NVIDIA Spectrum-X: Hardware-Accelerated Multiplane Networking for Large-Scale AI Training

Sajy Khashab, Albert Gran Alcoz, Alon Gal, Jacky Romano, Rani Abboud,
Yonatan Piasetzky, Lior Maman, Amit Nishry, Barak Gafni, Omer Shabtai, Matty Kadosh,
Dror Goldenberg, Gilad Shainer, Mark Silberstein *

NVIDIA

arXiv:2605.21187v2 [cs.NI] 20 Oct 2026

Abstract

As distributed model training scales to span hundreds of thousands of GPUs, scale-out networks face unprecedented performance and efficiency demands. NVIDIA Spectrum-X Ethernet has been designed from the ground up to achieve predictable and stable network performance with high utilization and low latency. This paper presents the Spectrum-X multiplane architecture, which replaces hierarchical depth with topological parallelism, and introduces hardware-accelerated load balancing in NICs and switches as the key architectural approach to provide fast reaction to highly dynamic network conditions at the microsecond timescales that AI training workloads demand. We describe the motivation, design principles, evaluation methodology and performance on state-of-the-art benchmarks, as well as the lessons we learned from deploying and debugging Spectrum-X networks in large-scale systems. Our evaluation highlights production-grade AI infrastructure performance across four core dimensions: 98% of the theoretical line rate with low jitter-free latency; strong cross-tenant isolation for concurrent workloads; robust, capacity-proportional bisection bandwidth and 7% latency increase for 10% fabric link failures; and rapid reaction to host and fabric link flaps during LLM training workloads.

1 Introduction

The rapid scaling of AI models has driven the deployment of training clusters with hundreds of thousands of GPUs, making the scale-out network a first-order bottleneck for distributed training. In Ethernet-based AI clusters, RDMA over Converged Ethernet (RoCE) has emerged as the dominant transport substrate [6, 15].

Despite well-known limitations, the current generation of AI training systems has been able to achieve high performance with RoCE through careful engineering and tuning. These systems operate under several constraints: (i) RoCE assumes near in-order delivery, (ii) it relies on coarse-grained Equal-Cost Multi-Path (ECMP) load balancing that fails to saturate the network due to the low-entropy traffic patterns of AI workloads [6], (iii) it uses lossless Ethernet which can induce head-of-the-line blocking and congestion propagation [18,41], causing jitter and unstable network performance, and (iv) it relies on congestion control mechanisms such as DCQCN which are difficult to tune for collective-heavy workloads, often trading modest completion-time gains for degraded PFC behavior [6]. Nonetheless, these challenges have been mitigated sufficiently to enable AI training at current scales. However, ad hoc RoCE tuning is reaching its limits. Emerging AI data centers executing a single AI training workload on hundreds of thousands of GPUs pose far stricter efficiency and robustness requirements than before, and expose fundamental structural limitations in existing network fabrics.

Fast fabric response. Modern training workloads are dominated by synchronous collectives, which generate highly bursty and structured traffic patterns [6]. With 800 Gbps and higher per-GPU network bandwidth, flows of a few to tens of megabytes complete in $O(100)$ μs so even small variations in path delay due to short-lived congestion or minor load imbalances have direct impact on the completion time. Similarly, at high fabric utilization, insufficient inter- and intra-job isolation among concurrent collectives introduces jitter, affecting collective completion time significantly. Due to tight temporal synchronization, such transient events disproportionately impact end-to-end performance, leading to underutilization of tens of thousands of GPUs. Metrics such as high-percentile latency and worst-flow completion time, rather than averages, best capture training efficiency [19,35]. To optimize them, the fabric must react *extremely quickly* to bursts, intermittent congestion, and transient load imbalances. Existing mechanisms cannot react to real-time congestion state at the microsecond timescales that collective synchronization demands.

Size and bandwidth scaling. Reaching the scale of hundreds of thousands of GPUs in a single network necessitates adding network tiers, increasing the path length. Furthermore, scaling the bisection bandwidth under current technological constraints can be achieved via link parallelism, which increases

* Technion - Israel Institute of Technology and NVIDIA

topological complexity and reduces the available switch radix. As a result, conventional approaches to scaling the network size and bandwidth amplify latency, jitter, and load imbalance, particularly under synchronized traffic patterns.

High fabric resilience. At the required network scale, intermittent faults, link flaps and persistent anomalies become steady-state operating conditions rather than exceptions [19, 28, 32, 37]. For AI clusters, this is not merely an availability concern: the network must remain bandwidth-proportional under failures and recover quickly enough that collective performance is not repeatedly perturbed.

Performance tuning at scale. The scale and dynamism of AI clusters make performance tuning and debugging inherently challenging. The interaction between routing, congestion control, topology, and workload behavior creates a high-dimensional optimization space. Rapid and precise localization of faults and anomalies is critical for maintaining performance and reducing operational overhead.

This paper presents **Spectrum-X** (SPX [24]), NVIDIA's **Spectrum** switches and **ConnectX** NIC's based AI solution architecture for large-scale Ethernet-based training clusters. At its core, SPX is a fully hardware-accelerated multi-plane network architecture with packet-granular load balancing. The key design principle is a separation of hardware-accelerated control loops by *scope*, *signal*, and *responsibility*. At the edge, NICs implement multi-plane load balancing together with per-plane congestion control. Inside each plane, switches perform per-packet adaptive routing to react to local imbalance and transient congestion at packet time-scales. Because the NICs and in-network control loops operate on different signals and at different scopes, they can be tuned independently: switch-local decisions continuously equalize path utilization within a plane, while the NIC can run a more aggressive congestion controller across planes without being destabilized by intra-plane path transients. This decomposition is central to achieving both high utilization and strong isolation.

SPX couples this transport architecture with fast fault handling and telemetry-driven operations. The multi-plane design provides fault containment and bandwidth scaling, while fast inter-plane failover absorbs transient and permanent faults with 3 ms recovery. The same architectural regularity that enables scale also aids operations: exploiting fabric symmetry together with high-frequency telemetry enables rapid localization of faults and performance anomalies. As we show in the paper, this design yields near-perfect isolation between co-executing collectives, whether from different training jobs or separate communication streams within the same job.

We comprehensively evaluate SPX on multiple 1K-GPU testbeds designed to serve as proxies for large-scale deployments, as well as in high-fidelity simulations with up to 256,000 endpoints. In addition, SPX is deployed in production across tens of customer clusters—including some of the largest AI clusters in the world—spanning a total of over 1 million GPUs.

SPX sustains 98% of theoretical line rate with p99 latency of just 8–9 μs. It demonstrates near-ideal isolation among concurrent collectives and full-scale LLM training workloads. It reaches theoretical maximum rate in complex intra-plane failures scenarios, and reacts within 3 ms to the theoretical maximum rate in case of dynamic failures. Last, it shows negligible performance degradation when running tightly coupled collectives over massive scale under aggressive link flap rates, both in the fabric and in the end-hosts.

Contributions. This paper's main contributions are:

- We identify the key challenges in networking for large-scale AI training, focusing on tail performance, scale, and failure resilience.
- We explain the principles and details of the Spectrum-X design, including its multi-plane architecture, hardware-accelerated adaptive routing and plane load balancing.
- We share lessons learned from deploying Spectrum-X across multiple large-scale clusters, including our performance tuning and development methodology using proxy-scale environments.
- We evaluate Spectrum-X using state-of-the-art benchmarks and production workloads, demonstrating strong performance and resilience.

2 Motivation

The rise of extreme-scale distributed AI exposes the fundamental limitations of existing network infrastructure in AI data centers, posing harsh capacity and performance requirements to achieve efficient training at scale as discussed next.

2.1 Low latency and jitter

It is well known that the Collective Completion Time (CCT) of synchronous collectives (AllReduce, AllGather, All2All) is determined by network stragglers [13] - individual slow flows that delay the CCT, and in turn collective performance affects the performance of the entire training run.

As network bandwidth scales beyond 800 Gbps, we observe that collectives are also becoming highly sensitive to network *latency*. Fig. 1a illustrates the impact of network latency on All2All collectives. Higher latencies significantly degrade the collective performance for message sizes commonly used in training collectives [6, 26]. Similar behavior is observed for Ring All-Reduce and other collectives.

