Resilient AI Supercomputer Networking using MRC and SRv6

OpenAI	Microsoft	AMD	Broadcom	NVidia
Joao Araujo	*S. Anantharamu*	*K. Doddapaneni*	*Eric Davis*	*Aviv Barnea*
Alex Chow	*Abhishek Dosi*	*Murali Garimella*	*Dragos Dumitrescu*	*Idan Burstein*
*Mark Handley**	*Mohit Garg*	*Vipin Jain*	*Mohan Kalkunte*	*Elazar Cohen*
Ryder Lewis	*Mahdieh Ghazi*	*Yanfang Le*	*Bhaswar Mitra*	*Yamin Friedman*
Christoph Paasch	*Torsten Hoefler*	*H. Nagulapalli*	*Guglielmo Morandin*	*Noam Katz*
Jitendra Padhye	*Deepal Jayasinghe*	*S. Narayanan*	*Adrian Popa*	*Masoud Moshref*
Michael Papamichael	*Jithin Jose**	*Rong Pan*	*Costin Raiciu*	*Yuval Shpigelman*
Greg Steinbrecher	*Abdul Kabbani*	*Rathina Sabesan*	*Eric Spada**	*Shahaf Shuler*
Amin Tootoonchian	*Guohan Lu*	*Raghava Sivaramu*	*John Spillane*	*Shy Shyman*
Lihua Yuan	*Yang Wang*	*Rip Sohan**	*Niranjan Vaidya*	*Sayantan Sur**

Abstract

Tail latency dominates the performance of synchronous pretraining jobs when running at very large scales. We describe a three-pronged approach: (1) a new RDMA-based transport protocol, MRC, sprays across many paths and actively load-balances between them, eliminating the issue of flow collisions (2) the use of multi-plane Clos topologies to get the benefits of high switch radix and redundancy, allowing training clusters well over 100K GPUs to be built as two-tier topologies while increasing physical redundancy, and (3) the use of static source-routing using SRv6 to allow MRC the freedom to bypass failures by itself. We describe our experiences running MRC and static SRv6 routing in production in OpenAI and Microsoft's largest training clusters, where it has been used to train the latest frontier models. We demonstrate how MRC allows AI training jobs to ride out many network failures that previously would have interrupted training.

1 Introduction

As networks for AI training scale into the hundreds of thousands of GPUs it becomes increasingly difficult to achieve acceptable uptime and performance for the largest training jobs. This is especially the case for synchronous pretraining, where each step of computation is performed in lock-step by large numbers of GPUs, interspersed with communication between nodes to perform a combination of pipeline parallelism, data parallelism, tensor parallelism and expert parallelism [4, 36, 44]. The goal is to overlap communication and computation to the maximum extent possible, so that the critical path is balanced and computation-dominated. Achieving this at scale is challenging, as the duration of each communications round is determined by the slowest transfer.

[1]Corresponding authors: mjh@openai.com, jijos@microsoft.com, rip.sohan@amd.com, eric.spada@broadcom.com, ssur@nvidia.com

As computations scale, communication becomes increasingly outlier-dominated [11] a phenomenon long known in the HPC community as system or network "noise" [22,32]. To make matters worse, network failures become more prevalent with scale; when they cause job failues they become expensive in lost GPU time. How then can we maintain acceptable training performance with jobs that require large numbers of GPUs (up to 100,000 or more) to perform a synchronous computation? Broadly, any solution needs to do three things:

- Load balance the network evenly, so as to prevent congestion due to flow collisions;
- Handle incast-based congestion without creating outliers;
- Handle link and fabric failures gracefully, without bringing down the training job.

To add to the problem, very small teams of people need to be able to manage the networks of multiple supercomputers each with many thousands of switches running multiple simultaneous training jobs with their own unique traffic patterns. We cannot rely on human intervention to diagnose and fix individual network failures in a timely manner, so the protocol stack needs to be failure tolerant by design and the network itself should have a very simple control plane, which requires almost no active management. Failed links or misbehaving switches need to be bypassed automatically. Failed nodes however, can already easily be removed from the running jobs, without requiring global coordination.

To address these issues in very large training clusters we designed, built and deployed Multipath RC (MRC), which extends the RoCE [26] Reliable Connection (RC) semantic layer and draws upon lessons from Ultra Ethernet Transport (UET) [10, 23]. Like UET, MRC employs packet spraying, adaptive load balancing based on ECN, out-of-order memory placement of received data, selective retransmission, and uses packet trimming to mitigate incast. Unlike UET, MRC is a

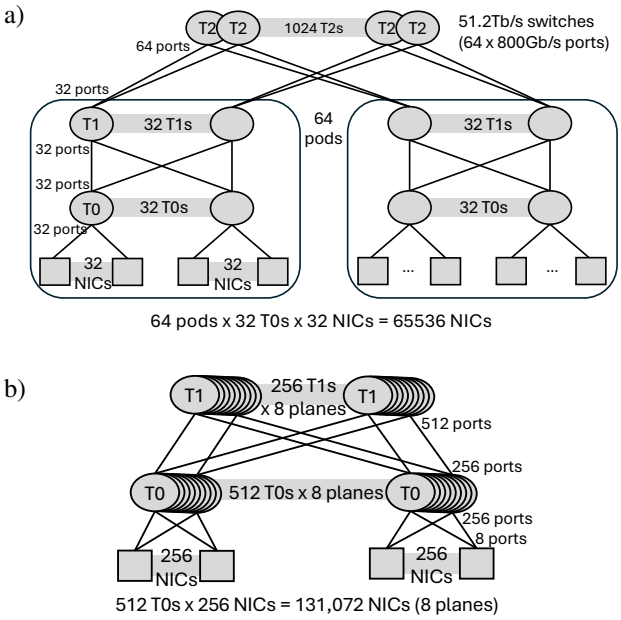


Figure 1: (a) 3-Tier 800 Gb/s single-plane topology vs (b) 2-Tier 8x100 Gb/s multi-plane topology

minimal extension to RoCE; MRC leverages and extends the existing Verbs API but our AI workloads only require a subset of the functionality so, at the transport level, only the RDMA write and write-with-immediate operations are supported.

Knowing that we would deploy MRC allowed us to co-design the topology of training clusters with very high resilience in mind. Further, MRC's adaptive load balancing is extremely good at routing around failures by itself. We took the unusual position of disabling dynamic routing in the switches because we didn't want two adaptive routing mechanisms interacting with each other and dynamic routing wasn't adding anything. Instead data packets are source-routed along static paths using IPv6 segment routing (SRv6). Using static routing seems at first glance to be the opposite of what we wish to achieve, but we will describe how this combination leads to a highly resilient high performance training cluster at the largest scale with a low operational burden.

In this paper we describe our experiences designing, deploying and operating MRC at OpenAI and Microsoft. We implemented MRC in 400 and 800Gb/s RDMA NICs: NVIDIA ConnectX-8, AMD Pollara and Vulcano, and Broadcom Thor Ultra. We also implemented support for SRv6 in NVIDIA Spectrum-4 and 5 switches running both Cumulus and SONiC, and collaborated with Arista to implement it in EOS on Broadcom Tomahawk 5 switches. MRC is in large-scale production use in multiple very large AI training clusters, where it has been used to train frontier large language models (LLMs) for ChatGPT and Codex. We have released the MRC specification [41] under through OCP under an open license for anyone to use.

2 Multi-plane Topology Co-Design

Consider a hypothetical cluster with 100,000 GPUs, each paired with an 800Gb/s NIC. We wish to achieve full bisection bandwidth to simplify workload placement. One option is a conventional three-tier Clos topology using today's fastest Ethernet switches (Fig. 1a). Currently datacenter-class switches can achieve 51.2Tb/s giving 64 ports at 800Gb/s. Each Tier-0 (T0) switch connects down to 32 NICs and connects up to 32 Tier-1 (T1) switches, giving a pod size of 1024 NICs. Each T2 switch connects to 64 different pods giving a cluster size of 64K NICs. If we wish to connect 100K GPUs we either need to use four tiers of switches, oversubscribe the network, or build multiple independent rails.

Alternatively, we can break out the 800Gb/s NIC by lane and use it as 8 x 100Gb/s ports (Fig. 1b). We build eight parallel 100Gb/s Clos planes using the same 51.2Tb/s switches, but now each switch has 512 ports. Each T0 switch connects down to 256 NIC ports and connects up to 256 T1 switches. Each T1 switch connects in turn down to 512 T0 switches, giving a network of 131,072 GPUs. In this arrangement we can easily accommodate our 100K GPUs using *only two tiers of switches*. Such a multi-plane network has many advantages:

- Latency is lower because the longest path traverses only three switches rather than five or seven.
- Many more nodes are reachable in one hop (256 rather than 32), so it is easier to take advantage of locality in the job, reducing latency and reducing load on T0 uplinks.
- Cost and power consumption are reduced - for full bisection bandwidth we require $^2/_3$ of the optics and $^3/_5$ the number of switches compared to a 3-tier network.
- The impact of an in-network failure is much less. For example, losing a T0-T1 link reduces capacity from a node by 3% in an 800Gb/s plane, vs 0.4% in a 100Gb/s plane.
- It is possible to lose a NIC-T0 link without bringing down the training job. We still lose 12% of the NIC bandwidth, but can easily ride out a link flap with the remaining capacity.

It seems clear, therefore, that a multi-plane design has many advantages over a single-plane design, but there are some challenges too. First, the workload needs to be able to survive link and NIC port failures. Second, to fully use the network we need to be able to load all network planes equally and load balance across all the many paths within a plane without suffering a loss of performance due to flow collisions. This is hard to do with traditional single-path transport protocols [2] and is made harder by using lower speed network links which are easier to overload. This is where MRC comes in.

