss within the transport layer, Celeris bounds its impact and leverages recovery mechanisms within ML pipelines itself—such as Hadamard Transform [1].

By aligning transport semantics with ML workload characteristics, Celeris avoids the systemic buildup of tail latency from cluster-wide delays due to packet loss and stragglers. Preliminary results from our FPGA prototype and simulation-based evaluations show that Celeris reduces 99th-percentile latency by up to 2.3×, cuts BRAM usage by 67%, and nearly doubles hardware fault resilience. These early results suggest that rethinking transport guarantees through an ML-centric lens can unlock significant gains in performance, scalability, and resilience.

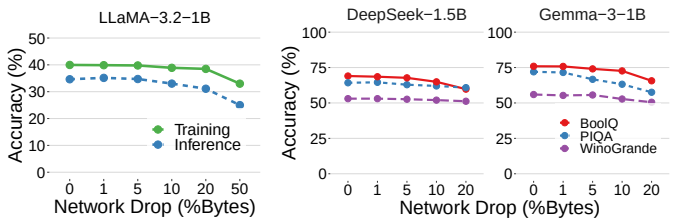
## II. Background and Motivation

### A. Communication Bottlenecks in ML Workloads

Modern ML workloads—both training and inference—depend heavily on collective communication across large clusters of accelerators. These collectives enable a variety of parallelism strategies: Data parallelism aggregates gradients using AllReduce. Tensor and pipeline parallelism, employed in both training and inference, exchange activations and weight shards using ReduceScatter/AllGather (and occasionally AllReduce) [9]. For long-sequence inference, context parallelism partitions the attention context and performs step-wise All-to-All exchanges (for K/V or token blocks) along with localized AllGather/ReduceScatter for residuals and projections.

| Design Aspect | RoCE | IRN [7] | SRNIC [6] | Celeris |
|---|---|---|---|---|
| PFC Required | Yes | No | No | No |
| Transport Reliability | Go-back-N | Selective Repeat | Selective Repeat (SW) | None (best-effort) |
| Packet Reordering | Dropped | Buffered in NIC | SW reordering | Offset-based placement |
| Congestion Control | Hardware | Hardware | Hardware | Hardware |
| WQE Cache | Present (HW) | Present (HW) | Eliminated | Eliminated |
| NIC State per QP | 407B | 596B | 242B | 52B |
| QP Scalability | 10K | 8K | 20K | 80K |
| Target Workloads | General RDMA | General RDMA | General RDMA + ML | ML Collectives |
| **Core Focus** | High performance | +Network efficient | +Connection scalable | +Tail optimal |

**TABLE I: Comparison of RDMA NIC designs: Celeris unifies performance, efficiency, and scalability from prior work, while adding tail-optimized support for loss-tolerant ML collectives.**




**(a)** Training and inference on the ARC dataset.

**(b)** Inference-only accuracy using multiple datasets (BoolQ, PIQA, and more).

**Fig. 1: Training and inference accuracy of all models remains stable under partial network drops ($\leq$ 5%).**

Expert parallelism (Mixture-of-Experts, MoE) uses All-to-All communication to dynamically route tokens between experts. And, hybrid parallelism blends these strategies to support increasingly complex workloads.

As model sizes and cluster scales grow, these collective operations increasingly dominate end-to-end performance [1], [5]. Because they synchronize large numbers of accelerators, even rare stragglers can stall an entire iteration, and their overhead grows rapidly with system scale. Yet the data they exchange—intermediate tensors, activations, routing metadata, and partial outputs—are often transient, later aggregated, recomputed, or subsampled downstream. This creates a fundamental mismatch: today's transports enforce strict in-order reliable delivery, while what matters most to ML workloads is minimizing collective tail latency. Empirical studies show that collectives can consume up to 70% of iteration time in large-scale systems [1], highlighting their role as the dominant communication bottleneck.