Latency jitter, i.e., latency variations over time, is another significant performance factor. Jitter is a result of poor fabric performance isolation and load balancing. Intra-collective jitter arises due to congestion within a single collective, whereas inter-collective jitter occurs in a loaded network when neighboring traffic degrades victim throughput. Finally, jitter is

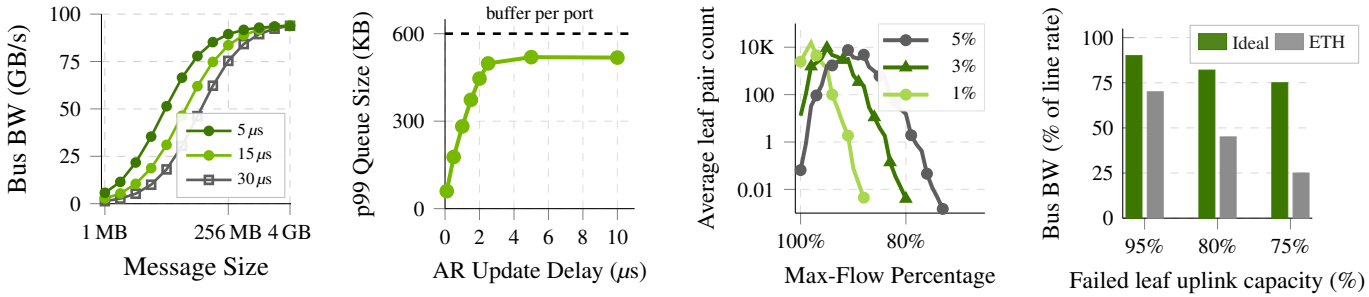


Figure 1: (1a) Impact of network latency on All2All collective (256-endpoint simulation). (1b) Impact of switch load balancing delay on queue size. (1c) Leaf-to-leaf max-flow distribution simulation. (1d) All2All bandwidth under partial uplink failure.

a result of variable queuing depth as the network alternates between underloaded and overloaded states.

We provide concrete production examples of such jitter and its diagnosis via bandwidth histograms and high-frequency telemetry in §5.2 and §5.3.

Maintaining jitter-free low latency in AI training workloads is hard: collectives generate coordinated traffic bursts that arrive simultaneously at switch ports. Thus, queue build-up and latency increase are inevitable, unless *the network fabric is able to quickly balance the load at sub-RTT timescales*.

The graph in Fig. 1b shows the impact of load balancing delay on the queue size in a switch that implements per-packet Join-Shortest-Queue load balancing among its 256 egress ports. For 100 ns delay the queues are small, however even at 1μs they grow almost 5-fold, and saturate at 2.5μs because at that point the decisions are random in practice. Such queues add about 20μs to the tail latency, and significantly impact the collective performance (Fig. 1a).

2.2 Network scale

Multiplane topologies [4, 26] address the challenge to scale networks to hundreds of thousands of nodes by decomposing each NIC bandwidth into multiple lower-speed interfaces (e.g., 4×200 Gbps from a single 800 Gbps ConnectX-8 NIC), each connecting into an independent *network plane*. Each plane is typically realized as a two-tier fat tree, but can also be extended to a multi-pod three-tier fat-tree as well. At this scale, the maximal size of a two-tier topology is increased to 128K, and the respective three-tier to 16M.

Multiplane topology, however, is only effective if the traffic is perfectly balanced across the planes. Achieving such a balance is challenging. One common approach is to partition collective operations across planes at the software level [21], assigning different message transfers to different planes through explicit scheduling in the collective library. Such approaches operate at RDMA message granularity, which is too coarse for optimal balance, and require software modifications.

Using per-packet spraying among the planes below the transport layer [33, 38] achieves better scheduling granularity. However, it uses a single transport CC loop, and assigns packets to planes without visibility into the per-plane congestion state. When a plane is degraded, i.e., due to fabric link flaps, the load symmetry among the planes is violated. Thus the performance of the load-oblivious spraying is dictated by the slowest plane. Worse, this approach dramatically increases the blast radius of the in-fabric link failure, which is a common place at such scales.

More critically, a transient plane failure due to a link flap at the host leads to a stall of the traffic from all the planes, since a large portion of the packets from all the flows get dropped. Host-based non-HW-accelerated approaches, have slow reaction to plane failure, and therefore their slow recovery incurs high performance loss as shown in Section 6.6.

Inter-plane load balancing must therefore react quickly and precisely to per-plane congestion and transient failures.

2.3 Time-to-AI

Minimizing *Time-to-AI*, i.e., the time from the moment GPUs are installed at the facility until they run a full-scale training job, is a crucial goal in modern AI training clusters. Unlike conventional hyperscale data centers that afford gradual growth over 1–2 years to reach capacity [39] the timescales for building modern-scale AI factories are over *an order of magnitude* shorter. For example, X-AI Colossus has achieved its full capacity in four months [5, 36].

Network fabric is perhaps the most challenging part to get up and running quickly, as it requires connecting hundreds of thousands of fragile optical cables. A non-negligible portion of these cables malfunction. For example, dust at the construction site might affect optical connectors, and manifest itself as link flaps, inability to establish connectivity, or high Bit Error Rate (BER). Such faulty links are proactively disabled to prevent them from degrading cluster performance.

Link failures are inevitable, but their effect on the cluster

performance should be proportional to the reduction in the physical connectivity. This is challenging, as failed links in the fabric may result in multiple source–destination pairs experiencing significant connectivity degradation, due to the combined impact of faults along the path.

Fig. 1c shows the maximum flow between leaf-pairs using a simulation of a 32K endpoint leaf-spine topology for different percentages of uniformly distributed random link failures. This degradation in capacity introduces extreme bandwidth asymmetry in the network that is especially present during early stages of cluster life and strongly affects the Time-to-AI. Fig. 1d measures the performance impact of such connectivity loss in an All2All collective running over a 480-endpoint H100 cluster. We systematically drop uplinks from a single leaf switch, varying the fraction of remaining uplink capacity (x-axis), and measure the achieved collective bandwidth. Traditional Ethernet solutions degrade in a non-proportional way to the bandwidth loss, particularly when asymmetry is severe, as is common in early-stage clusters.

3 SPX Overview

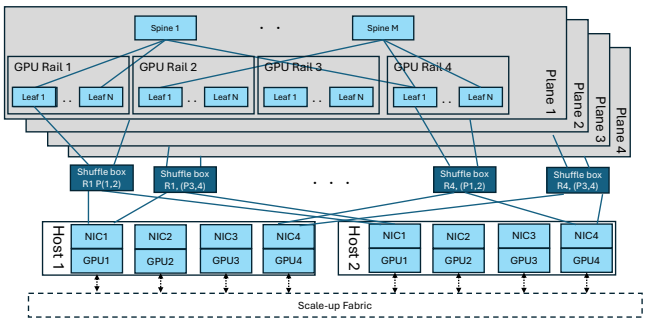


Figure 2: SPX topology overview: 2-level FT Multiplane rail-optimized topology for 4 disconnected planes. Larger, 3-level topologies keep rail separation in the spines, connecting on the 3rd tier.

Spectrum-X (SPX) is NVIDIA's reference architecture for scale-out GPU networks. It started from the Hopper generation (H100) [20] and evolved alongside next generations of Grace-Blackwell (GB200), Grace-Blackwell Ultra (GB300), and lately Vera-Rubin (VR200). This section focuses on the Spectrum-X architecture for the Blackwell Ultra generation. The discussion of the north-south [29], scale across [23], scale-up (NVLINK), storage [2], and computing technologies is out of scope.

SPX implements hardware-accelerated load balancing within the network via switch Adaptive Routing (AR), and across planes via the NIC Plane Load Balancer (PLB). Both mechanisms utilize fine-grained per-packet load balancing. Congestion in the network is controlled with combined lossless (PFC) and sender-based congestion control rate limiting mechanism, co-designed to preserve low tail latency at high load and present robust, linear performance with respect to available bisection bandwith under failures.

Packet-level load balancing schemes introduce out-of-order packet arrival [16], which the transport in SPX handles via direct data placement without buffering data in the receiver NIC and by reordering completions transparently to applications.

The non-blocking topology and Spectrum-X load-balancing features are designed to remain *job-allocation agnostic* inside the data center. They aim to provide consistent performance regardless of the physical alignment of the workloads to the collective patterns and locality, while ensuring performance isolation for neighboring jobs, simplifying the job-scheduling restrictions, and improving cluster utilization.

3.1 Network topology

Spectrum-X topology is based on a multi-plane design (Fig. 2), where planes are interconnected at the host side (e.g., via a shuffle-box), enabling high-BW fabrics with uniform link speeds.

Single plane network. We utilize leaf-spine or fat-tree topologies. Topologies are typically non-blocking and rail-optimized. We utilize *parallel links* between switches when building non-max-scale topologies to improve port utilization and achieve better resiliency to link failures.