2.1 MRC Overview

MRC extends the RoCEv2 Reliable Connection (RC) transport protocol to support multi-path operation borrowing several features of UET [10, 23]. It supports the normal RoCE verbs interface and Queue Pair (QP) abstraction, but only for write and write-with-immediate operations. At a protocol level, the main features MRC adds are the following:

- Every data packet contains the RDMA virtual address and remote key so the receiving NIC can write each arriving packet to memory immediately, no matter the arrival order.
- Each packet contains an entropy value (EV) that dictates its path through the network. The 32-bit EV is striped across the UDP source port and IPv6 flow label in an MRC packet. In a conventional network, changing the EV causes switches to hash each packet to a different path from the ECMP set. At QP startup, the sender generates an EV set for that QP—typically 128 to 256 entries. The sender then rotates through this set, using a different EV for each packet, so that all packets of a QP are sprayed across many paths on all planes in a multi-plane network without the application needing to know. This serves to load balance the network.
- Spraying is hard to combine with the priority flow control (PFC) mechanism used in lossless Ethernet because a single flow reaches the last-hop switch over hundreds of paths. Further, PFC tends to create head-of-line blocking between different collectives, hurting tail latency. Thus MRC disables PFC and uses Ethernet in best-effort (lossy) mode.
- The combination of best-effort Ethernet and out-of-order delivery places a greater burden on recovering losses quickly. MRC implements fast selective retransmission, using Selective ACK (SACK) packets to indicate precisely which packets have arrived at the receiver.
- To further increase retransmission speed, especially under incast, MRC can use packet trimming [10, 20]. With packet trimming, a packet that would have been dropped due to congestion has its payload trimmed off and is priority-forwarded to the destination. The receiving NIC then generates a NACK to trigger fast retransmission. This also lets MRC distinguish congestion loss from other packet loss, which in AI clusters is mostly due to link flaps and failures.

A protocol like MRC, designed around packet spraying, is a very good fit for a multi-plane network. Each EV corresponds to a specific path on a specific network plane. When MRC generates its EV set, it chooses an equal number of EVs per plane. This immediately equalizes the traffic between planes.

For each EV, MRC keeps a few bits of state about path health. In each switch, we enable Explicit Congestion Notification (ECN) in the normal randomized manner, but disable ECN on the last hop to the receiver. In a network with full bisection bandwidth, the traffic aggregate should not experience congestion, except from incast on the last hop, so ECN now acts as a load-balancing signal. The receiver echoes the ECN signal back to the sender, indicating that this specific path is more congested than others, and the sender temporarily avoids it. Different MRC senders do not coordinate when choosing their EV sets, so even though each sender load balances well, the aggregate may be slightly uneven. ECN-based load balancing smooths out this unevenness, keeping internal queues from growing enough to cause congestive loss.

When a packet is not trimmed but actually lost, MRC assumes the path has failed and immediately stops using the corresponding EV. Of course, not all loss is due to failed paths - packets can suffer bit errors or other issues - so permanently retiring an EV after one lost packet may leave us short of working EVs. To avoid this, MRC sends background path probes to determine whether paths it assumed were bad are actually bad, and also detect if failed links have recovered. If enough probes succeed, the EV is resurrected.

At this point we have a transport protocol that can detect path failures and bypass them in a few tens of microseconds.

2.2 Static Segment Routing

In a conventional datacenter network we would use a dynamic routing protocol such as BGP to determine reachability and route around failed links. Generally this works, though convergence can take a very large number of RTTs. Unfortunately, high-radix two-tier topologies make this worse. Every destination corresponds to a large ECMP set - up to 256 entries in a 512-port T0 switch, one per uplink. This is not a problem without failures, but most T1 switches may have at least one failed downlink. As a result a specific destination is unreachable via some T1 switches, so a T0 switch cannot use the default ECMP set balanced across *all* uplinks to reach it. The number of large ECMP sets required in each T0 switch grows close to the total number of T0 switches. Maintaining so many large ECMP sets can be challenging for dynamic routing and for switch forwarding engines to support. Diagnosing routing problems at scale is notoriously difficult.

Given MRC will rapidly avoid failed paths and that we build redundant topologies, do we still need dynamic routing? In fact, combining end-system load balancing with switch-based dynamic routing makes network behavior harder to understand than either running alone. MRC reacts to a failure first, avoiding the broken path; then dynamic routing re-routes, changing ECMP mappings and disturbing load balancing.

We took the position that, when running MRC, dynamic routing caused more problems than it solved, so we simply disabled it, but we still need a way to map EVs to specific network paths. One option would be to run MRC with statically configured ECMP routes, relying on MRC to avoid bad paths. The problem is that there still is not a simple mapping from EV to path. For example, it would be hard to correlate MRC's

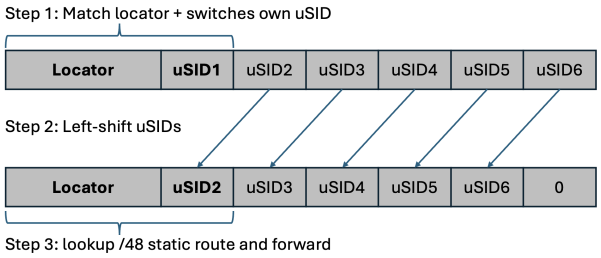


Figure 2: SRv6 forwarding using uN uSIDs

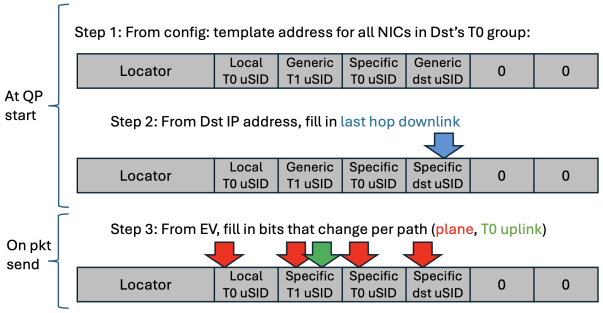


Figure 3: Creating the SRv6 address from an EV and template

EV-based view of bad paths with the precise physical path, so we can report failures for repair. This led us to spray using source routing, as prior work has suggested [20].

The approach we chose was to deploy IPv6 segment routing (SRv6) [15]. In the MRC NIC, at QP startup a set of entropy values (EVs) are chosen, such that bits in each EV *directly embed* the path choice available at each hop in the network. Each EV then maps to a specific unique path through the network to the destination NIC.

SRv6 is a family of solutions; we use the micro-segment ID (uSID) format [9], where the destination IPv6 address consists of a 32-bit locator prefix followed by a sequence of 16-bit uSIDs each corresponding to a specific switch along the path. We use the uN style of uSID, where each switch along the path is explicitly named. We describe the relation between EVs and SRv6 addressing in detail in Section 2.3.

The SRv6 forwarding process is shown in Fig. 2. When a packet arrives, the switch compares the first 48 bits of the destination address with the switch's configured SRv6 locator and uSID; if they match this is a valid SRv6 packet for the switch to process. The switch then left-shifts the uSID part of the address by 16 bits, moving the next-hop uSID into the first 48 bits. This new address is then looked up in the switch's static forwarding table in the normal way. This forwarding table was configured when the switch was installed, and is generally never changed. The static route dictates which egress port the packet uses. In all switches we deploy, this uN style of SRv6 forwarding can be performed at line rate.

MRC packets are IPv6 in IPv6 encapsulated, with the outer destination address being the SRv6 path and the inner destination address containing the destination NIC's own address, so that the receiving NIC can recognize and decapsulate the packet before feeding it to the MRC RDMA pipeline.

2.3 Mapping EVs to SRv6 Addresses

MRC was designed to work with either hash-based ECMP forwarding or SRv6. The EV is embedded in each packet, striped across the UDP source port and the IPv6 flow label, both of which are hashed by switches performing ECMP forwarding. The EV is also echoed in SACK and NACK packets to indicate the congestion state on a path.

When using SRv6, although the EV is not hashed by switches, it still needs to be carried in data packets so the receiver can echo it. The SRv6 address itself cannot be echoed, as it is erased by the shifting process during forwarding.

To avoid holding both an SRv6 address and EV state per path, we use an algorithmic mapping between each EV and its corresponding SRv6 address. Switch uSIDs are allocated according to the network structure, allowing the EV value to be a compressed representation of the bits that vary between SRv6 paths to that destination, plus the NIC port to be used.

Fig. 3 illustrates how SRv6 destination addresses are created for a QP traversing src→T0→T1→T0→dst. On QP startup, the NIC looks up the destination address prefix in a node-specific configuration file to obtain a generic SRv6 address template for nodes in that row. The dst uSID in this template is specialized for this destination by copying in the last-hop downlink number. This template is used by all packets sent by the QP.

The NIC also creates a set of EVs for this QP, striping across planes and paths within each plane. Each time a packet is sent, a new EV is selected from the active set. The template is then further specialized to create the final destination address by copying the plane number from the EV into all uSIDs and the T0 uplink number into the T1 uSID. We also use an extension of this scheme to forward between clusters.

2.4 Choosing Working Paths

With a conventional protocol, resource management is split between routing and transport: routing delivers working paths to transport, and transport manages congestion on those paths. In a statically source-routed MRC network, all resource management and failure handling is the role of MRC.

On QP startup, MRC must select a set of EVs, each mapping to a unique path on a specific plane. MRC chooses EVs equally split across the planes, then randomly selects a subset of paths within each plane. Different senders in the same T0 group do not coordinate their choices.

At pretraining job startup we ensure all nodes in the job have all NIC ports operational. MRC supports a denylist, allowing us to avoid paths that traverse links known to have failed. To enable this, we implemented Clustermapper con-

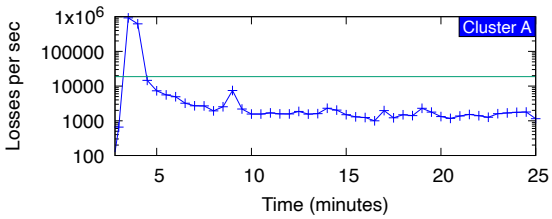


Figure 4: Startup losses without mapping out bad paths

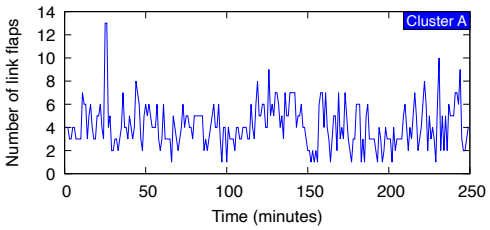


Figure 5: Link flaps between T0 and T1

sisting of an agent on every node; together these map which links are currently down or have excessive loss. Static SRv6 routing makes this very simple: unlike with ECMP hashing, we know exactly which path a Clustermapper probe packet will take, and know it is the same path an equivalent MRC packet will take. Unlike switch-based telemetry, the resulting map gives ground truth about forwarding-plane health.

In fact, for pretraining it has not proved necessary to use Clustermapper to set denylists for failed T0–T1 links. QPs are very long lived and MRC quickly routes around failures and settles on good paths. On CX8 MRC each QP starts by populating a large EV set, typically over 100 entries, plus a backup EV set for failures. The QP starts spraying packets across the corresponding paths. Some paths are down; packets are lost and retransmitted, and the EV is swapped with one from the backup set. Even without prior knowledge, MRC maps out bad paths quickly enough that the slight startup performance loss is not a problem.