### *B. ML is Resilient to Loss*

Despite their sensitivity to tail latency, ML workloads are inherently robust to packet loss and partial delivery. Stochastic gradient descent (SGD) naturally smooths over noise and missing updates, and recent work has exploited this tolerance to reduce bandwidth via gradient sparsification, quantization, and bfloat16 compression [3], [4]. In-network aggregation systems, such as NVIDIA SHARP, perform lossy reductions in the dataplane without hurting convergence. And as shown in [1], [2], models maintain high accuracy even under substantial gradient drops during training communication (Figure 1a).

Importantly, this resilience is not limited to gradients. Intermediate tensors, activations, and routing metadata—often exchanged in collectives like AllGather and All-to-All—are frequently sparse, recomputable, or overwritten in later iterations. Even in inference [9], partial loss can be absorbed through redundancy, expert fallback paths, or statistical softmax smoothing (Figure 1b). This tolerance opens the door to a radically simpler transport: one that delivers timely data, even if it is occasionally incomplete.[1]

### *C. The Cost of Reliable Transports*

Mainstream RDMA transports like RoCE enforce strict delivery guarantees—go-back-N retransmissions, in-order delivery, and loss prevention via Priority Flow Control (PFC). These features ensure correctness but require deep integration of control state into the NIC: reorder queues, sequence number tables, retry logic, and recovery timers are tightly coupled with the datapath. While effective for storage and database workloads, this design scales poorly in large ML clusters, where collective synchronization amplifies even rare packet drops into full-step stalls.

Efforts like IRN [7] remove PFC and replace go-back-N with selective repeat and SACK-based recovery, improving network scalability. SRNIC [6] goes further by offloading reordering and retransmissions to software and removing the WQE cache. These designs reduce NIC memory usage (Table I), but still retain transport-layer recovery. As a result, they continue to trigger slow-path handling under loss—a mechanism that becomes problematic at scale. In large jobs, what appears rare at a single node becomes frequent in aggregate, creating persistent tail latency through tail-at-scale effects [8]. Meta has demonstrated limited success with RoCE in carefully tuned AI clusters [5], but only under tight (HPC-like) conditions—dedicated workloads, fixed topologies, and specialized routing—conditions infeasible in typical datacenter or cloud environments.

### *D. Reliability Hurts Resilience*

Beyond latency, the real cost of enforcing transport reliability is reduced system resilience. Per-QP state—retry counters, sequence numbers, recovery timers—is typically stored in on-chip SRAM. These structures are large, tightly coupled with critical datapath logic, and vulnerable to soft errors [10]. Even if each NIC reports a high Mean Time Between Failures (MTBF), e.g., 400,000 hours, at a 10,000-node scale, faults can occur every 40 hours. Worse, these faults often affect precisely the components responsible for enforcing correctness: a stuck retry timer or corrupted sequence number can silently stall a QP and block an entire collective [1], [2].

Our key observation is that ML workloads do not require these mechanisms to make progress or maintain correctness.

[1]Best-effort transport introduces nondeterminism, but this is acceptable in large-scale LLMs [1], [2] and can be mitigated through per-step logging of lost data to reproduce outcomes during debugging.

Instead of hardening unreliable machinery, we eliminate it. By removing retransmissions, reordering, and per-packet tracking entirely, Celeris (§III) reduces per-QP state to just 20 bytes. There are no retry counters, timers, or window logic—only minimal metadata required to push data. This not only improves tail latency but nearly doubles hardware fault tolerance, as shown in our FPGA prototype and MTBF analysis (§IV).

## III. Celeris: A Tail-Optimal RDMA NIC

Celeris is a domain-specific RDMA transport that removes traditional delivery guarantees to prioritize simplicity, scalability, and tail-optimal performance. Unlike prior work that implements loss-tolerant communication in software (e.g., OptiReduce [1] and MLT [2]), Celeris extends this approach through a co-designed hardware-software implementation. Instead of enforcing retransmissions or in-order delivery, Celeris delivers best-effort, unordered data directly to application memory—exploiting the statistical resilience of ML workloads (§II). It minimizes NIC state, bounds communication via software-driven timeouts, and shifts loss handling to the ML framework. This design reduces tail latency and hardware overhead, enabling efficient scaling across thousands of GPUs without compromising model convergence.