Plane separation. In the SPX topology planes are disconnected, with each NIC connected to all planes, utilizing passive optical shuffle-boxes to reduce cabling. This design preserves full NIC-to-NIC reachability while exposing substantially more path diversity at the host edge. Plane separation is the key enabler for scaling a two-tier fabric to cluster sizes that typically required an additional network tier.

The multi-plane design requires a new load-balancing layer at the NIC. Load balancing decisions now span not only paths inside a plane, but also the choice of the plane itself. This NIC-level balancing mechanism is a first-order design consideration as discussed below.

3.2 Design principles

Hardware acceleration of per-packet load balancing. As link rates reach 800 Gbps and beyond, bandwidth-delay products grow while propagation delay – governed by the speed of light – remains fixed. Effective congestion avoidance requires reacting before in-flight data exceeds buffer capacity at the congestion point.

Spectrum-X targets three such congestion points with their respective congestion avoidance control loops:

- **Switch: fabric ports**, where per-packet adaptive routing prevents in-switch queue buildup;
- **Switch: endpoint ports**, where congestion control throttles senders before egress buffers saturate; and

- **NIC: plane ports**, where plane load balancing avoids cross-plane imbalance per packet.

Closing all three control loops within their BDP budgets is beyond the reach of software control paths. Therefore, hardware acceleration is a structural requirement of the fine-grain load balancing architecture, not a performance optimization.

Separate Problems, Separate Control Loops. Handling the three control loops simultaneously creates complex interplay between their respective controllers. A micro-burst at a fabric port raises the RTT that congestion control measures; this can be misread as endpoint incast and throttle the sender across all planes, or erroneously trigger plane load balancing to migrate traffic away from an otherwise healthy plane.

We address this by decomposing control into separate loops with distinct objectives, signals, and timescales:

- **Adaptive Routing:** stateless, per-packet load balancing reacting within hundreds of nanoseconds.
- **Congestion Control:** stateful, per-destination rate control at RTT timescales, triggered only by incast that cannot be resolved in-network.
- **Plane Load Balancing:** per-packet NIC-based balancing across planes, combining per-plane congestion signals with local queue feedback.

This separation of concerns avoids interference between the load balancing and congestion control loops and allows each to operate at its natural control horizon.

Appropriate state granularity for fast convergence. Fast fabric convergence, i.e., quickly determining the forwarding path and the sending rate given network conditions, is essential for high end-to-end performance.

SPX adopts stateless, flow-agnostic intra-plane load balancing and stateful, per-destination inter-plane load balancing. The former reacts at sub-RTT timescales to local imbalance without host involvement, converging quickly to a packet-granular solution within the plane; the latter is oblivious to the intra-plane paths, it leverages NIC acceleration and per-plane congestion control to adapt across a small number of planes within a few RTTs.

This design maintains minimal but sufficient state at the NIC: per-destination, per-plane congestion control. A single global controller would fail to adapt to inter-plane asymmetry.

Hardware-accelerated fault-tolerance. At massive scales, there is always some fraction of degraded links, some disabled permanently, some experiencing a transient disruption (a flap).

SPX strives to react as quickly as possible to *transient failures* and converge to the degraded capacity-proportional bandwidth, whereas the handling of *permanent failures* is pushed off the critical path. Therefore, handling transient failures is hardware-accelerated: adaptive routing excludes a locally failed fabric link within O(100) ns, and the plane load balancer stops sending on an inaccessible plane within a few RTTs. Permanent failures are handled at software timescales, and involve computing forwarding weights proportional to remaining healthy capacity (§4.4).

Handling asymmetries: stateless fabric, stateful planes. Failures introduce asymmetries across the network. Asymmetries between paths within the plane and between planes are inherently different. Intra-plane asymmetries are typically small due to high radix, while inter-plane asymmetries are larger due to the limited number of planes. For example, a failure of one leaf uplink out of a few hundred increases load on remaining uplinks by $< 1\%$, while a failure of a single plane out of four overloads the remaining planes disproportionately.

Whether a failure requires traffic redistribution depends on load: lightly loaded planes can absorb capacity loss, while heavily loaded ones cannot. Plane load balancing must therefore account for both topology and traffic.

Traffic composition introduces further asymmetry: a small fraction of control-plane traffic requires in-order delivery and cannot use per-packet load balancing. However, production telemetry shows that over 97% of traffic tolerates out-of-order delivery, so this constraint has minimal impact. These flows need no hardware acceleration, and they can be rate-limited, so the standard host network stack suffices.

Accordingly, we use stateless adaptive routing within the fabric to ensure scalability and avoid large per-path state within switches, and stateful per-plane control at the NIC to handle inter-plane imbalance efficiently.

Lossless Link Layer. A lossless fabric is preferred because per-packet load balancing introduces out-of-order delivery, making packet loss indistinguishable from reordering. Enforcing lossless operation removes this ambiguity: in a healthy network, losses are rare and typically confined to failure scenarios, thereby simplifying recovery.

PFC storms [18] are often cited as a common argument against lossless fabrics. We observe no evidence of such storms in our own operational experience across more than two years of production operation of SPX deployed on the largest AI clusters. Similar conclusions have been reported in other large-scale deployments [3, 6]. As a safeguard against misconfigurations and misbehaving devices, we deploy a PFC watchdog [31] that detects and mitigates PFC storms. Congestion caused by slow receivers, which can cause PFC storms, is handled by congestion control, which throttles senders before the congestion spreads through the network (§4.2).

4 Forwarding and load balancing

4.1 Per-packet Adaptive Routing

SPX switches implement per-packet Adaptive Routing (AR) via a quantized approximation of Join-Shortest-Queue (JSQ) [7, 38]: for each arriving packet, the switch scores every egress port in the ECMP group by its current queue depth, sampled at sub-microsecond intervals, and forwards the packet to one of the least-congested ports

We further extend this mechanism with *weighted-adaptive load balancing* [8, 40] to account for remote capacity imbal-

ance to a given destination caused by remote failures of links in the fabric, as described in §4.4.

4.2 Congestion Control

SPX congestion control is tailored for AI collectives, prioritizing short completion time over steady-state or extreme-incast convergence. It handles three types of congestion: incast, fabric congestion, and slow receivers.

Incast. Collective traffic is short, synchronized, and bursty: bandwidth-optimal collectives generate synchronized bursts across a few high-rate flows rather than sustained incast. Conventional CC algorithms (e.g., DCQCN [41], Swift [11]) either overreact to short synchronized collective bursts, throttling senders and failing to recover in a timely fashion within a single collective, or fail to react, leading to packet drops.

SPX CC handles the resulting endpoint congestion with RTT probes and sender-side rate control: the sender maintains a dynamic RTT target that guides its rate increases and decreases; the exact update rules are beyond the scope of this paper.

Fabric congestion. SPX topologies are non-blocking by design. While the fabric remains non-blocking, AR balances load across paths. However, inevitable fabric failures can make the network blocking; queues then grow beyond the level expected from AR alone, and CC must reduce the sending rate. To achieve this behavior, SPX CC combines RTT probes with explicit switch signals: ECN thresholds are configured to afford enough time for the AR logic to react to short-lived load imbalance. Thus, ECN marks only when load-balancing capacity is exhausted (Fig. 3), and CC reacts to fabric congestion only upon the resulting Congestion Notification Packets (CNPs). §4.4 describes how SPX handles fabric failures.

Slow receivers. Backpressure from the GPU or the intra-server interconnect can slow a receiver, building queues first in the NIC receive buffers and then in the network. RTT probes capture this queuing, so CC throttles senders to the slow receiver before PFC pauses propagate through the network and affect other flows. Switch and NIC buffer headroom is provisioned to give CC enough time to react before the PFC watchdog engages and falsely flags such cases as PFC storms. Persistent slow receivers generally result from misconfiguration or poor application behavior and must be fixed at their source.

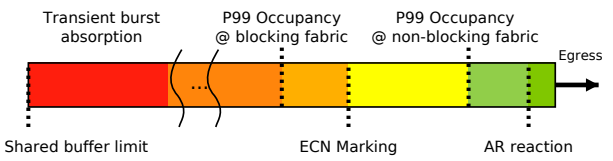


Figure 3: Congestion signaling and load balancing reaction points in switches (growing from right to left).