Fig. 4 shows the packet loss rate as a 75K GPU pretraining job starts without pre-populating the denylist. The figure shows the total losses per second on one of the four NICs per node, summed across all nodes in the job. The graph uses a logscale to make the background rate visible. The horizontal line shows one loss per second per NIC - at 800Gb/s this would be a loss rate of 1 packet in 25 million. The loss rate across the whole job falls well below this point within a couple of minutes, but even in the first minute less than 5 packets per QP are lost. Given that we have to ramp up large training jobs slowly to avoid destabilizing the power grid, this startup transient has minimal impact on training time.

Reverse Paths: MRC sends RoCE cumulative ACKs, MRC SACKs and MRC NACKs on the reverse path. The EV set on the forward path is updated based on SACK and NACK information, but what EV should reverse path packets use? MRC isn't especially sensitive to reverse-path loss, but it does have some impact on tail latency.

With bidirectional traffic, control packets could use any EV from that endpoint's active EV set, but many collective QPs are unidirectional at any instant. We could reverse the SRv6 forward path, but unfortunately this won't always work.

One solution we have found to work well is to keep a small reverse EV set for control packets, with at least one EV per plane. Each RTT that the QP is active inbound but generates no outbound traffic, the receiver sends an EV probe packet using a randomly chosen EV. If the probe is acknowledged, the reverse EV for that plane is set to the probe's EV. If data traffic is sent, the reverse EV is updated from data SACKs instead of needing EV probes. Thus the reverse EV set always contains EVs known to be up and working.

3 Operations

A goal of MRC was to simplify network operations, so that very large supercomputer networks can be operated by small teams. Key to this is that with MRC, most network failures do not require quick reactions from the network operators.

Our experience has been that link failures and flapping links between T0 and T1 switches can largely be ignored. We still schedule failed links for repair, but at lower priority. Figure 5 shows the total number of link flaps per minute between T0 and T1, as reported by the switches, while running a very large synchronous pretraining job on Cluster A (Table 1). These links are left in service because they simply do not cause a problem. MRC spreads traffic across enough paths that when a link used by a QP flaps, only a very small number of packets per QP are lost, often just one, and the corresponding EV is removed. The missing packet is selectively retransmitted on a different path, and the impact on the job is negligible.

Prior to MRC, when a flappy link was banned for repair, it would be administratively taken down so data traffic could not use it. Only once it was repaired and tested would it be brought back up. Initially we took this approach with MRC too, but it became clear it was unnecessary. Now we leave links in service while they are being repaired: MRC maps them out when they drop, and only brings them back when enough probes succeed over time. This requires less coordination and ensures links are available whenever they actually work, but not used when too unreliable. It is also robust to the inevitable disruption caused when a technician repairing one link disturbs neighboring links, causing them to flap.

Switch software is complex and so prone to bugs [28,39,45] which can hinder fault detection and sometimes cause the control plane to fail to route around failures. According to a datacenter study covering 180,000 switches over three months [39], 17% of switch failures are due to software bugs. Such bugs can cause control-plane and dataplane divergence, where the control plane fails to accurately detect dataplane failures [28, 39]. These studies match our own experience: we have

Table 1: Experiment Platform Configurations

Cluster	NIC	Switch	Topology
Cluster A	NVIDIA GB200 + CX8 (800 Gbps)	NVIDIA SP4 & BRCM TH5	2-Tier 4 x 200 Gbps Multi-plane
Cluster B	NVIDIA GB200 + CX8 (800 Gbps)	NVIDIA Spectrum 5	2-Tier 8 x 100 Gbps Multi-plane
Cluster C	AMD MI355 + Pollara (400 Gbps)	Broadcom Tomahawk 5	2-Tier 4 x 100 Gbps Multi-plane
Cluster D	NVIDIA RTX 6000 + Broadcom Thor Ultra	Broadcom Tomahawk 5	2-Tier 400 Gbps Single-plane

seen a wide range of bugs, including cases where the control plane and links are up, but the switch stops forwarding packets. This last case is particularly pernicious.

When using static SRv6, MRC doesn't care about the state of the control-plane: if packets don't flow, it removes the path. When we see a T1 switch that is not happy, we simply reboot it without concern for routing convergence or needing to coordinate with active training jobs. MRC maps out EVs that traverse the failed switch and restores them afterwards, with negligible effect on job performance. We do have to be more careful with NIC-T0 links and T0 switches, as these do have a measurable effect. A failed NIC link will not cause a QP failure; rather the NIC detects the link has dropped and MRC remaps EVs to avoid the failed port. Many packets are lost right after the link drops. Remapping all the EVs from the many QPs in use is not instantaneous in the CX8 MRC implementation, so this causes a glitch in job throughput. MRC then uses a port state bitmap in SACK packets to notify remote QP endpoints that the port is down, so they also remap their EVs to avoid the failed plane. Once this has happened, which typically takes a few seconds, the QPs are back to being fully functional, but using one fewer plane.

We have deployed both four-plane (4 x 200Gb/s) and eight-plane (8 x 100Gb/s) MRC supercomputers. The impact of a NIC port failure is obviously smaller with eight-planes, but even with four planes the effect on training job performance can be relatively small – the precise amount depends on job layout, but can be significantly less than the lost capacity. We detect the failure and allow the job to continue with reduced performance. Most of the time the port recovers relatively quickly with no lasting impact. When a port fails down and stays down, we ban the node and report it for repair.

Not requiring control-plane action for a training job to ride out failures is only part of the operations story. We still need good telemetry to root-cause failures, tune job performance, and of course debug MRC itself.

Clustermapper is key here. The Clustermapper agents running on all cluster nodes together probe every link in the network every millisecond. This gives us fine-grained health data, allowing us to schedule links for repair or switches for rebooting. A Clustermapper agent on each node probes each of the 16 or 32 directly connected T0s (one per port, on each of the four NICs per node) by sending probes source-routed to the T0 and back to the same agent. This identifies issues with NIC–T0 links or T0 switches. Each agent also probes a subset of the T1 switches, again source-routing to the T1 and back, so that all T0–T1 links are probed at high frequency. The combination of T0 and T1 self-probes lets us immediately localize precisely which link or switch has a problem: if T0 probes show no problem but T1 probes do, we know the issue is the T0–T1 link, not the NIC–T0 link. We could run this probing on demand, as soon as MRC reports a path problem, but in practice we found it useful for Clustermapper to detect issues even when no workload is running, and the cost of continuous probes is low.

Without SRv6, it is harder to get this level of ground truth on network health. It is much easier to interpret data from probes sent from an agent through the network back to itself, rather than mechanisms like pingmesh [19], which must send to a remote node that may not be up. Unlike with ECMP hashing, there is no ambiguity about which path a probe packet takes, there is no dynamic routing changing forwarding paths underneath, and we know MRC data packets take the same paths. While directly pinging switches is possible without SRv6, ICMP probes are handled by the control plane, which limits probing frequency. With SRv6, the switch treats a probe like any other data traffic, handling it in the dataplane and enabling high-frequency probing.

4 Inter-plane Loading

We took two decisions to simplify MRC behavior that have consequences. First, when MRC maps an EV out of its active set, it replaces it with one from the same plane. This ensures load balancing remains exactly equal between planes when all NICs have all planes available. We made this choice to avoid false incast at the destination, where mild congestion on different flows' T0 uplinks can cause them to load planes unevenly, making some planes more congested than others when flows converge at a destination.

Keeping the active EV set equal between planes means two cases are not handled well by default. First, if any single-path traffic is present on the back-end MRC network, MRC will be constrained by the most congested plane and lose capacity. This is obviously not a problem if the back-end network only runs MRC. A consequence is that if we lost sufficiently many T0–T1 links in a single plane, that plane would become the bottleneck and we would lose performance. In practice, we have not seen this in production.

Second, and partly as a consequence of keeping planes evenly loaded, if a NIC–T0 link does not fail completely but instead has an unacceptably high packet loss rate, MRC cannot rebalance to avoid the bad plane because it cannot reliably tell whether the problem is at its end or the remote end.

Clustermapper can, because it probes to the local T0 and back. Thus we delegate detecting this case to a Clustermapper policy decision; the bad plane can then be avoided by specifying a matching denylist entry.

In practice, keeping all planes evenly loaded is a very useful invariant. All planes normally look the same in network stats once MRC has avoided bad links, so if one plane looks worse than the others, it generally points to a network problem.

5 Experiments

We present results from four different AI Training clusters; the cluster configuration and topology details are presented in Table 1. All the clusters follow the 2-Tier multi-plane topology as described in Figure 1b. Each multi-plane NIC connects to a group of T0 switches, one per plane. T0 switches of corresponding planes are merged at T1 layer.

If a flow goes from one NIC to another NIC in the same T0 group, we refer to it as T0-local. Conversely, if the end-points are in different T0 groups, the flow must traverse the T1 switches; we refer to it as cross-T1.

5.1 Training Results

MRC has been used to train OpenAI's most recent frontier models at very large scale. The constant rate of T0–T1 link flaps in Cluster A, shown in Fig. 5, has had almost no impact on performance. In fact, fixing these flaps is a very low priority activity, as the operational effort is better spent elsewhere.

We have observed many back-end network failures during training jobs; very few cause job failure or significant performance degradation. Losing NIC–T0 links does affect performance, though we observe that most such events are transient and the job can often recover to full speed quickly without evicting nodes.

Fig. 6 shows an event that occurred during a 50K GPU production pretraining job at OpenAI using MRC on CX-8 NICs in Cluster A. This supercomputer network is configured as 4 x 200Gb/s ports per NIC, with MRC spraying each QP across all four ports. An optical transceiver on a T0 switch suffered a glitch, and flapped all its four links in rapid succession. These four links connected to NICs on four different nodes, of which three were active in the training job at the time. The graph shows the times the NICs think the link was down and when the switches think the link was down, which are not precisely the same times.

As this is a synchronous pretraining job, the slowest node dictates overall performance. In Fig. 6, throughput suffered approximately a 25% reduction over the minute of flaps, then recovered to full speed immediately afterwards. The job did not crash, no QP failed, and the affected nodes did not need to be removed from the job. Such transceiver glitches are rarer than individual link flaps, but they do happen occasionally.

