RDMA over Ethernet for Distributed AI Training at Meta Scale

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, Hongyi Zeng
Meta

ABSTRACT

The rapid growth in both computational density and scale in AI models in recent years motivates the construction of an efficient and reliable dedicated network infrastructure. This paper presents the design, implementation, and operation of Meta's Remote Direct Memory Access over Converged Ethernet (RoCE) networks for distributed AI training.

Our design principles involve a deep understanding of the workloads, and we translated these insights into the design of various network components: **Network Topology** - To support the rapid evolution of generations of AI hardware platforms, we separated GPU-based training into its own "backend" network. **Routing** - Training workloads inherently impose load imbalance and burstiness, so we deployed several iterations of routing schemes to achieve near-optimal traffic distribution. **Transport** - We outline how we initially attempted to use DCQCN for congestion management but then pivoted away from DCQCN to instead leverage the collective library itself to manage congestion. **Operations** - We share our experience operating large-scale AI networks, including toolings we developed and troubleshooting examples.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Data center networks**; **Network protocols**; **Traffic engineering algorithms**.

KEYWORDS

RDMA, Distributed Training

ACM Reference Format:
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, Hongyi Zeng. 2024. RDMA over Ethernet for Distributed AI Training at Meta Scale. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3651890.3672233

1 INTRODUCTION

The growing prevalence of Artificial Intelligence (AI) in fields like image recognition, natural language processing, and recommendation systems has introduced a new era of communication demands. Distributed training, in particular, imposes the most significant strain on data center networking infrastructure. For instance, a typical generative AI job may necessitate tight coordination of thousands of GPUs over the course of several weeks. Constructing a reliable, high-performance network infrastructure capable of accommodating this burgeoning demand necessitates a reevaluation of data center network design.

The inter-GPU communication in distributed training workloads typically comprises two stages. In each training node, housing 4-8 GPUs, these GPUs are interconnected using high-speed transports like NVLink[26], commonly known as "intra-node communication." When a training job requires additional GPUs, the "inter-node communication" will happen across a network. The industry commonly employs two design approaches for this inter-node communication. One utilizes standard TCP/IP or modified socket implementations (e.g., fastsocket[7]). However, this method is susceptible to performance degradation because of CPU overhead and increased latency. The second design approach involves proprietary interconnects, such as InfiniBand[23], NVSwitch[26], Elastic Fabric Adaptor[2], and Inter-rack ICI[8]. Although these methods deliver significantly improved performance, their proprietary nature restricts their deployment flexibility.

When Meta introduced distributed, GPU-based training, we decided to construct specialized data center networks tailored for these GPU clusters. We opted for RoCE[11] as the inter-node communication transport. Remote Direct Memory Access (RDMA) is a method to access the remote memory without involving the CPU. The choice of RoCE was motivated by:

1) RoCE adheres to the established RDMA verbs semantics, well-known to the training workload community. This ensured a seamless transition for existing training applications and facilitated the development of new ones.

2) By utilizing Ethernet, we could employ a significant portion of our existing data center design components and tools. This enabled us to build the network using a largely consistent Clos-based design, and it simplified our operations by reusing existing tools.

3) The entire stack is grounded in open standards and offers support from multiple vendors, ensuring compatibility and flexibility in our network infrastructure.

We gained significant experience in designing and operating our RoCE network as follows:

Dedicated backend network: We built a dedicated backend network specifically for distributed training. This allowed us to evolve, operate, and scale independently from the rest of the data center network. To support Large Language Models (LLMs), we expanded the backend network towards the DC-scale, e.g., incorporating topology-awareness into the training job scheduler.

Evolution of routing schemes: Given the poor performance of the default ECMP (Equal-Cost Multi-Path) routing and its alternatives

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia
© 2024 ACM.
ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672233

Collectives	Model Dist. purpose	Traffic pattern	Network congestion	GPU Msg (MB)	NIC Msg (KB)	Flow entropy per NIC	Topology need
`AlltoAll(v)`	Embedding distribution	full mesh with imbalanced traffic	N-to-N with possible incast	1	128	$\log(M * N)$	Full bisection bandwidth
`AllReduce`	DDP	Tree or Ring	2-to-1 incast for Tree	4 (Tree) 1 (Ring)	512	$\log(M)$	Tolerate over-subscription
`AllGather`	FSDP	Ring	1-to-1 low congestion	1	512	$\log(M)$	Tolerate over-subscription
`ReduceScatter`	FSDP	Ring	1-to-1 low congestion	1	512	$\log(M)$	Tolerate over-subscription

Table 1: Transport implications from model and collectives for large collective sizes. M is number of channels used in NCCL. N is number of RDMA NICs. Full-mesh implementation assumed for `AlltoAll`

in our initial stages, we deployed a combination of centralized traffic engineering and an Enhanced ECMP scheme to achieve optimal load distribution for training workloads.

Congestion control for collectives: We found that it is very challenging to tune DCQCN, the mainstream congestion control scheme used in RoCE deployments, to gain significant benefit in collective completion time for distributed training tasks. Instead, we have designed a receiver-driven traffic admission via the collective library to achieve superior performance. This involves co-tuning of both the collective library configuration as well as the underlying network configuration to achieve optimal performance.