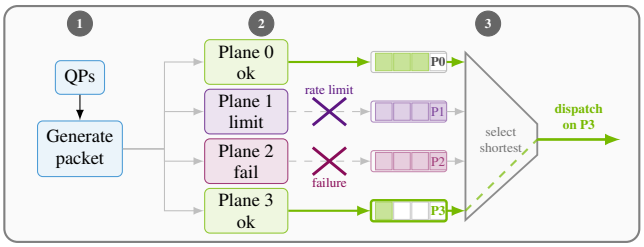


Figure 4: NIC per-packet plane selection: **(1)** generate packet from QPs, **(2)** query CC contexts and mask rate-limited or failed planes, **(3)** choose shallowest eligible egress queue.

4.3 Plane Load Balancing

Application Model. Multiplane is transparent to applications. Each NIC exposes a single RoCE device regardless of the number of underlying planes. Applications open queue pairs (QPs) on that device using standard RDMA verbs; all traffic distribution, load balancing, and plane resiliency are handled entirely in NIC hardware. Collective communication libraries such as NCCL and the transport layer require no modification.

Plane selection. Plane load balancing combines adaptive routing with end-to-end congestion control.

For each destination, the NIC maintains four independent CC contexts, one per plane, shared by all QPs to that destination. Each context independently issues RTT probes on its assigned plane and processes incoming CNPs to calculate its plane's rate allowance. This per-destination per-plane state lets the NIC distinguish congestion on one plane from healthy conditions on another, rather than reacting uniformly across all planes.

When a packet is ready for transmission, the NIC performs a two-stage hierarchical plane selection (Fig. 4). **Rate filter (E2E congestion):** The CC rate allowance for each plane is compared against the current transmission rate (computed based on transmission time of last packet burst). Planes whose allowance falls below that rate are excluded, yielding an eligible set of uncongested planes. **Local queue selection:** From the eligible planes, the NIC selects the plane with shallowest local egress queue, mirroring switch adaptive routing.

This hierarchy ensures that E2E congestion state takes precedence: congested planes are excluded before local queue depth is consulted. Thus, congestion on one plane lowers only that plane's allowance for the destination and diverts its traffic to the other planes, whereas endpoint incast congests all four contexts and throttles all traffic to that destination.

4.4 Resiliency

4.4.1 Endpoint failure and recovery

Recovery from host plane failures must be fast to avoid CCT degradation. Transient link flaps that resolve within O(ms) should be absorbed transparently by the network. Local host

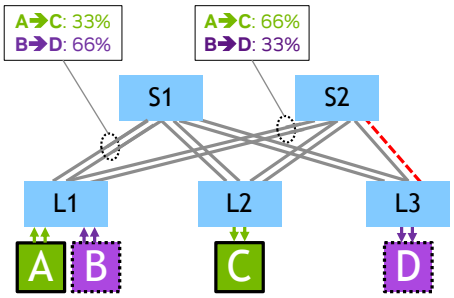


Figure 5: Asymmetrical load balancing. Weighted Adaptive Routing example in which remote destination D has reduced capacity. When traffic to D is active, the Adaptive Routing weights steer it towards S1, causing traffic to destination C to shift towards S2.

plane failures are detected directly by the NIC PLB, enabling immediate in-hardware reaction. Remote host plane failures are detected via consecutive RTT probe timeouts on that plane. Once a threshold is reached, the plane is removed from the eligible set and no new traffic is forwarded over it. Detection is fully handled in NIC hardware and completes within a few RTTs, prior to control-plane involvement.

4.4.2 Fabric link failures

Spectrum switch adaptive routing reacts to local fabric link failures in O(100 ns) leveraging fully hardware-accelerated mechanisms. In contrast, when symmetry is broken due to non-local *permanent* fabric failure, the response is handled via *Weighted Adaptive Routing (AR)*, which accounts for the effective bandwidth capacity of remote path to destinations. The weights are computed by a BGP-based control plane [14], which tracks topology changes and distributes weighted routing state. Once installed, the hardware AR engine combines local queue depth with remote weights, enabling traffic to shift away from degraded regions at packet-balancing timescales as illustrated in Fig. 5.

This separation allows fast, in-hardware reaction for local events, while relying on slower control-plane updates for global asymmetry.

5 Lessons learned in large deployments

Over two years of SPX production deployment, we have found *end-to-end observability* across compute, collectives, NIC, and switch to be essential. Our observations are confirmed by correlating direct step-time degradation with the identified respective root cause. We share representative examples below.

5.1 Debug experience of Adaptive Routing

Hardware-based AR provides a unique operational advantage: it produces a structurally symmetric traffic pattern in the fabric. Because AR distributes traffic evenly across links, any deviation from a perfect balance serves as a sign of a problem.

For example, consider the per-port BW of uplinks in a particular leaf switch. Fig. 6(a) shows a perfectly uniform traffic. (b) misconfigured NIC, switch, or collective lib, is immediately apparent by breaking the symmetry, as shown in Fig. 6(b).

Uniformity anomalies detected across multiple so-called *symmetry groups* such as leaf uplinks, rails and planes. Any outliers are likely indicators of hardware faults or software bugs.

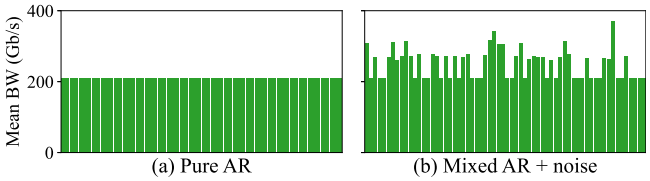


Figure 6: Leaf switch per-port uplink BW: (a) AR expected uniform pattern; (b) non-AR traffic interference.

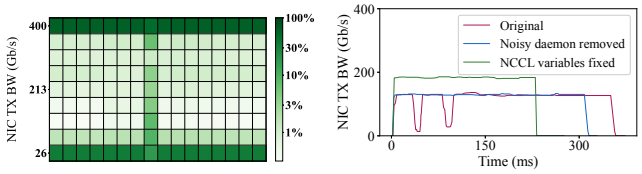
5.2 Catching stragglers via network indicators

In AI clusters, detecting anomalous low-performing nodes (stragglers) is crucial to maintaining high cluster utilization. We have found that a large category of issues, whether networking-related or not, is detectable using networking tools only. This coarse-grained approach works because identifying stragglers is more time-critical than diagnosing their precise root causes.

Tightly-coupled collectives split large transfers into sequences of small, dependent messages - each only a few MB and transmitted in O(10μs). A single straggler, fails to saturate the port BW and forces every other rank to stall at each message boundary. Consequently, while the straggler's BW fluctuates, all healthy participants exhibit a bi-modal BW distribution: either sending at line rate or idling. SPX's switch hardware exposes this intra-collective behavior through per-μs histograms, as shown in Fig. 7a.

5.3 Network debugging with HFT

High-Frequency Telemetry (HFT) serves as a microscope for network traffic, exposing dynamics that standard polling intervals miss entirely. This visibility is essential for studying short-lived dynamics, such as transient queue build-up, bandwidth dips, and CC throttling, that coarse-grained counters cannot capture. We generally consider streaming sampling intervals in the 100μs to 10ms range as HFT. In addition



(a) (b)

Figure 7: Debugging using high-frequency telemetry (HFT). a Per-NIC BW histogram with straggler. b HFT time-series debugging: daemon-induced BW drops and incorrect NCCL flags causing consistent BW under-utilization.

to the previously presented histograms, SPX support time-series HW accelerated HFT, from both NICs and switches, as demonstrated in this section.

For example, during the initial bringup, we discovered occasional drops in training benchmark performance (see Fig. 7b, top). HFT plots have shown sharp transient BW drops during the collective, which led us to isolate a periodic process that was hogging system resources, and stop it. We also saw the NIC did not reach full line rate (marked as "Noisy daemon removed" in the graph); this led us to review the application configuration, which contained an error in passing NCCL flags.

HFT was also vital in tuning congestion-control to accommodate the needs of ever-evolving workloads and topologies. Using switch HFT we can view egress queues, BW, and PFCs, all on the same timeline and at high frequency. Such metrics are essential for tuning congestion control mechanisms.

6 Evaluation

We evaluate SPX on a variety of benchmarks and scenarios. The main evaluation results are summarized in Tab. 1.

6.1 Methodology

Hardware Setup. We evaluate the performance of SPX on three generations of clusters (see Tab. 2).

Topologies are scaled down by replacing many sparsely populated switches with fewer fully populated ones connected by parallel links; for instance, 100 spines at 10% population become 10 fully populated spines with 10 parallel links, reducing hardware cost while preserving bisection bandwidth. This consolidation mirrors real sub-max-scale production deployments. It introduces additional challenges when evaluating link failures, which weighted-AR accounts for (§4.4).