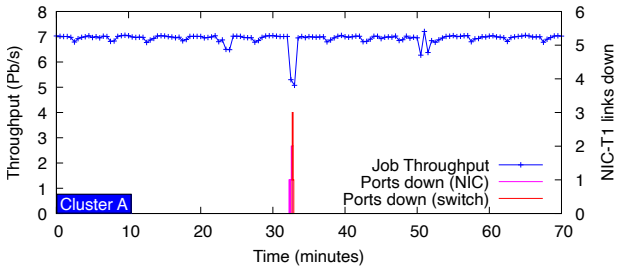


Figure 6: Impact of a flapping NIC-T0 switch transceiver

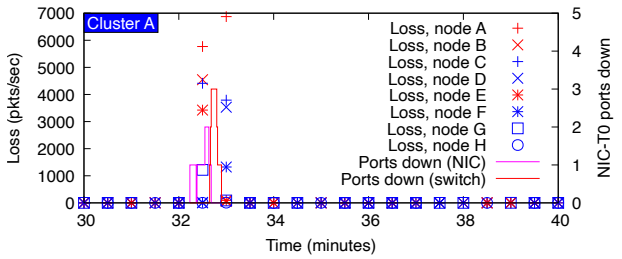


Figure 7: Packet loss rates during the event in Fig. 6

Fig. 7 zooms in on the loss rates of the eight most affected nodes. Red nodes are those whose ports flapped, whereas blue ones were actively sending to the affected nodes at the time of the flap. MRC recovered the losses quickly enough that the impact on the job was relatively small.

During a 75K GPU pretraining job, we had four cases where T1 switches had to be rebooted. Fig. 8 shows one of them. The red line shows the percentage of packets the switch dropped, as seen by Clustermapper. The switch stopped forwarding packets, though it remained up. This was detected by automation and it was rebooted at t=0. The switch fully resumed forwarding by t=2 mins. The purple line shows the percentage of packets dropped each minute, multiplied by 10K to make it visible. Around a quarter of QPs were affected and around 580K packets were dropped. The blue line shows total job throughput: it dips when the switch initially failed, as many QPs lost packets, but they quickly mapped out the bad path and throughput was largely unaffected afterwards. When the switch actually rebooted, there was no impact.

Despite this resiliency, some single points of failure remain. We use 800 Gb/s NIC optics, split into 4x200 Gb/s links. If the NIC transceiver itself flaps, we lose all ports on the NIC, and cannot ride this out because QPs fail. We do see such events when training at scale, but fortunately they are rare. A different optical design might avoid this issue altogether.

5.2 Testbed Results

We present a range of experiments that demonstrate how MRC behaves in more controlled testing environments using the MRC implementations on NVIDIA's CX-8 NIC, AMD's Pollara NIC and Broadcom's Thor Ultra NIC.

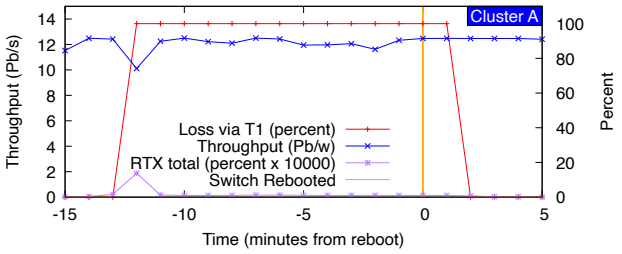


Figure 8: Impact of a T1 switch failure and reboot

5.2.1 Point-to-Point Communication Performance

The table below presents the point-to-point latency and bandwidth communication performance of MRC on CX-8, measured on Cluster B using the `ib_write_lat` and `ib_write_bw` perftest between a single client and a single server with one and four QPs, respectively.

Topology	Message Size	Metric	Result
T0-Local	2 B	Latency	5.09 µs
T0-Local	32 KB	Bandwidth	≈ 770 Gb/s
Cross-T1	2 B	Latency	6.54 µs
Cross-T1	32 KB	Bandwidth	≈ 770 Gb/s

The bandwidth test performs back-to-back writes of a fixed 32 KB message size and reports the achieved application-level GPU-to-GPU bandwidth. Both *T0-local* and *cross-T1* configurations achieve approximately 770 Gb/s, corresponding to 96% of the theoretical peak bandwidth. This indicates that steady-state throughput is not constrained by whether traffic remains within a single T0 or traverses T0 boundaries.

The latency test uses two-byte messages and measures the time from posting the write until completion is delivered at the sender, dividing the result by two to approximate one-way latency. *T0-local* communication achieves a latency of 5.09 μs, while *cross-T1* communication exhibits a higher latency of 6.54 μs. The T0-local latency reflects fixed MRC control-path overheads such as queue pair processing, work request handling, and path management. The additional latency observed in the cross-T1 case is attributable to extra switch hops when crossing between T0 domains. These effects modestly impact short-message latency while having negligible effect on large-message bandwidth.

5.2.2 MRC Response to Link Down and Flap Events

We study how MRC responds to network failures on Cluster B, including link down and link flap events at different points in the network, as well as transceiver flaps. Results for similar experiments run on Cluster D are shown in the Appendix.

We first examine how the throughput of a transfer using four QPs degrades gracefully as NIC–T0 links fail. We run bidirectional `ib_write_bw` between two T0-local NICs and sequentially bring NIC–T0 links down. As shown in Fig. 9a, throughput decreases as links are taken offline one at a time and recovers as they are restored. On link failure, MRC detects the link down event and re-balances EVs across the remaining planes. It also updates the link-state bitmap in SACK packets to notify the remote endpoint that the link is unusable, prompting the remote MRC instance to remap its EV set to avoid the failed plane. We observe a brief interruption of less than a second during failure detection, EV remapping, and packet retransmission, after which both inbound and outbound QPs stabilize over the remaining links. As links recover, traffic in both directions quickly resumes using them.

In Fig. 9b, we flap four of the eight links on a single CX8 NIC to emulate a realistic production failure scenario in which multiple links may be impacted simultaneously due to a shared OSFP port or transceiver. Such failures can cause all links associated with the port to go down at once, resulting in the loss of up to four links concurrently. As shown in Fig. 9b, MRC quickly stabilizes at approximately half the nominal bandwidth when all four links are taken down (around the 5 second mark), and rapidly recovers once the links are restored (around the 17 second mark). While link flap durations in practice are typically much shorter, this experiment synthetically induces a flap by disabling the links and re-enabling them after a fixed interval.

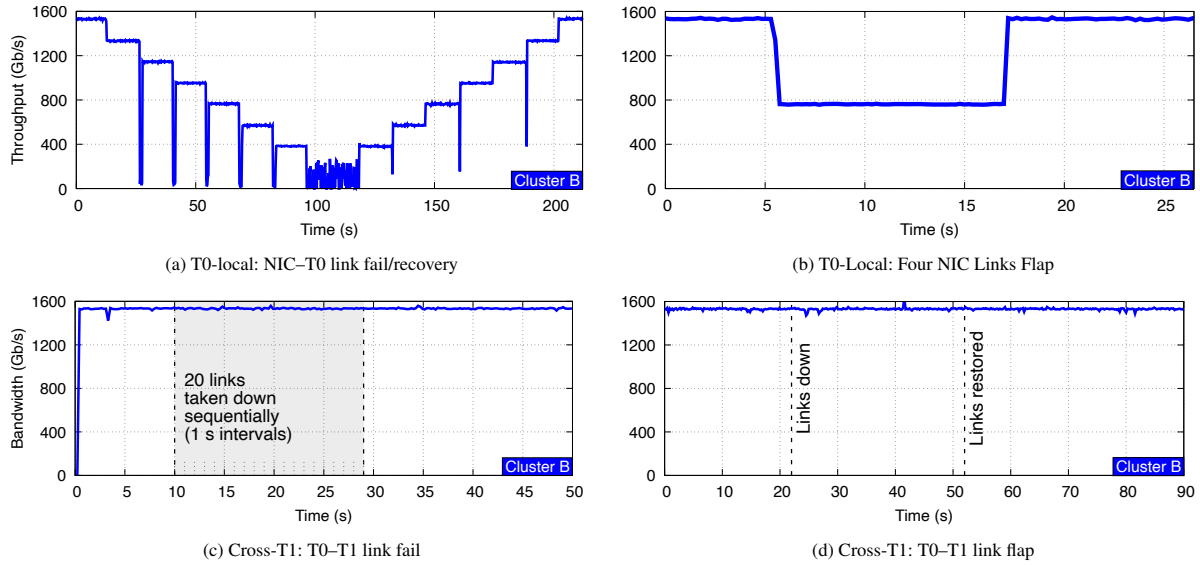
Fig. 9c illustrates the impact of T0–T1 link failures on MRC traffic. Because MRC sprays packets from each QP across a large number of paths, T0–T1 link failures have a substantially smaller impact than NIC–T0 failures. In this experiment, we run cross-T1 `ib_write_bw` and synthetically disable twenty links sequentially. Telemetry collected during the run indicates that the majority of these links were actively carrying traffic prior to failure, demonstrating broad utilization of the available paths. As links are taken down, MRC maps out the affected EVs and retransmits lost packets over alternate paths. Bandwidth is reported at approximately 200 ms granularity, and we observe minimal impact on overall throughput even as active links are removed.

Fig. 9d also shows the effect of flapping eight T0–T1 links simultaneously and restoring them after a few seconds, as may occur due to a switch transceiver flap. Telemetry shows that these links were carrying traffic prior to the flap. As in the previous experiment, MRC reroutes traffic and recovers rapidly, resulting in negligible impact on workload-level performance.

5.2.3 MRC Behavior with T0/T1 Switch Failures

Although uncommon, switches may become unresponsive or crash during production workloads in large-scale training clusters. To evaluate the resulting impact on workload performance, we conducted experiments on Cluster B in which T0 and T1 switches were taken down while running `ib_write_bw` to drive network traffic.

Fig. 10 illustrates the impact of taking a T0 switch down during the experiment. After EV remapping completes, the steady-state bandwidth drops by approximately 100 Gb/s, re-



(a) T0-local: NIC–T0 link fail/recovery

(b) T0-Local: Four NIC Links Flap

(c) Cross-T1: T0–T1 link fail

(d) Cross-T1: T0–T1 link flap

Figure 9: T0-Local and Cross-T1 Reliability Results with ib_write_bw (bi-directional)

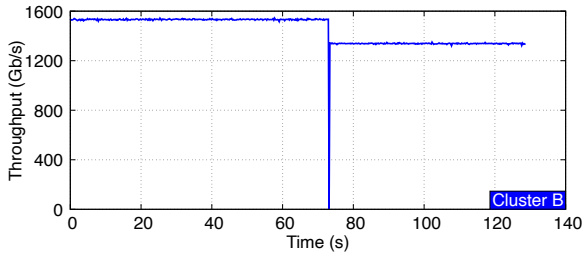


Figure 10: T0-Local ib_write_bw during T0 switch failure.