### A. Hardware – Stateless RDMA Transport

Celeris retains the foundational structure of conventional RDMA NICs: it supports QP-based communication, uses DMA to move data from GPU memory, and implements flow-level congestion control (e.g., DCQCN). However, it departs from prior designs like RoCE and IRN [7] by eliminating all packet-level reliability mechanisms. There are no retransmissions, reorder queues, selective repeat buffers, or outstanding request tables in the NIC.

Instead, each packet includes a logical offset into the target buffer, allowing direct placement without requiring reassembly or sequencing [6], [7]. This enables unordered delivery: packets are placed as they arrive, with no NIC-side tracking of order or completion. The NIC is unaware of loss or duplication—it simply forwards data.

With no per-packet state or bookkeeping, the per-QP context is dramatically reduced—only 20 bytes (plus 32 bytes in case of DCQCN), compared to hundreds of bytes in RoCE, IRN, and SRNIC (Table I). This compact context allows the NIC to support 10× more connections using the same SRAM budget, enhancing scalability across large ML clusters. The only active datapath logic is congestion control, which remains in hardware to ensure fair bandwidth allocation.[2]

Celeris removes all reliability mechanisms from the NIC, and simplifies the datapath to a streamlined push engine focused purely on data movement. This reduces BRAM usage, eliminates fault-prone state, and enables predictable, low-latency communication at scale. The design aligns with a core insight of Celeris: if ML workloads can tolerate loss, enforcing strict delivery guarantees in hardware is not only unnecessary, it is counterproductive.

[2]A software variant atop RDMA's UC and UD is possible, but would shift congestion control to software, increasing CPU overhead and sacrificing Celeris's hardware simplicity.

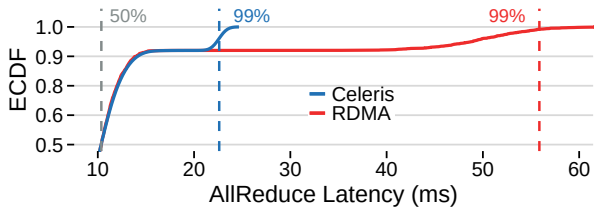



**Fig. 2: AllReduce step times under background contention. Celeris bounds tail latency with per-step timeouts.**

### B. Software – Bounded Timeouts & Cluster Coordination

In Celeris, the software stack assumes responsibility for progress and coordination, replacing NIC-managed reliability with bounded delivery windows and lightweight recovery strategies. Each collective operation begins with the sender issuing WQEs to transmit data. The NIC streams packets toward receivers without tracking their delivery or acknowledgment.

On the receiver side, software defines a step-level timeout that bounds the delivery window for each collective operation. Packets arriving after this window are discarded. At the end of the timeout, the collective step is finalized using only the data that arrived on time. Because distributed training typically involves multiple concurrent collectives, such as data, tensor, or expert-parallel groups, the software maintains an independent timeout profile for each.

Timeouts are dynamically adjusted. After each step, the system measures how much data was received and how long the step took. If all data arrived, the next timeout is updated to match the observed duration. If only partial data was received, the system estimates the required duration for full delivery and sets the next timeout accordingly [1]. These updates are smoothed using exponential averaging and bounded within a fixed range to ensure stable behavior.

To maintain cluster-wide coordination, nodes share their local timeout estimates at the end of each step. All nodes then use the median of the reported values for the next round. This synchronization prevents stragglers from dominating step duration and ensures consistent progress across the cluster, even under packet loss or transient congestion.