Simulation. We use the NSX simulator [10] to evaluate network behavior at large scale. NSX is an event-driven, GPU-accelerated network simulator that models the core technologies of SPX, including multi-plane, adaptive routing, and congestion control. Its results are continuously validated against smaller development clusters, showing strong correlation in key performance metrics.

Workloads and scenarios. We define our experiments as combinations of workloads (*what* we test) and scenarios (*how* we test). Workloads cover: **RDMA bisection** microbenchmarks; **NCCL collectives** (e.g., AllGather, AllReduce, All2All); and **AI model** training. We apply these across three scenarios: **Baseline** (standalone, optimal performance), **Isolation** (victim workload under concurrent background traffic), and **Resiliency** (performance under emulated faults such as link failures or flaps).

Due to operational constraints, we do not evaluate all workloads and scenarios on all setups. Table 1 summarizes the combinations presented in this section. Clusters **Hopper_SP** and **Blackwell_SP** run an identical networking stack, which is also identical to the single-plane solution of **Blackwell_Ultra_MP** that adds multiplane load balancing. When running workloads that do not use *all* nodes, we trim down the fabric links to get a 1:1 symmetric non-blocking topology in order to not evaluate an under-subscribed network.

Metrics. We quantify performance using metrics tailored to each workload, focusing on measuring tail performance as discussed in §2, unless stated otherwise. For NCCL tests, we evaluate the *bus bandwidth* across varying message sizes. Bus bandwidth [22] is a collective-agnostic metric that normalizes inter-GPU communication speed, allowing us to compare achieved throughput directly against the theoretical hardware capacity, independent of the number of participating GPUs. The message size reflects the total data size that the collective operates on. Finally, for end-to-end AI workloads, we report the measured training step time.

Reference Solutions. In some experiments we compare the performance of **SPX** against a standard **Ethernet (ETH)** baseline representing traditional RoCEv2 deployments. The ETH configuration uses DCQCN congestion control and conventional ECMP.

6.2 Performance under high utilization

We evaluate the maximum sustained bandwidth and compare it to ETH. We stress-test a 64-node subset of the Hopper_SP cluster with a RDMA bisection benchmark. We measure the achieved bandwidth across GPU pairs across all participating nodes in a worst-case allocation pattern that forces all traffic to traverse a spine.

Fig. 8a shows ETH's bandwidth variability across pairs, with the median at 300 Gbps and some pairs collapsing to 25 Gbps, due to ECMP's static flow hashing. Conversely, SPX achieves a tight distribution. With a 1st percentile bandwidth of 377.23 Gbps (98% of the theoretical line rate), SPX ensures predictable, sustained throughput across all GPU pairs.

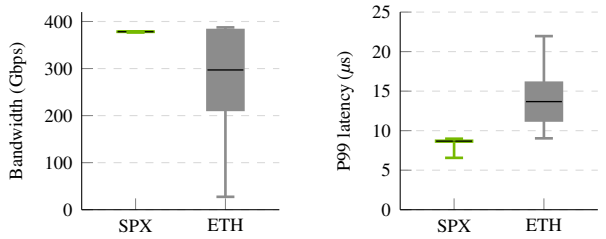
Fig. 8b compares P99 latency across pairs for both SPX and ETH at 300 Gbps (rate-limiting to match the median bandwidth sustained by ETH). We run `ib_send_lat` concur-

Category	Test	Platform	Key Result	Insight
Performance at High Utilization (§6.2)	RDMA bisection bandwidth p99 latency under 75% load	HSP	p01 BW is 98% of line rate 8–9 μs at 75% load	SPX provides consistently high bandwidth and low latency
Isolation (§6.3)	NCCL All2All with noise DeepSeek-V3 training with noise	BSP	No degradation Stable 668 ms/step	SPX offers robust isolation across concurrent workloads
Static Resiliency (§6.4)	RDMA under 10% link failures NCCL Collectives, 10% failures	HSP	-11% BW, $+7\%$ p99 LAT 3-10% from ideal	SPX's performance scales proportionally to hardware failures
Dynamic Resiliency (§6.5)	Single host link flap recovery LLM training during link flap	BUMP	<3 ms recovery No impact (fabric), <5% (host)	SPX achieves millisecond-scale reaction time to failures
Large-Scale Resiliency (§6.6)	Fabric link flap Host link flap	NSX	No visible P99 CCT impact Fast convergence is crucial	SPX maintains, at scale, close to optimal performance despite failures
Multiplane Load Balancing (§6.7)	Asymmetry: one-to-many, All2All State per-plane vs. per-path	BUMP	SPX: <3% degradation SPX $\sim$180 GB/s at 4 GB	SPX preserves fault-free performance under dynamic asymmetry

Table 1: SPX evaluation (HSP = Hopper_SP, BSP = Blackwell_SP, BUMP = Blackwell_Ultra_MP, NSX = NSX simulator).

Cluster	GPUs	Nodes	Planes	Topology	NIC
Hopper_SP	1024	128	1	3LFT (rail-opt)	CX7
Blackwell_SP	144	36	1	2LFT	CX7
Blackwell_Ultra_MP	1152	288	4	2LFT(rail-opt)	CX8

Table 2: Cluster configurations used in evaluation.



(a) BW distr. under max load (b) P99 Latency at 300Gbps

Figure 8: Performance under load: SPX vs. Ethernet (ETH).

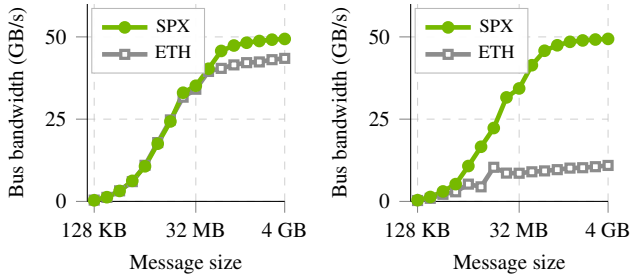


Figure 9: Single All2All (left); two concurrent All2All (right).

rently with `ib_write_bw` background traffic. ETH achieves the median P99 latency among pairs of $\sim$13 μs, with a very broad spread reaching 22 μs. These delays translate into severe bottlenecks for AI workloads (§6.3). In contrast, SPX maintains a low, tightly clustered p99 latency with a median of 8–9 μs, showing stable performance under load.

6.3 Performance isolation

Fig. 9 (left) shows that for isolated All2All collectives, ETH performs on par with SPX for small message sizes, but suffers at higher loads, peaking at 43 GB/s. In contrast, SPX achieves 49.3 GB/s (1.13×), nearly 99.5% of the theoretical 49.5 GB/s hardware capacity. This aligns with §6.2 as ETH cannot sustain maximum bandwidth under heavy load.

Fig. 9 (right) shows the impact of contention. Here we execute two All2All collectives in parallel, allocating 16 nodes to the "victim" workload and 48 nodes to the background "noise" collective. With ETH, the victim's bandwidth stagnates below 10.93 GB/s, about 80% performance drop. In contrast, SPX achieves near-perfect isolation.

Impact of background load on training step times. We run a DeepSeek-V3 LLM model training session on a 16-node subset of the Blackwell_SP cluster. The LLM nodes are distributed randomly uniform across the cluster. As shown in Fig. 10, in the standalone run SPX achieves an average training step time of 667 ms, whereas ETH averages 735 ms. Under background load of RDMA bisection noise on an additional 16 nodes, ETH suffers severe degradation, with step times increasing by 1.6× to 1.18 s. Conversely, SPX maintains the same step time of 668 ms, demonstrating robust performance isolation.

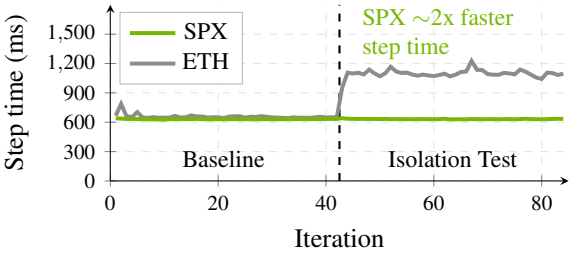


Figure 10: DeepSeek-V3 Isolation 16N NVL8 proxy model.

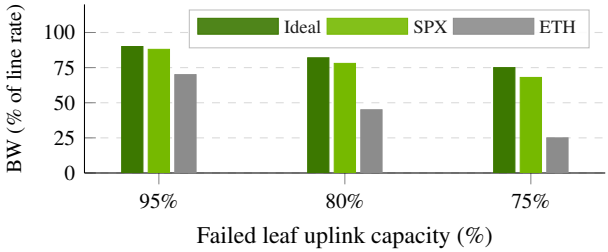


Figure 11: All2All performance under blocking fabric.