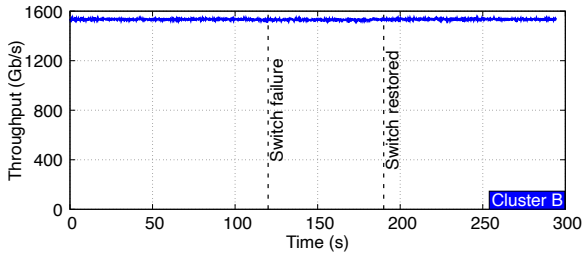


Figure 11: Cross-T1 ib_write_bw during T1 failure/recovery.

flecting the removal of EVs associated with the failed T0 switch from the active EV set. Throughput then stabilizes at a level proportional to the remaining available network capacity. This behavior closely mirrors the results presented in Fig. 9a, confirming that MRC failure handling is driven by end-to-end path availability rather than the specific location or type of underlying fabric fault. As a result, switch-level failures manifest as a predictable reduction in usable path diversity, while preserving application progress and maintaining stable throughput over the remaining healthy paths.

Fig. 11 illustrates the impact of taking a T1 switch down during a cross-T1 `ib_write_bw` experiment with four QPs. In this scenario, each QP is sprayed across a large number of paths, and sufficient alternative EVs remain available to sustain the aggregate network capacity. Consequently, no steady-state bandwidth degradation is observed despite the failure. Once the T1 switch is restored, transient fluctuations subside and bandwidth returns to a stable level.

5.2.4 Robustness to Path-Level Packet Loss

To evaluate the robustness of MRC under transient link impairments, we conducted this experiment on Cluster B using NVIDIA's MRC debug capability to inject controlled errors at the EV level. Specifically, we configured a selected EV to drop 20% of its packets and measured the resulting EV state transitions and end-to-end performance using the cross-T1 bidirectional `ib_write_bw` benchmark. To isolate and clearly observe EV status updates, we restricted the system to a small fixed set of 16 paths, forcing just two paths per each of the eight planes.

During the experiment, we continuously monitored EV status to capture how the system reacts to injected faults. As shown in Fig. 13, EV-A is initially active while EV-B remains inactive. At approximately 51 s, when packet drop is induced on EV-A, its status transitions immediately to inactive, and EV-B is activated as a replacement. This behavior demonstrates MRC's ability to promptly detect faults and switch traffic to an alternative EV, aligning with the sustained line-rate bandwidth observed in Fig. 12.

We repeated the experiment with different packet drop rates and observed the same qualitative behavior: the affected EV was promptly removed from the active set, traffic was transparently redirected to alternative paths, and application-level bandwidth remained stable.

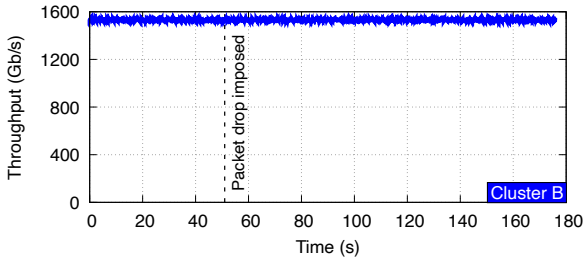


Figure 12: Packet-drop reliability experiment.

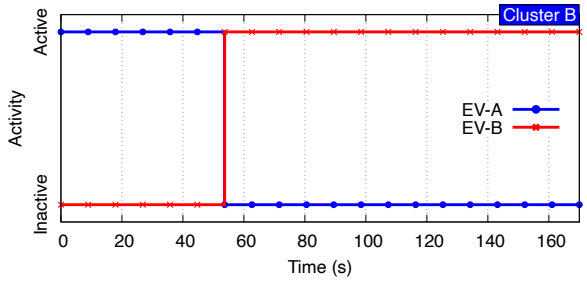


Figure 13: Path activity during packet-drop experiment.

5.2.5 Load Balancing Across EVs

We evaluate MRC's ability to dynamically balance load across EVs using a controlled two-flow experiment in Cluster B. The setup consists of two communication pairs operating over two EVs (EV-A and EV-B) within the same plane. We first establish an initial flow between `client1` and `server1`. In the steady state, this flow is mapped entirely onto EV-A, achieving near line-rate bandwidth while EV-B remains idle. At approximately 65 s, we introduce a second flow between `client2` and `server2`, which is explicitly forced to use EV-A. This creates transient congestion, with both flows contending for the same EV.

Upon detecting this congestion via ECN, MRC triggers traffic redistribution to rebalance load across the available EVs. Specifically, the `client1-server1` flow is migrated to EV-B, while the `client2-server2` flow continues on EV-A. This transition is clearly visible in the per-EV activity traces shown in Fig. 14: EV A transitions from serving a single flow to hosting the second flow exclusively, while EV-B becomes active as it assumes responsibility for the migrated flow. Importantly, this redistribution occurs without observable performance degradation. Both client-server flows sustain near-peak bandwidth throughout the transition, with aggregate throughput remaining close to line rate. The results demonstrate that MRC can effectively rebalance traffic across EVs in response to dynamic load changes, maintaining stable, high throughput with minimal disruption.

5.2.6 NCCL Collective Execution at Scale

We also conducted NCCL microbenchmark experiments on Cluster B to evaluate the scalability of MRC. Specifically, we

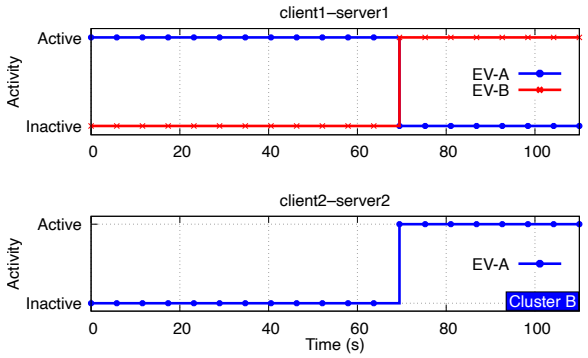


Figure 14: Path activity during load balancing experiment.

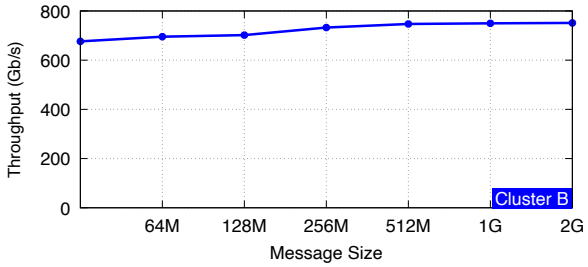


Figure 15: NCCL send/recv performance at 42K GPUs scale.

used the NCCL-tests `sendrecv` benchmark, in which each node concurrently sends data to and receives data from a neighboring peer, forming a steady-state bidirectional communication pattern. The reported bandwidth corresponds to the measured per-NIC throughput. As shown in Fig. 15, NCCL over MRC achieves up to 92 GBytes/s for large message sizes at a scale of 42K GPUs.

5.2.7 Comparison with RoCE

We don't have a large deployment that can perform a direct comparison between MRC and RoCE, but we built two small scale testbeds, one using AMD Pollara NICs (Cluster C) and one using Broadcom Thor Ultra NICs (Cluster D), to compare.

Cluster C consists of 64 GPUs, each paired with a Pollara 400 Gb/s NIC, using TH5 switches in a two-tier Clos topology. For RoCE, we configure a single-plane network, as is common, with 400 Gb/s links and ECMP routing. In this configuration, PFC is enabled and DCQCN is used as congestion control to avoid excessive PFC activation. For MRC, we configure a four-plane network with four 100 Gb/s links per NIC; we disable PFC and use SRv6 routing. This is the closest like-for-like comparison we can build while giving each protocol its preferred environment: the GPUs, NICs, switches, and aggregate bandwidth are identical.

We wish to understand two things. First, does MRC load balance better than RoCE, even with QP scaling? Second, does MRC's selective retransmit really help typical AI collectives in the presence of network problems? To answer this, we run two sets of experiments. First, we run all-reduce on a ring,

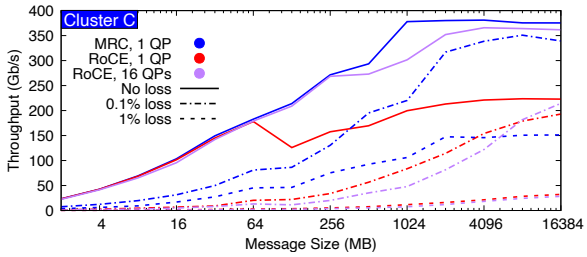


Figure 16: MRC and RoCE performing 64-way ring all-reduce, for varying message size and loss rates

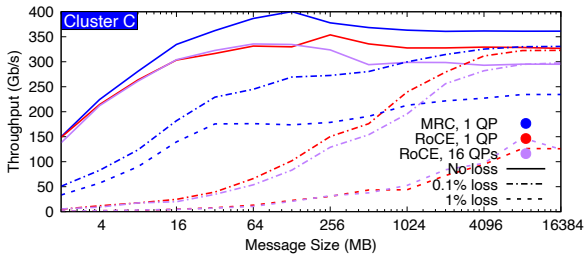


Figure 17: MRC and RoCE performing 64-way all-to-all, for varying message size and loss rates

which generally requires each node to send to one node and receive from one node simultaneously, with the goal that all transfers reach line rate. Any glitch will cause a bubble and hurt overall performance. Second, we run all-to-all, which requires every node to send to and receive from multiple nodes simultaneously. The goal is to observe load balancing and retransmit performance during a collective that can cause incast congestion.

Fig. 16 shows mean bandwidth when performing all-reduce for increasing message sizes. For MRC we use one QP, whereas for RoCEv2 we show the results using one QP and 16 QPs. The solid curve shows the baseline results with no induced packet loss. For small message sizes, the collective is latency-bound, and there is little difference between RoCE and MRC. For larger message sizes we become increasing bandwidth-bound. RoCE with one QP suffers from ECMP hash collisions and generally achieves only half the possible throughput. The conventional way to improve load balancing with RoCE is by QP scaling - load balancing a transfer across multiple QPs to reduce the impact of flow collisions. We find that, for RoCE, QP scaling does indeed help, but we see very little gain beyond 8 QPs (the figure shows 16). In contrast, one MRC QP spraying across 256 paths achieves better performance than 16 QPs, primarily due to doing a good job of load-balancing the network.