Since Celeris omits transport-layer recovery, it relies on the ML model's inherent tolerance to partial data. Critical information (like activation shards) can be prioritized and split across packets for partial recovery, with lightweight coding schemes (e.g., XOR, Hadamard) used to reconstruct lost fragments—mitigating the impact on model accuracy.

By shifting transport semantics to match ML's needs, Celeris enables scalable communication that is robust to loss, simple to implement, and optimized for tail latency.

## IV. Preliminary Results

We evaluate Celeris to validate our core claim: simplifying the RDMA transport layer for ML workloads improves tail latency, reduces hardware overhead, and enhances system resilience. Our evaluation spans three axes—latency, resource efficiency, and fault tolerance—using both FPGA prototypes and cluster-scale simulation.

*Evaluation Setup.* We implement Celeris on an AMD Alveo U250 FPGA using Coyote, an open-source RoCE-compatible

NIC shell. To support RoCE, IRN [7], and SRNIC [6] baselines, we extend Coyote's QP context to 407B, 596B, and 210B, respectively (Table I). In contrast, Celeris uses a small 52B per-QP by eliminating retransmissions, in-order delivery, and packet tracking. Packets carry an explicit offset, enabling direct placement without reorder buffers.

For cluster-scale simulation, we integrate Celeris with AstraSim and NS-3, modeling a 128-node Clos network. Each node performs 25MB rounds while randomized, bursty background traffic is injected to create contention.

### A. Performance: Reduction in Tail Latency

We evaluate tail latency under contention using the above 128-node AllReduce setup. The baseline uses a RoCE-style RDMA stack with retransmissions and in-order delivery. As shown in Figure 2, it suffers from high tail latency, with the 99th percentile exceeding 5× the median due to retransmission delays and head-of-line blocking.

Celeris mitigates this by eliminating transport-layer recovery and instead applying a software-managed timeout per collective step. Once the timeout expires, each node finalizes the round using data received up to that point, discarding any late packets. We set the timeout to the *median plus one standard deviation* of the baseline distribution. Figure 2 shows that Celeris cuts the 99th-percentile latency by 2.3× while preserving median latency. Even without retransmissions, less than 1% of total data is lost, as most nodes complete transmission before the timeout. As discussed in §II, this level of loss is well within ML's convergence tolerance.

### B. Resource Efficiency: Reduced NIC Logic and Memory

Celeris reduces hardware complexity by eliminating stateful transport-layer components such as retransmission queues, sequence tracking, and reorder logic. These modules consume significant logic and memory in traditional RDMA designs, especially when supporting thousands of QPs. We evaluate this by synthesizing Celeris and three baselines—RoCE, IRN, and SRNIC—using Vivado 2022.1 on an AMD Alveo U250 FPGA with 10K QPs. As shown in Table II, Celeris reduces LUTs by up to 6.6%, LUTRAMs by 10.2%, and FFs by 5.2%. BRAM usage drops by 63.5–72.7% compared to RoCE and IRN, due to the elimination of bitmaps, per-QP window state, and reassembly buffers. Power also improves by up to 9%, reflecting reduced switching activity across datapath FSMs.

To assess ASIC feasibility, we apply standard FPGA-to-ASIC scaling models for 7nm TSMC technology. Celeris has the smallest area, using approximately 57% less silicon than IRN and about 28% less than SRNIC. This compact footprint improves integration density and lowers thermal and validation complexity. By reducing the NIC to its essential functions—DMA, header parsing, and congestion control—Celeris not only improves performance but also delivers a streamlined, scalable transport engine for constrained environments.