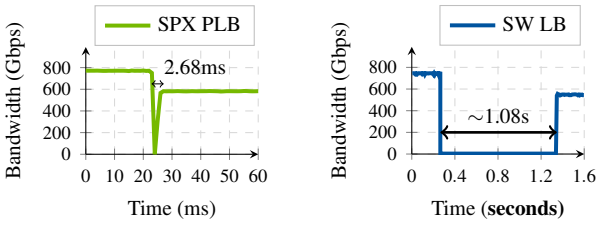


Figure 12: Endpoint single flap recovery in Blackwell_Ultra_MP with SPX hardware multi-plane (PLB) vs. software load balancer (SW LB).

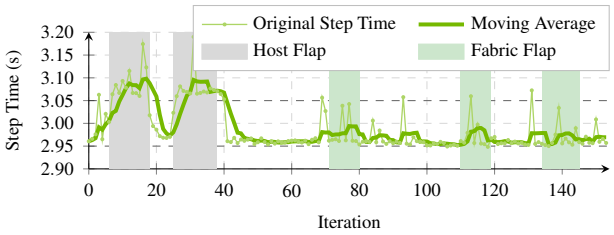


Figure 13: Nemotron 3 Ultra training under dynamic failures.

6.4 Static resiliency

We aim to show that the network performance degrades proportionally to the loss of bisection capacity. We configure a 60-node subset of the Hopper_SP cluster in a trimmed topology. We compare a failure-free, pristine baseline against a degraded network with leaf-to-spine links that are permanently disabled matching the max-flow capacity seen between different leaf pairs at early stages of a cluster's life (§ 2.3).

We execute two types of workloads. First, to analyze the impact of failures under maximum utilization, we reproduce the RDMA bisection test from §6.2. We run `ib_write_bw` across GPU pairs in the same rail to generate bisection traffic, while concurrently measuring the p99 tail latency across pairs using `ib_send_lat`. Second, we run NCCL All2All collective benchmark from §2.3.

RDMA. Our results show that under the degraded fabric with 10% less links, the p01 bandwidth gracefully degrades by 11% from 377.80 Gbps to 335.31 Gbps, closely tracking the actual physical network capacity. Similarly, the p99 latency shows an increase from 14.97 μs to 15.96 μs.

NCCL collectives. Fig. 11 depicts link failure percentages and the performance impact in an All2All collective, as described in §2.3. While traditional Ethernet solutions degrade in a non-proportional way to the bandwidth loss, SPX maintains 3-10% of the ideal solution bandwidth.

6.5 Dynamic resiliency

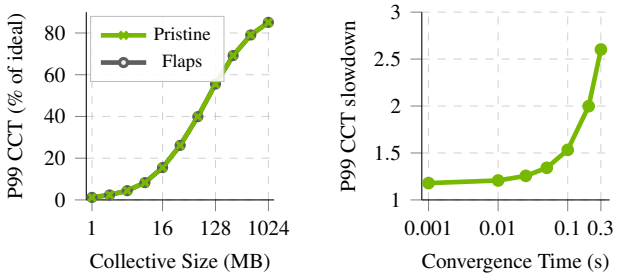
We compare the performance of SPX's hardware-accelerated Plane Load Balancer (PLB) to a software-based NCCL reference solution (SW LB) and a single-plane configuration.

Recovery time for a single host link flap. We execute `ib_write_bw` between node pairs and inject a link failure in a single host link, failing one plane. In the single-plane case, the RDMA connection crashes and communication halts entirely. Fig. 12 (left) shows the plane failover under SPX. The flow initially sustains line-rate bandwidth. Upon link failure, bandwidth drops to zero as packets on the failed plane are lost, triggering retransmissions. However, in under 3 ms, PLB redistributes traffic across the three other planes and reaches 75% of the original line rate. Since the software load balancer operates above the NCCL layer, we test its performance using a SendReceive benchmark across node pairs. Fig. 12 (right) shows that SW LB requires 1.08 *seconds* to recover, about 400× slower than hardware PLB.

Link flaps impact on LLM training. We inject link flaps during a training job and measure the impact on step time. We train Nemotron 3 Ultra on a 64-node subset of the Blackwell_Ultra_MP cluster. Fig. 13 shows the measured step times. We first inject two consecutive failures on a host-to-leaf link, affecting one of the four planes (gray shaded areas). SPX falls back to the three healthy planes in a single iteration. Once the link recovers, SPX instantly restores traffic to all four planes, returning to the 2.95s baseline. We later inject *three independent flaps* on a leaf-to-spine uplink (green shaded regions). These have a negligible impact on step time stability, showing SPX's robust dynamic resiliency also at the fabric-tier.

6.6 Large-scale resiliency

In this section we evaluate the expected SPX performance under failures in massive-scale clusters. As we do not have



(a) Normalized P99 CCT, 64K nodes, with and without fabric flaps

(b) P99 CCT slowdown over pristine, 256K nodes, as function of access link failure convergence

Figure 14: Large-scale fault tolerance simulation.

access to such clusters, we simulate them using NSX [10].

Fabric flaps. We generate random link flaps using a fixed value of Mean Time Between Failures (MTBF) per link corresponding to 10 flaps/minute in a 64K single plane two level fat tree cluster. Upon flap the respective switch port is locally disabled for a *flap duration*, 10 seconds, and immediately re-enabled. The control plane is not aware of the flap. The MTBF values and flap duration are derived from conservative worst-case estimates from real large-scale clusters deployments. The workload is 256 Ring-AllGather/ReduceScatter collectives, each 256 ranks. We aim to estimate P99 CCT among all these collectives. This value represents a good proxy for the expected performance in a large-scale training session over all cluster GPUs.

The fabric experiences performance degradation as a function of the number of concurrent failed links due to the flap, which has a Poisson distribution defined by the flap frequency and duration. Consequently, we run the simulations of this workload while varying the number of concurrent link failures from 1 to 10, record the respective P99 CCT for each, and obtain the expected P99 CCT of each collective as a weighted sum of these values according to their probabilities.

Fig. 14a shows the results normalized by the ideal CCT rate, with and without flaps. The curves overlap, indicating no visible performance impact.

End point link flap. We simulate a multi-plane cluster with 256K GPUs and 4 planes. We aim to evaluate the cluster performance as a function of the fabric convergence time, i.e., the time it takes the NIC to converge to the degraded bandwidth over three planes. Until the convergence, we assume that the traffic over the faulty access link gets dropped, which mirrors the real hardware behavior. We measure P99 CCT among 1024 collectives each 256 ranks.

For this simulation, we consider each NIC to be in one of the three state: pristine (no failed planes), failed (one failed plane before convergence), degraded (one failed plane after convergence to 75% of the line rate of four planes). We evaluate the latency of a single collective assuming one failed NIC, and record the performance of the collective in the pristine, failed and degraded state.

To evaluate the full workload, we generate failure events in the 256K cluster using the same MTBF as before. Every fault is followed by a convergence event after the convergence time we evaluate, and then a plane restoration event after 10 seconds, matching the flap duration. We run multiple iterations of each set of 1024 collectives, and calculate the collective's performance separately according to the state of the NICs it uses, using the network simulation results. We assume one failure per ring at any point. We estimate P99 CCT over all collectives in the same iteration, and average it over all iterations in the trace.

The MTBF and flap duration are the same as in the fabric. We note that such a flap rate is very high and was observed only in a small fraction of the cluster time.

As shown in Figure 14b the P99 CCT increases by 20% due to plane bandwidth loss, as long as the convergence is within 10 milliseconds. This is inevitable, since there is at least one fault affecting at least one collective. Higher convergence causes CCT increase by 53% at 100 msec and 260% at 300 msec, highlighting the crucial importance of fast convergence in large-scale training.

6.7 Multiplane load balancing

We evaluate the core design decisions behind SPX's Plane Load Balancer (PLB), focusing on how per-plane congestion control (CC) manages network asymmetry, both in load distribution and reaction time, and demonstrating the importance of decoupled control loops (cf. §3.2).