The dashed lines show performance as we add packet loss at 0.1% and 1% levels. We do this by adding an inline P4 [1] program to all NICs in the cluster that randomly drops incoming packets at a prescribed rate. RoCE was not designed to be resilient to packet loss and, as expected, it suffers poor performance. MRC is more resilient - with large message sizes it can retransmit fast enough that 0.1% loss has little impact. At smaller message sizes the collective is more latency-bound, and recovering packet losses has more impact due to tail losses not being possible to mask. MRC still performs relatively well though. We don't expect an AI cluster to have persistent high loss rates, but we do see a range of causes of loss transients; synchronous pretraining does not care about mean performance—only the tail matters—so at scale, having a protocol that tolerates loss is important.

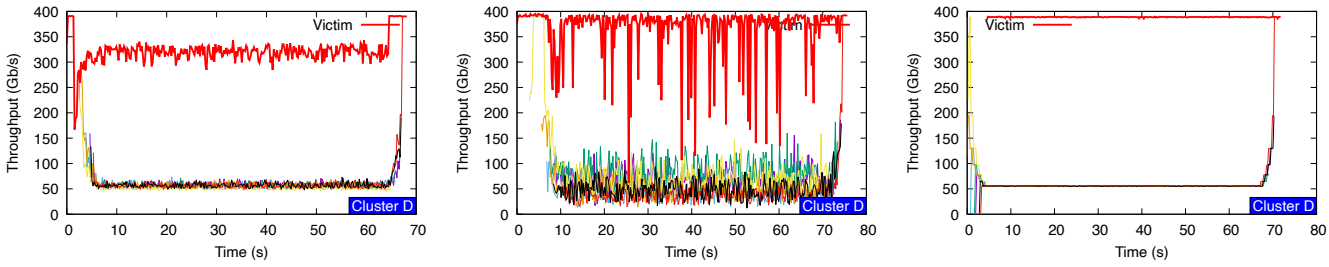
At 1% loss RoCE is pretty much ususable and even MRC only gets around a third of the intended throughput. This is enough to ride out a short transient burst of loss, but not good enough for continued training. It is worth noting that this test shows worse performance than MRC would get in typical real-world scenarios because the loss is applied on all planes simultaneously. If a T0-T1 link is dropping at a high rate, MRC will simply stop using it, as in Fig. 13. If a NIC-T0 link is dropping at a high rate, our Clustermapper monitor will detect it, and a denylist entry can be added to avoid the broken plane, limiting the performance degradtion to just the fraction of bandwidth lost by dropping one plane. Whether this is sufficient to allow the node to persist in a training job is then a policy decision.

Fig. 17 shows mean bandwidth when performing all-to-all for increasing message sizes. Again the solid lines are baseline with no loss, and the dashed lines add random loss. In this case more QPs are active simultaneously, so RoCE's load balancing is not such an issue. Indeed, we see that QP scaling is not helping RoCE: 16 RoCE QPs per destination actually perform slightly worse than one QP and performance starts to drop off slightly with more than two QPs. For all message sizes MRC outperforms RoCE, but the difference is greater in the regime that is bandwidth-bound.

With loss, RoCE suffers worse than MRC, as expected, but the difference is much greater at smaller message sizes. At large message sizes and 0.1% loss, RoCE barely sees any degradation. This is because so many QPs are active simultaneously that few packets are in flight on each one. When the transfers are large enough one QP pausing for retransmission is mostly masked by others being active. At small message sizes, there isn't time to mask loss, and RoCE does very poorly. MRC's SACK-based retransmission helps greatly here, despite this test causing loss on all planes.

5.2.8 Collateral Damage

Multiple collectives performing different axes of training parallelism may run simultaneously and can interfere with each other. This can particularly be a problem with PFC, which MRC was designed to avoid. Lossless networks struggle with incast traffic patterns when the congestion spreads and can affect unrelated "victim" traffic. To test this behaviour we run a cross-spine 7 to 1 incast traffic pattern, and in parallel we have another "victim" connection to an idle destination in the



(a) RoCEv2 with DCQCN, 1 QP (b) RoCEv2 with DCQCN, 10 QPs (c) MRC, 1 QP

Figure 18: 7 to 1 incast with a victim flow destined to a different node in the same rack.

same rack as the incast target.

The testbed for this experiment is Cluster 4, a small-scale testbed with 16 servers each with one RTX6000 GPU and a Broadcom Thor Ultra NIC. We use VRFs on Broadcom TH5 switches to emulate a single-plane 2-tier Clos topology with 4 racks each with 4 servers and four spine switches. Due to server limitations we run all links in this testbed at 400 Gb/s.

With RoCE, if we do not use DCQCN and rely solely on PFC to manage congestion, the victim flow is pulled down to a rate not much greater than the incast flows (see Appendix for details). DCQCN is designed to greatly reduce PFC, and we find it does help, but is not sufficient. With a single QP (Fig. 18a), victim-flow performance still degrades by about 25%. With eight QPs (Fig. 18b), sharing is worse, but the average impact on the victim flow is smaller; however, there are one-second intervals where the victim's throughput is 100Gbps, a 75% drop from optimal. In both cases, DCQCN cannot properly control the bottleneck queue and still generates some PFCs. In contrast, MRC (Fig. 18c) almost perfectly shares the bottleneck link among the incast flows and has no impact on the victim flow.

In principle DCQCN parameter tuning should reduce this issue. However, configuring DCQCN properly is very hard because it is traffic pattern specific, to the point that some hyperscalers have disabled it in production [16]. We provide some tuning results in the Appendix that illustrate this.

6 Related Work

Load balancing effectiveness is defined by distribution granularity: per-flow, per subflow and per packet. Single path transports (RoCEv2) using per-flow ECMP [24] are simple but collision-prone. Solutions to improve flow placement span centralized (Hedera [2] or MicroTE [5]), switch-based (Flowlet switching [42], Presto [21] and DLB [8]) and host-based approaches (e.g. PLB [34], Flowcut [6], and FlowBender [27]). All these approaches often react too slowly for bursty AI traffic, and have had limited success under high load. Multipath operation is needed for collisions to be efficiently avoided. Switch-based multipathing solutions exist that provide in-order per flow delivery to hosts (Drill [18], CONGA [3] and Stardust [49]; commercial offerings from Broadcom and Cisco) - however, these are proprietary solutions and require homogeneous networks.

Many deployments today rely on RoCEv2 with ECMP routing but use application-level multipath transports, typically implemented in the collective communication layer to reduce the effects of colisions, e.g. NCCL [25] QP scaling, MSCCL [37], NCCLX [38], UCCL [47]. Application-level multipathing mitigates the issues but does not completely solve them, as we have shown in our evaluation.

Multipath TCP (MPTCP [35]) improves utilization by spreading a single connection across multiple subflows. MPTCP and similar approaches must keep state per subflow however, but they can be used for any datacenter topology and work particularly well with long running traffic. Google's Falcon [40] uses this approach to implement a multipath replacement for RoCEv2 in the NIC.

Per-packet load balancing (or packet spraying) has been proposed for Clos networks to reduce the amount of state. RPS [12], Homa [31] and NDP [20] are oblivious to path state, but struggle with asymmetric congestion and partial/gray failures. By keeping a small amount of state per (virtual) path, feedback-driven methods like MPRDMA [29], Hermes [46], REPS [7] or Strack utilize ECN or delay signals to move traffic away from congested or failed paths.

MRC changes RoCEv2 to enable packet-level load balancing, selective retransmissions and improve reliability in multi-plane networks; its target deployment is best-effort (i.e. lossy) networks that support packet trimming. To achieve this, MRC builds upon a wide body of prior work as follows. IRN was one of the first approaches to add selective retransmission to RoCEv2 [30], while MPRDMA added multipath operation and selective retransmission as well as keeping per path state [29]. These approaches still rely on lossless networks, but it is difficult to completely avoid HoL blocking issues.

MRC is similar to the Ultra Ethernet Transport, an industry-driven standard that replaces RoCEv2 with a brand new protocol stack aiming to support both HPC and AI workloads [10]. UET has standardized packet trimming, and also targets operation in best-effort networks.

In contrast to most existing works which use host-driven ECMP routing or switch spraying, MRC uses source routing with SRv6 to increase robustness at scale. Filsfils et al. [14] have validated SRv6 micro-segment (uSID) based path placement for single path RoCEv2.

We are not the first to propose topology co-design for AI network (see [17] for an overview of this space). Alibaba's HPN [33] uses a dual-ToR, rail optimized networks, connecting up to 15K GPUs in a two-tier design. Wang et al. [43] propose using a rail-only approach to have a single tier of switches. However, we are among the first to design and deploy a multi-plane network to reach 100K+ GPUs with two switch tiers.

Experiences from multiple hyperscalers have highlighted the negative effect of failures on training jobs [13, 33]. Our multi-plane, static-routed SRv6 approach directly targets being able to ride out network failures.

7 Conclusions

MRC is designed to load balance a multi-plane network by spraying each QP across all planes and many paths in each plane, performing fine-grain active load balancing and routing around failures. We implemented MRC in 800Gb/s NICs from Nvidia, Broadcom, and AMD, and built multiple supercomputers that use a two-tier multi-plane topology running MRC in the back-end network. MRC's ability to route around failures allowed us to disable dynamic routing; instead, we use SRv6 source routing with static routes in the switches. These supercomputers have been used to train OpenAI's latest frontier models, and we have observed that this design allows very large AI pretraining jobs to ride out network failures that would previously have caused the job to fail, while most failures have minimal impact on job step time. We find that static source routing gives us very good observability and reduces operational burden, while MRC's resilience means that many network failures are not even urgent to repair.