### C. Resilience: Improved Hardware Fault Tolerance

Transport-layer reliability features such as retry engines, sequence tracking, and reorder queues introduce not only complexity but also vulnerability. These mechanisms rely on SRAM-backed per-QP state and tightly coupled control logic, which are known sources of soft errors in high-density FPGA and NIC deployments. To quantify this, we use the Xilinx SEU Estimator [10] to model Mean Time Between Failures (MTBF) for a 15,000-node datacenter operating at 100°C. We assume a 10% CRAM essential bit ratio and a standard SRAM FIT rate of $10^{-11}$ per bit [10]. Under this model, RoCE and IRN—each maintaining hundreds of bytes of control state per QP—show MTBFs of 42.8 and 34.3 hours, respectively (Table II). SRNIC improves this to 57.8 hours by offloading some recovery logic to software. In contrast, Celeris, with just 52B of per-QP state and no transport-layer recovery, achieves an MTBF of 80.5 hours (nearly 2× improvement).

| Metric | RoCE | IRN | SRNIC | Celeris |
|---|---|---|---|---|
| LUT | 312,449 | 319,567 | 304,497 | 298,435 |
| LUTRAM | 23,277 | 24,221 | 22,460 | 21,743 |
| FF | 562,129 | 573,116 | 551,526 | 542,972 |
| BRAM | 1450.5 | 1941.5 | 939.5 | 529.5 |
| Power (W) | 34.7 | 35.9 | 33.5 | 32.5 |
| MTBF (hrs) | 42.8 | 34.3 | 57.8 | 80.5 |

**TABLE II: Celeris delivers the best balance of area, power, and reliability across FPGA/ASIC metrics.**

## V. Conclusion

Celeris rethinks RDMA transport for ML workloads by eliminating delivery guarantees in favor of tail-optimal, loss-tolerant performance. By simplifying the datapath and aligning with ML's resilience to loss and partial data, it offers a scalable, efficient, and more robust foundation for collective communication at the cluster scale. Our early prototype reduces 99th-percentile latency by 2.3×, cuts BRAM usage by over 70%, and nearly doubles hardware MTBF. These gains come not from additional complexity, but from removing complex reliability mechanisms that ML does not need. Celeris shows that ML-aware transport can deliver both simplicity and performance at scale.

## References


[1] E. Warraich, O. Shabtai, K. Manaa, S. Vargaftik, Y. Piasetzky, M. Kadosh, L. Suresh, and M. Shahbaz, "OptiReduce: Resilient and Tail-Optimal AllReduce for Distributed Deep Learning in the Cloud," in *USENIX NSDI*, 2025.

[2] H. Wang, H. Tian, J. Chen, X. Wan, J. Xia, G. Zeng, W. Bai, J. Jiang, Y. Wang, and K. Chen, "Towards Domain-Specific Network Transport for Distributed DNN Training," in *USENIX NSDI*, 2024.

[3] J. Fei, C.-Y. Ho, A. N. Sahu, M. Canini, and A. Sapio, "Efficient Sparse Collective Communication and its Application to Accelerate Distributed Deep Learning," in *ACM SIGCOMM*, 2021.

[4] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding," *NeurIPS*, 2017.

[5] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang, S. Zhang, M. J. Fernandez, S. Gandham, and H. Zeng, "RDMA over Ethernet for Distributed Training at Meta Scale," in *ACM SIGCOMM*, 2024.

[6] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng, T. Wang, W. Ling, K. Huo, P. An, K. Ji, S. Zhang, B. Xu, R. Feng, T. Ding, K. Chen, and C. Guo, "SRNIC: A Scalable Architecture for RDMA NICs," in *USENIX NSDI*, 2023.

[7] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting Network Support for RDMA," in *ACM SIGCOMM*, 2018.

[8] J. Dean and L. A. Barroso, "The Tail at Scale," *CACM*, 2013.

[9] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "DeepSpeed-MoE: Advancing MoE Inference and Training to Power Next-Generation AI Scale," in *PMLR*, 2022.

[10] "AMD's Soft Error Mitigation," https://docs.amd.com/r/en-US/pg187-ultrascale-sem/Understanding-the-Soft-Error-Mitigation-Requirement, viewed: 5/2025.