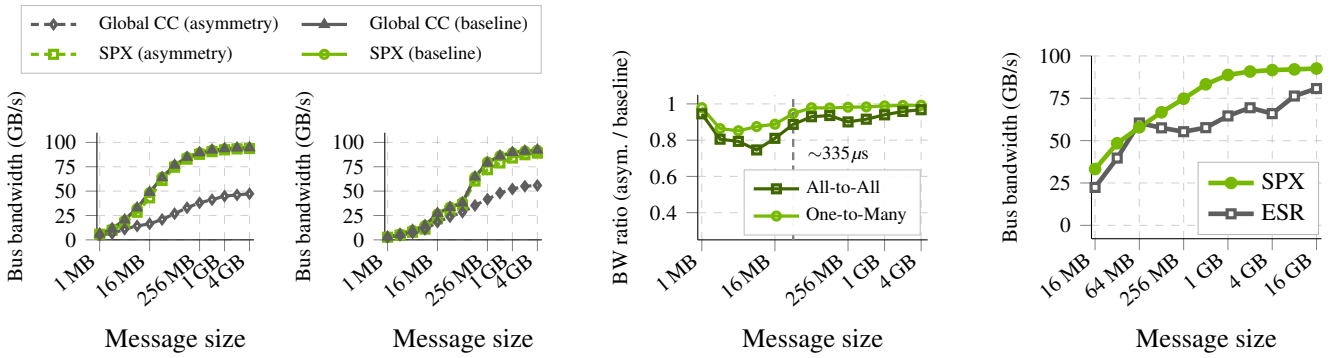
No degradation under dynamic asymmetry. Dynamic asymmetry occurs when some plane(s) becomes degraded and gets saturated, whereas other planes still have spare capacity. Such scenarios are challenging for load balancing.

We configure a 16-NIC-per-leaf subset of the Blackwell_Ultra_MP cluster (Fig. 16), trimming the uplinks on Leaf 2 (plane 2) and Leaf 3 (plane 3) from a non-blocking 16×200 Gbps to 4×200 Gbps. We run the main workload on 8 NICs per leaf and noise on the remaining 8 NICs.

We execute the main workload in two modes: *one-to-many* (NICs under Leaf 1 send repeated bursts to hosts under Leaf 2 and Leaf 3) and *All2All* executed by all NICs. Both workloads stress the separation of control loops: the CC must throttle per-destination rates under incast pressure, while the PLB must simultaneously divert traffic away from the degraded planes. These two reactions must proceed independently to avoid interfering with each other's signals.

We establish a baseline by executing the tests without asymmetry, and compare SPX PLB against **Global CC**, a variant using a single shared CC context across all planes.

Fig. 15 shows the results. Under uniform fabric SPX and Global CC perform identically, confirming that per-plane state introduces no overhead. Under asymmetry, however, they di-



(a) One-to-Many (b) All-to-All (c) Normalized SPX BW (d) SPX vs. ESR

Figure 15: Collective bandwidth under noise-induced asymmetry (SPX green, Global CC gray; solid: symmetric baseline, dashed: asymmetry); normalized SPX bandwidth ratio (dashed line: ~335 μs convergence); All2All SPX vs. Entropy Source Routing.

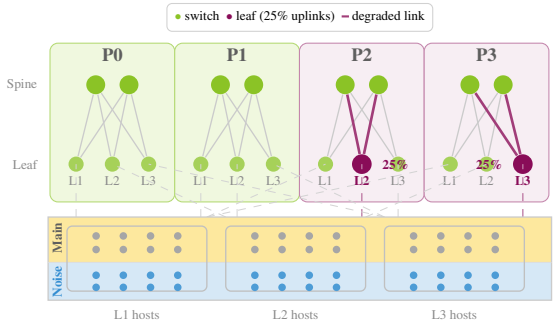


Figure 16: Multiplane load balancing testbed. Four planes (P0–P3), each with 2 spines and 3 leaves (L1–L3). P2/L2 and P3/L3 degraded to 25% uplinks (red). Hosts grouped by leaf type in three 16-NIC blocks (8 Main / 8 Noise each).

verge: without per-plane CC, Global CC's bandwidth collapses from 94.5 GB/s to 47.3 GB/s (a $>$ 50% drop) for the one-to-many workload, and from 92.2 GB/s to 55.9 GB/s (a $>$ 40% drop) for All2All. In contrast, SPX isolates the congestion, sustaining 93.7 GB/s (one-to-many) and 89.0 GB/s (All2All) for 4 GB messages. We see near-baseline performance despite the 75% uplink reduction on two leaf switches.

Fast convergence within critical message size ranges. Fig. 15c plots SPX's normalized bandwidth under dynamic asymmetry. For messages below ~4 MB, the PLB has not yet accumulated sufficient per-plane congestion signals to detect the imbalance, yielding normalized performance ratios as low as 0.85 for one-to-many and 0.75 for All2All. Since the background noise is continuous, the PLB state resets between bursts and must re-learn at each iteration. However, for message sizes above ~335 μs threshold (32 MB), PLB fully converges, allowing SPX to sustain $>$ 97% and $>$ 90% of its baseline bandwidth for the one-to-many and All2All workloads, respectively.

Conflicting control loops: entropy-based source routing. We assess the importance of decoupled control loops by comparing SPX's PLB to **entropy-based source routing (ESR)**. In ESR, entropy values jointly encode both the target plane and the intra-plane path, fundamentally entangling congestion control (CC) and load balancing. As a result, the aggregate CC state cannot provide independent steering signals for plane selection and intra-plane routing. We measure the impact of this entanglement by running four concurrent All2All collectives across 32 nodes with NVLink disabled (scale-out network traffic only). As shown in Fig. 15d, SPX scales smoothly to ~92 GB/s. In contrast, ESR exhibits oscillating throughput (55–80 GB/s above 256 MB) due to its conflicting control loops. These results confirm that per-plane CC design is beneficial for multiplane load balancing.

7 Related work

Production large-scale RoCE and AI fabrics. This paper introduces a novel hardware-accelerated multi-plane architecture deployed in production large-scale clusters, complementing prior works on RDMA-over-Ethernet and AI-training fabrics at hyperscale, including Meta [6], Microsoft [3] and Alibaba [15, 26], and Tencent's Astral with its same-rail tier-2 interconnect for half-a-million-GPU training [17].

Multi-path RDMA transport and packet spraying. A parallel line of work pursues multi-path load balancing at the transport layer. The Ultra Ethernet Consortium's UET specification [33] standardizes entropy-based per-packet spraying, Google's PLB [27] uses host-side congestion signals for exploiting multiple paths. These schemes react at multi-RTT, host-controlled timescales and typically maintain a single congestion-control loop with no per-path visibility. SPX takes a different approach of hardware-accelerated in-network load balancing and control loop separation.

In-network load balancing and adaptive routing. Fine-grained in-network load balancing has been studied extensively [1, 7, 25, 30, 34, 38], Weighted multi-path forwarding

under capacity asymmetry has been addressed by WCMP [40] and, more recently, by Juniper's weighted packet spray for AI/ML fabrics [8, 9], which bias per-packet forwarding by static link weights but do not specify composing those weights with queue-aware adaptive port selection. SPX introduces many novel components, taking adaptive routing to extreme with hardware-accelerated multi-plane design.

Congestion control. SPX builds on a well-established RDMA CC design space, including ECN-based rate control (DCQCN [41]), delay-based signaling (Swift [11]), in-network telemetry (HPCC [12]), and revisited loss-recovery assumptions (IRN [18]). We show how to combine these with adaptive routing to optimize for AI training workloads, keeping the control loop tuned to sustained collective congestion rather than transient micro-bursts that can be balanced in the fabric.

8 Conclusions

We described Spectrum-X, a field-tested Ethernet fabric running in multiple large-scale AI training clusters. Following our rich deployment experience, four takeaways stand out. First, topology matters as much as transport. Given the growing demand for scale, building shallow multiplane topologies with high-radix switches is key to achieving better clusters across every critical dimension: performance, fault tolerance, power, cost, and cabling complexity. Second, hardware acceleration with clear signal and control loop separation is what makes the fabric work at 800 Gbps and beyond. Host-based or software-driven alternatives failed in our evaluations to provide the convergence time that synchronous collectives demand. Third, operating a fabric at this scale depends on the visibility and ease-of-tuning. Last, validating cluster and networking technology at this scale requires a well-defined set of benchmarks with clear KPIs that can be run consistently across production clusters, proxy-scale testbeds, and simulators; defining and standardizing this benchmark set is, in our view, a prerequisite for healthy progress in the field.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. We also thank our colleagues at NVIDIA, including those in the Networking Software, Networking Research, Networking Software & Systems Architecture, and Networking Product Marketing organizations, for their contributions to the development of NVIDIA Spectrum-X.

References

[1] 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 *ACM SIGCOMM*, 2014.

[2] Taylor Allison. Accelerating AI Storage by up to 48% with NVIDIA Spectrum-X Networking Platform and Partners. https://developer.nvidia.com/blog/accelerating-ai-storage-by-up-to-48-with-nvidia-spectrum-x-networking-platform-and-partners/, 2025.