8 Acknowledgments

Developing and successfully deploying MRC has involved a great many people; we would particularly like to thank the following (in alphabetical order): Rukhsana Ansari, Dragos Argint, Cristi Baciu, Bar Becker, Omri Ben David, Shai Ben Haim, Paul Blakey, Jeremias Blendin, Brian Box, Greg Brockman, Mihai Brodschi, Evan Burness, Trevor Cai, Rory Carmichael, Miguel Castro, Peng Cheng, James Crooks, Janet Cui, Valerie Cutts, Michael Dalton, Biswa Dash, Shawn Dashuai Zhang, Mark Debbage, Karl Deng, Weixin Deng, Gregor Dick, Saurabh Dighe, Qixin Dong, Gili Doweck, Iulian Dracea, Peter Dunning, Yakov Dyadkin, Madan Easwaramoorthy, Elliot Edmunds, Sally Egan , Lior Erets Kdosha, Ze Gan, Naren Gathoo, Renaud Gaubert, Ahmad Ghalayini, Jeff Glover, Guru Harakere, Damian Hazen, John Huber, Richard Hughes, Rita Hui, Tony Hurson, Changho Hwang, Iva Ivanov, Vivek Jain, Riff Jiang, Anuj Kalia, Ali Kamali, Nemanja Kamenica, Vivek Kashyap, Bhunu Kathavarayan, Ady Khalifa, Xinhao Kong, Shilpa Kothapalli, Gawaskar Kumar, Ariel Levkovich, Binyang Li, Xin Liu, Jie Mao, Ilias Marinos, Charlie Mbariky, Scott McDaniel, John Mead, Sharad Mehrotra, Luke Melton, Yan Mo, Scott Moe, Malek Musleh, Nikhil Nanal, Suresh Nedunchezhian, Duc Phong Nguyen, Lisa Nguyen, Wael Noureddine, Vlad Olteanu, Shane O'Neil, Shahar Oren, Kumaresh Perumal, Jonas Pfefferle, Rajesh Pukhraj Jain, Catalin Puscoci, Melur Raghuraman, Mahdi Ramezani, David Riddoch, Uday Ruddarraju, Kathryn Russell, Neelabh Sahay, Lorenzo Saino, Rafael Salas, Siva Santosh Pyla, Emnaual Scaria, Brent Schartung, Karen Schramm, Hemal Shah, Gilad Shainer, Sanjay Shanbhogue, Rohit Sharma, Eden Shimoni Delna Sholapurwalla, Pradeep Sindhu, Prince Sunny, Vijay Swaminathan, Jason Teplitz, Gaurav Thareja, Bejoy Thomas, Sam Truslow, Dev Upadhyay, Vamsi Vadlamuri, Srihari Vegesna, Ram Velaga, Jijun Wang, Yossi Wortzel, Changrong Wu, Weijia Yuan, Reza Zamani, Jie Zhang, Yanzhao Zhang,

References

[1] P4 Open Source Programming Language. https://p4.org/.

[2] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2010.

[3] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2014.

[4] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Comput. Surv.*, 52(4), August 2019.

[5] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA, 2011. Association for Computing Machinery.

[6] Tommaso Bonato, Daniele De Sensi, Salvatore Di Girolamo, Abdulla Bataineh, David Hewson, Duncan Roweth, and Torsten Hoefler. Flowcut switching: High-performance adaptive routing with in-order delivery

guarantees. *IEEE Transactions on Networking*, 34:1974–1987, 2026.

[7] Tommaso Bonato, Abdul Kabbani, Ahmad Ghalayini, Michael Papamichael, Mohammad Dohadwala, Lukas Gianinazzi, Mikhail Khalilov, Elias Achermann, Daniele De Sensi, and Torsten Hoefler. REPS: Recycled entropy packet spraying for adaptive load balancing and failure mitigation, 2026.

[8] Broadcom. ECMP Dynamic Load Balancing. https://docs.broadcom.com/doc/56980-DS, 2019.

[9] Weiqiang Cheng, Clarence Filsfils, Zhenbin Li, Bruno Decraene, Dezhong Cai, Daniel Voyer, Francois Clad, Shay Zadok, Jim Guichard, Aihua Liu, Robert Raszuk, and Cheng Li. Compressed SRv6 Segment List Encoding. RFC 9800, June 2025.

[10] Ultra Ethernet Consortium. Ultra Ethernet specification v1.0.1, 2025.

[11] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, February 2013.

[12] Advait Dixit, Pawan Prokash, Charlie Y. Hu, and Ramona R Kompella. On the Impact of Packet Spraying in Data Center Networks. In *International Conference on Computer Communications (INFOCOM)*. IEEE, 2013.

[13] Aaron Grattafiori et al. The LLaMa 3 herd of models, 2024.

[14] Clarence Filsfils, Pablo Camarillo, Ahmed Abdelsalam, Arianna Quinci, Angelo Tulumello, Andrea Mayer, Pierpaolo Loreti, Lorenzo Bracciale, and Stefano Salsano. Toward deterministic path placement in AI backends: A practical SRv6-based architecture. In *21st International Conference on Network and Service Management (CNSM)*, Bologna, Italy, October 2025. IFIP.

[15] Clarence Filsfils, Darren Dukes, Stefano Previdi, John Leddy, Satoru Matsushima, and Daniel Voyer. Segment Routing over IPv6 (SRv6) Network Programming. RFC 8986, February 2021.

[16] 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. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 57–70, New York, NY, USA, 2024. Association for Computing Machinery.

[17] Alexandru M. Gherghescu, Vlad-Andrei Bădoiu, Alexandru Agache, Mihai-Valentin Dumitru, Iuliu Vasilescu, Radu Mantu, and Costin Raiciu. I've got 99 problems but FLOPS ain't one. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, HotNets '24, page 195–204, New York, NY, USA, 2024. Association for Computing Machinery.

[18] Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. DRILL: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 225–238, New York, NY, USA, 2017. Association for Computing Machinery.

[19] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. *SIGCOMM Comput. Commun. Rev.*, 45(4):139–152, August 2015.

[20] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting Datacenter Networks and Stacks for Low Latency and High Performance. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2017.

[21] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based load balancing for fast datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 465–478, New York, NY, USA, 2015. Association for Computing Machinery.

[22] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Characterizing the influence of system noise on large-scale applications by simulation. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, page 1–11, USA, 2010. IEEE Computer Society.

[23] Torsten Hoefler, Karen Schramm, Eric Spada, Keith Underwood, Cedell Alexander, Bob Alverson, Paul Bottorff, Adrian Caulfield, Mark Handley, Cathy Huang, Costin Raiciu, Abdul Kabbani, Eugene Opsasnick, Rong Pan, Adee Ran, and Rip Sohan. Ultra ethernet's design principles and architectural innovations, 2025.

[24] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, November 2009.

[25] Zhiyi Hu, Siyuan Shen, Tommaso Bonato, Sylvain Jeaugey, Cedell Alexander, Eric Spada, James Dinan, Jeff Hammond, and Torsten Hoefler. Demystifying NCCL: An in-depth analysis of GPU communication protocols and algorithms, 2026.

[26] InfiniBand Trade Association (IBTA). The RoCE initiative. (Accessed: May 2021).

[27] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. FlowBender: Flow-level Adaptive Routing for Improved Latency and Throughput in Datacenter Networks. In *Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. ACM, 2014.

[28] Hongqiang Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. CrystalNet: Faithfully emulating large production networks. In *SOSP '17 Proceedings of the 26th Symposium on Operating Systems Principles*, pages 599–613. ACM, October 2017.

[29] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. Multi-path transport for RDMA in datacenters. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 357–371, Renton, WA, April 2018. USENIX Association.

[30] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 313–326, New York, NY, USA, 2018. Association for Computing Machinery.

[31] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A Receiver-driven Low-latency Transport Protocol Using Network Priorities. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2018.

[32] Fabrizio Petrini, Darren J. Kerbyson, and Scott Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, page 55, New York, NY, USA, 2003. Association for Computing Machinery.

[33] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, and Dennis Cai. Alibaba HPN: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 691–706, New York, NY, USA, 2024. Association for Computing Machinery.

[34] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. PLB: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 207–218, New York, NY, USA, 2022. Association for Computing Machinery.

[35] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2010.

[36] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation ai scale, 2022.

[37] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. TACCL: Guiding collective algorithm synthesis using communication sketches, 2022.

[38] Min Si, Pavan Balaji, Yongzhou Chen, Ching-Hsiang Chu, Adi Gangidi, Saif Hasan, Subodh Iyengar, Dan Johnson, Bingzhe Liu, Regina Ren, Deep Shah, Ashmitha Jeevaraj Shetty, Greg Steinbrecher, Yulun Wang, Bruce Wu, Xinfeng Xie, Jingyi Yang, Mingran Yang, Kenny Yu, Minlan Yu, Cen Zhao, Wes Bland, Denis Boyda, Suman Gumudavelli, Prashanth Kannan, Cristian Lumezanu, Rui Miao, Zhe Qu, Venkat Ramesh, Maxim Samoylov, Jan Seidel, Srikanth Sundaresan, Feng Tian, Qiye Tan, Shuqiang Zhang, Yimeng Zhao, Shengbao Zheng, Art Zhu, and Hongyi Zeng. Collective communication for 100k+ GPUs, 2026.

[39] Rachee Singh, Muqeet Mukhtar, Ashay Krishna, Aniruddha Parkhi, Jitendra Padhye, and David Maltz. Surviving switch failures in cloud datacenters. *SIGCOMM Comput. Commun. Rev.*, 51(2):2–9, May 2021.

[40] 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. Falcon: A reliable, low latency hardware transport. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 248–263, New York, NY, USA, 2025. Association for Computing Machinery.

[41] Rip Sohan, Eric Spada, Eric Davis, Mark Handley, Idan Burstein, Tony Hurson, Jithin Jose, Vivek Kashyap, Rong Pan, and Sayantan Sur. Multipath Reliable Connection (MRC) Specification. Specification Version 1.0, Open Compute Project, 2026.

[42] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 407–420, Boston, MA, March 2017. USENIX Association.

[43] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. Rail-only: A low-cost high-performance network for training LLMs with trillion parameters. In *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 1–10, 2024.

[44] Zijie Yan, Hongxiao Bai, Xin Yao, Dennis Liu, Tong Liu, Hongbin Liu, Pingtian Li, Evan Wu, Shiqing Fan, Li Tao, et al. Scalable training of mixture-of-experts models with Megatron Core. *arXiv preprint arXiv:2603.07685*, 2026.

[45] Zuoning Yin, Matthew Caesar, and Yuanyuan Zhou. Towards understanding bugs in open source router software. *SIGCOMM Comput. Commun. Rev.*, 40(3):34–40, June 2010.

[46] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. Resilient datacenter load balancing in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 253–266, New York, NY, USA, 2017. Association for Computing Machinery.