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

[4] DeepSeek-AI. DeepSeek-V3 Technical Report, 2024.

[5] Doug Eadline. xAI Colossus: The Elon Project. https://www.hpcwire.com/2024/09/05/xai-colossus-the-elon-project/, 2024.

[6] 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 *ACM SIGCOMM*, 2024.

[7] Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. DRILL: Micro Load Balancing for Low-latency Data Center Networks. In *ACM SIGCOMM*, 2017.

[8] Juniper Networks. Networking the AI Data Center: Advanced Load Balancing (DLB, GLB, Weighted ECMP) for AI/ML Fabrics. https://www.juniper.net/content/dam/www/assets/white-papers/us/en/networking-the-ai-data-center.pdf, 2024.

[9] Juniper Networks. Weighted Packet Spray for Dynamic Load Balancing (Junos OS Evolved AI/ML Guide). https://www.juniper.net/documentation/us/en/software/junos/ai-ml-evo/topics/topic-map/weighted-packet-spray.html, 2024.

[10] Sajy Khashab, Hariharan Sezhiyan, Rani Abboud, Alex Normatov, Stefan Kaestle, Eliav Bar-Ilan, Mohammad Nassar, Omer Shabtai, Wei Bai, Matty Kadosh, Jiarong Xing, Mark Silberstein, T. S. Eugene Ng, and Ang Chen. NSX: Large-Scale Network Simulation on an AI Server. In *ACM NAIC*, 2025.

[11] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *ACM SIGCOMM*, 2020.

[12] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. HPCC: High Precision Congestion Control. In *ACM SIGCOMM*, 2019.

[13] Jinkun Lin, Ziheng Jiang, Zuquan Song, Sida Zhao, Menghan Yu, Zhanghan Wang, Chenyuan Wang, Zuocheng Shi, Xiang Shi, Wei Jia, Zherui Liu, Shuguang Wang, Haibin Lin, Xin Liu, Aurojit Panda, and Jinyang Li. Understanding stragglers in large model training using what-if analysis. In *USENIX OSDI*, 2025.

[14] Stephane Litkowski, SATYA R MOHANTY, Arie Vayner, Akshay Gattani, Ajay Kini, Jeff Tantsura, and Reshma Das. BGP link bandwidth extended community use cases. Internet-Draft draft-ietf-bess-ebgp-dmz-10, 2026.

[15] Jie Lu, Jiaqi Gao, Fei Feng, Zhiqiang He, Menglei Zheng, Kun Liu, Jun He, Binbin Liao, Suwei Xu, Ke Sun, Yongjia Mo, Qinghua Peng, Jilie Luo, Qingxu Li, Gang Lu, Zishu Wang, Jianbo Dong, Kunling He, Sheng Cheng, Jiamin Cao, Hairong Jiao, Pengcheng Zhang, Shu Ma, Lingjun Zhu, Chao Shi, Yangming Zhang, Yiquan Chen, Wei Wang, Shuhong Zhu, Xingru Li, Qiang Wang, Jiang Liu, Chao Wang, Wei Lin, Ennan Zhai, Jiesheng Wu, Qiang Liu, Binzhang Fu, and Dennis Cai. Alibaba Stellar: A New Generation RDMA Network for Cloud AI. In *ACM SIGCOMM*, 2025.

[16] Sarah McClure, Evyatar Cohen, Alex Shpiner, Mark Silberstein, Sylvia Ratnasamy, Scott Shenker, and Isaac Keslassy. Load Balancing for AI Training Workloads, 2026.

[17] Qingkai Meng, Hao Zheng, Zhenhui Zhang, ChonLam Lao, Chengyuan Huang, Baojia Li, Ziyuan Zhu, Hao Lu, Weizhen Dang, Zitong Lin, Weifeng Zhang, Lingfeng Liu, Yuanyuan Gong, Chunzhi He, Xiaoyuan Hu, Yinben Xia, Xiang Li, Zekun He, Yachen Wang, Xianneng Zou, Kun Yang, Gianni Antichi, Guihai Chen, and Chen Tian. Astral: A Datacenter Infrastructure for Large Language Model Training at Scale. In *ACM SIGCOMM*, 2025.

[18] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for RDMA. In *ACM SIGCOMM*, 2018.

[19] Pooria Namyar, Arvin Ghavidel, Daniel Crankshaw, Daniel S. Berger, Kevin Hsieh, Srikanth Kandula, Ramesh Govindan, and Behnaz Arzani. Enhancing Network Failure Mitigation with Performance-Aware Ranking. In *USENIX NSDI*, 2025.

[20] NVIDIA. NVIDIA Launches Accelerated Ethernet Platform for Hyperscale Generative AI (Spectrum-X). https://nvidianews.nvidia.com/news/nvidia-launches-accelerated-ethernet-platform-for-hyperscale-generative-ai, 2023.

[21] NVIDIA. NCCL Developer Guide and Environment Variables. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html, 2024.

[22] NVIDIA. NCCL Tests: Performance — Bus Bandwidth Metric. https://github.com/NVIDIA/nccl-tests/blob/master/doc/PERFORMANCE.md, 2024.

[23] NVIDIA. NVIDIA Introduces Spectrum-XGS Ethernet to Connect Distributed Data Centers Into Giga-Scale AI Super-Factories. https://nvidianews.nvidia.com/news/nvidia-introduces-spectrum-xgs-ethernet-to-connect-distributed-data-centers-into-giga-scale-ai-super-factories, 2025.

[24] NVIDIA. NVIDIA Spectrum-X Networking Platform. https://www.nvidia.com/en-us/networking/spectrumx/, 2026.

[25] Chenchen Qi, Wenfei Wu, Yongcan Wang, Keqiang He, Yu-Hsiang Kao, Zongying He, Chen-Yu Yen, Zhuo Jiang, Feng Luo, Surendra Anubolu, Yanjin Gao, Bingfeng Lin, Wenda Ni, Yiming Yang, Donglin Wei, Boyang Zhou, Jian Wang, and Shan Ding. SGLB: Scalable and Robust Global Load Balancing in Commodity AI Clusters. In *ACM SIGCOMM*, 2025.

[26] 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 *ACM SIGCOMM*, 2024.

[27] 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 *ACM SIGCOMM*, 2022.

[28] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C. Snoeren. Passive Realtime Datacenter Fault Detection and Localization. In *USENIX NSDI*, 2017.

[29] Shashank Sabhlok. North–South Networks: The Key to Faster Enterprise AI Workloads. https://developer.nvidia.com/blog/north-south-networks-the-key-to-faster-enterprise-ai-workloads/, 2025.

[30] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. Network Load Balancing with In-network Reordering Support for RDMA. In *ACM SIGCOMM*, 2023.

[31] SONiC Project. PFC watchdog design. https://github.com/sonic-net/SONiC/wiki/PFC-Watchdog-Design, 2018.

[32] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. NetBouncer: Active Device and Link Failure Localization in Data Center Networks. In *USENIX NSDI*, 2019.

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

[34] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *USENIX NSDI*, 2017.

[35] Ertza Warraich, Omer Shabtai, Khalid Manaa, Shay Vargaftik, Yonatan Piasetzky, Matty Kadosh, Lalith Suresh, and Muhammad Shahbaz. OptiReduce: Resilient and Tail-Optimal AllReduce for Distributed Deep Learning in the Cloud. In *USENIX NSDI*, 2025.

[36] xAI. Colossus: The World's Largest AI Supercomputer. https://x.ai/colossus, 2024.

[37] Zhiyi Yao, Pengbo Hu, Congcong Miao, Xuya Jia, Zuning Liang, Yuedong Xu, Chunzhi He, Hao Lu, Mingzhuo Chen, Xiang Li, Zekun He, Yachen Wang, Xianneng Zou, and Junchen Jiang. Holmes: Localizing Irregularities in LLM Training with Mega-scale GPU Clusters. In *USENIX NSDI*, 2025.

[38] Eitan Zahavi, Isaac Keslassy, and Avinoam Kolodny. Distributed adaptive routing for big-data applications running on data center networks. In *ACM/IEEE ANCS*, 2012.

[39] Shizhen Zhao, Rui Wang, Junlan Zhou, Joon Ong, Jeffrey C. Mogul, and Amin Vahdat. Minimal Rewiring: Efficient Live Expansion for Clos Data Center Networks. In *USENIX NSDI*, 2019.

[40] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. In *EuroSys*, 2014.

[41] 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 *ACM SIGCOMM*, 2015.