[47] Yang Zhou, Zhongjie Chen, Ziming Mao, ChonLam Lao, Shuo Yang, Pravein Govindan Kannan, Jiaqi Gao, Yilong Zhao, Yongji Wu, Kaichao You, Fengyuan Ren, Zhiying Xu, Costin Raiciu, and Ion Stoica. UCCL-Tran: An extensible software transport layer for machine learning workloads. *USENIX OSDI*, 2026.

[48] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale RDMA deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 523–536, New York, NY, USA, 2015. Association for Computing Machinery.

[49] Noa Zilberman, Gabi Bracha, and Golan Schzukin. Stardust: Divide and conquer in the data center network. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 141–160, Boston, MA, February 2019. USENIX Association.

Appendix

We provide some additional results demonstrating MRC's robustness using Broadcom's Thor Ultra NIC in Cluster D; this is a small 400Gb/s single-plane two-tier network. MRC uses SRv6, and we compare with the Thor Ultra plain RoCE implementation running with PFC and DCQCN enabled.

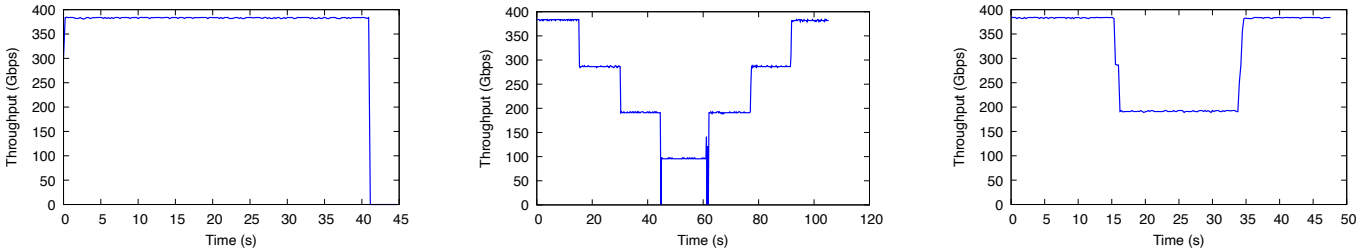
We first test the resilience of MRC on Thor Ultra to link failures. In Figure 19a NIC–T0 and T0–T1 links are all 400Gb/s, and we show the throughput reported by `IB_WRITE_BW` as we progressively fail three of the four T0–T1 links. In this case there is sufficient capacity on the remaining T0–T1 link and, as in Figure 9d, MRC on Thor Ultra is able to quickly recover from failed links with no discernible impact for end-to-end throughput.

In our next experiments we reduce the speed of the T0–T1 links to just 100 Gb/s, so that as links fail a bandwidth bottleneck is created. Figure 19b shows the effect of sequentially disabling three of four links and then adding them back. This experiment differs from that Figure 9a in that the failures are not the directly attached NIC links. MRC is able to fail onto the remaining path while giving throughput that tracks the remaining available capacity. Figure 19c shows a variation of the same experiment where we drop two links at once, and then bring them back; the results confirm that MRC can track the available capacity closely.

Load balancing micro-benchmarks. MRC's spraying and active load balancing is designed to eliminate congestion caused by flow collisions. We present some results using RoCEv2 that illustrate the nature of this problem.

We have two racks (8 hosts) send data to the other two racks, in a one-to-one pattern using `ib_write_bw` that fully loads the T0 uplinks. When using a single QP per transfer with RoCEv2, some flows get line rate, but some suffer from flow collisions and achieve only a half or a third of line rate; the outcome is the same when using DCQCN [48] or PFC alone and the number of collisions varies between runs. See Figure 20a for a run with PFC but no DCQCN. In contrast, in this same scenario, MRC using a single QP is able to achieve 390 Gbps for all flows (not shown).

When we use 8QPs per transfer, it matters whether we use PFC alone or DCQCN with PFC, as shown in Figures 20b and 20c. When only PFC is used, having multiple QPs only marginally helps: the QPs experiencing most congestion will cause the sending NIC to be throttled by PFC, which will in turn slow down all other QPs from the same host. DCQCN avoids this phenomenon by reducing the sending rate based

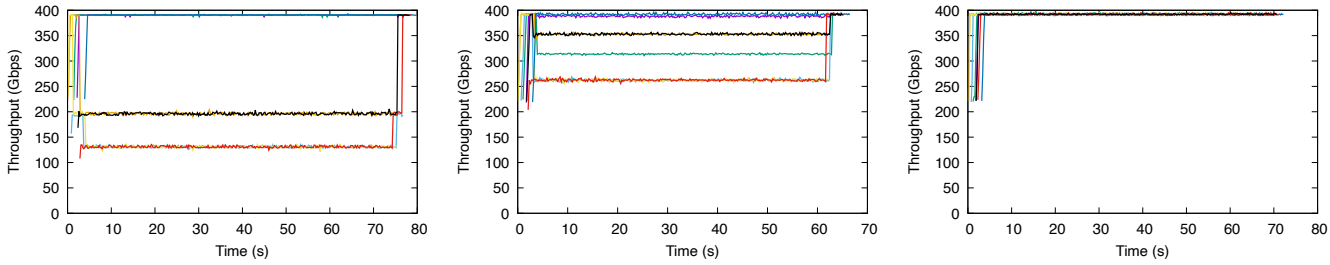


(a) Throughput while sequentially removing three out of four 400Gb/s T0-T1 links

(b) Sequentially removing and then adding three of four 100Gb/s T0-T1 links

(c) Sequentially removing and then adding three of four 100Gb/s T0-T1 links.

Figure 19: ib_write_bw performance between two servers in different racks during failures.

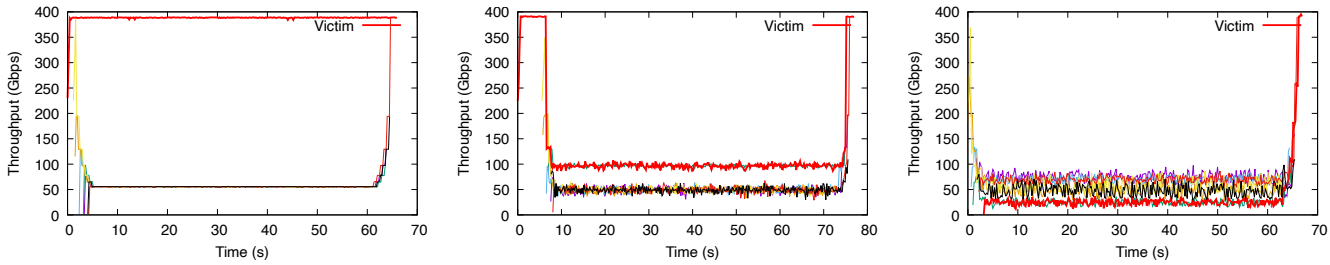


(a) RoCEv2 with PFC, 1QP

(b) RoCEv2 with PFC, 8QPs

(c) RoCEv2 with DCQCN, 8QPs

Figure 20: Permutation throughput when servers from two racks source flows to servers in other two racks.

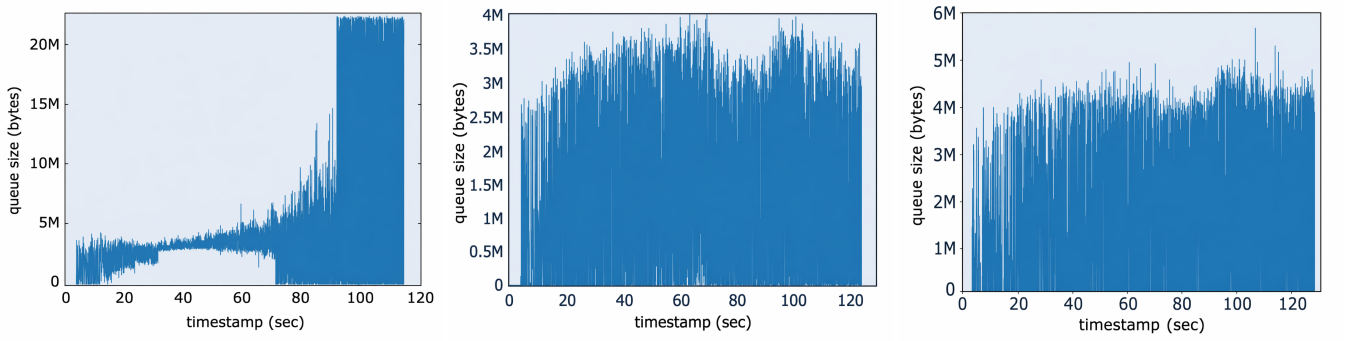


(a) MRC, 8QPs

(b) RoCEv2 with PFC only, 1QP

(c) RoCEv2 with PFC only, 8QPs

Figure 21: 7 to 1 incast with a victim flow destined to the same rack.



(a) Default DCQCN parameters

(b) More aggressive CC profile

(c) Most aggressive CC profile

Figure 22: DCQCN leaf-host queue dynamics in a 15:1 incast where flows arrive 5s apart.

on ECN signals [48]. Again, for MRC the throughput is line rate also with 8QPs.

Collateral damage of incast. Lossless networks struggle with incast traffic patterns when the congestion spreads and can affect unrelated "victim" traffic as discussed in Section 5.2.8. Here we present some additional results for the same cross-spine 7 to 1 incast traffic pattern run in parallel to a "victim" connection when using RoCEv2 with PFC only, and when using MRC with 8QPs. Results for RoCEv2 with DCQCN and MRC with 1QP have been presented in Figures 18a, 18b and 18c.

We show the results in Figures 21a, 21b and 21c. With PFC alone the effect on the victim flow is dire, with the victim flow only achieving 30 to 100Gbps depending on ECMP path choice and fairness. MRC with 8QPs behaves exactly the same as with 1QP, perfectly sharing the bottleneck link among the incast flows and has no impact on the victim flow.

In principle DCQCN parameter tuning should fix this issue. However, configuring DCQCN properly is very hard because it is traffic pattern specific.

To show why this is the case, we show the TOR queue size to the destination during a 15-to-one incast where flows arrive sequentially every 5s. We tested three different DCQCN profiles recommended to customers: the default one (Fig. 22a), a more aggressive CC profile (Fig. 22b) and a most aggressive one (Fig. 22c). In the default case, the queue controlled with a small number of flows (less than 10) and then it goes into PFC mode, but the bottleneck throughput is line-rate. In the more aggressive profile the queue is controlled (with some wild dynamic range) but the queue is sometimes empty and this leads to around 10% loss in bottleneck throughput. Finally, with the most aggressive setting, the queue usage spikes are similar but average utilization is lower, and the total throughput is 20% slower than line-rate.