We have successfully expanded our RoCE networks, evolving from prototypes to the deployment of numerous clusters, each accommodating thousands of GPUs. These RoCE clusters support an extensive range of production distributed GPU training jobs, including ranking, content recommendation, content understanding, natural language processing, and generative AI model training, among other workloads. We have previously shared high-level designs of RoCE clusters that support up to 32,000 GPUs.[18]

While RoCE has been a subject of networking research before [3, 6, 9, 19], its dominant application has historically been within storage networks. There has been a dearth of literature addressing RoCE's deployment in large-scale AI training scenarios. In this paper, we delve deeply into the intricacies of scaling the RoCE network to interconnect thousands of GPUs in each cluster. Our experience shows that, by understanding the workload well and by carefully designing each component of the network, including topology, routing, transport, and operational workflows, RoCE can support AI training at scale.

This work does not raise any ethical issues.

2 BACKGROUND

2.1 Distributed Model Training

Distributed training is scaled by sharding the model and/or the input data across multiple GPUs, using various parallelism strategies. A typical training process involves repeating training iterations. Each iteration consists of a forward pass to generate losses, a backward pass to compute gradients, and an optimizer step to update parameters. The GPUs involved need to synchronize either the gradients or the updated parameters multiple times within each iteration. This synchronization process requires the transfer of large amounts of data among the GPUs within milliseconds, repeated over multiple iterations until the model converges. This necessitates a network capable of delivering high bandwidth and low, predictable latency.

2.2 RoCE and Collective Communication

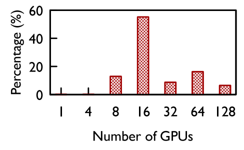
RDMA is an industry standard on hardware-assisted communication acceleration. RDMA implements "verbs" APIs such as `read` and `write`. Compared to TCP/IP-based communication, where a packet needs to be sent to the kernel before being copied into the memory, RDMA bypasses the kernels of both sender and receiver and transfers data directly from/to application memory. RoCEv2 is a protocol that implements RDMA: an RDMA verbs message is encapsulated in Ethernet/IPv6/UDP packets and carried through regular Ethernet networks. The encapsulation/decapsulation is handled in RDMA NIC hardware.

The collective communication library serves as the software abstraction between training workloads and the NIC, interfacing through the verbs API layer. It translates collectives[1] (e.g., `AllReduce`) into logical topology implementations (e.g., Ring or Tree) and further breaks those down into scheduling point-to-point data transactions between GPUs with verbs. These transactions require GPU-to-RDMA NIC support for optimal performance. The collective library schedules verbs calls over Queue Pairs (QP) created on source and destination NICs. NCCL[25], for example, implements all collective algorithms and point-to-point semantics using RDMA `write` operations. Each GPU-to-GPU pairwise transaction can be over multiple channels, and each NIC-to-NIC pairwise transaction can be over multiple queue pairs.

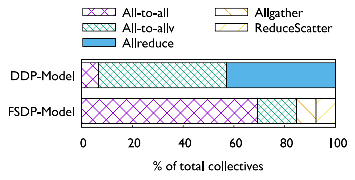
Table 1 provides characteristics of major collectives used for distributed training and specific requirements per collective. First, the collective is determined by the parallelism strategy. For example, Distributed Data Parallel (DDP) [14] uses `AllReduce`, Fully Sharded Data Parallel (FSDP) [41] uses `AllGather` and `ReduceScatter`. Ranking models (e.g., DLRM [21]) makes use of `AlltoAllv` (a vectorized `AlltoAll`) to distribute the embeddings for model parallelism.

Second, collectives can generate a diverse range of network traffic patterns. For example, `AlltoAllv` forms a full-mesh traffic pattern between all endpoints, potentially causing high ephemeral congestion. However, its high number of active flows simplifies routing, reducing persistent congestion risks with hashing schemes.

[1]More collective definitions can be found in [22].



(a) Job size.



(b) Collective type.

Figure 1: Distribution of job characteristics.[2]

Collectives	Avg. # of QPs per GPU	Buffer occupancy per leaf switch (MB)
`AlltoAll(v)`	15	65.6
`AllReduce`	4	13
`AllGather`	4	22.1
`ReduceScatter`	4	19.6

Table 2: Traffic statistics in production (128 GPU)

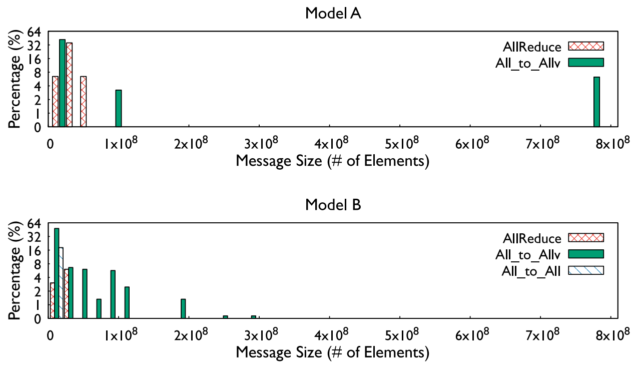


Figure 2: Collective message size distribution.

Third, the choice of logical topology from collective operations impacts network congestion and data exchange between GPUs. For instance, `AllReduce` implemented as Ring vs Tree has unique congestion and hash collision implications. NCCL optimizes the specific choices, based on factors like GPU count and message size. However, this approach has limitations, including potential inaccuracies due to hard-coded profiles, sub-optimal performance with certain message sizes or large scale jobs, and irrelevance of the collective algorithm in some implementations.

2.3 Training workloads

To understand what collective communication is actually observed in production, we leveraged Chakra [33] to gather collective statistics of ~30K randomly selected training jobs during 2023Q4.

Figure 1a shows the job size distribution from our collection. Notably, we do not include the long tail of >128 GPU jobs in the analysis, and so this excludes the LLM jobs. However, we note below that because of multi-dimensional parallelism, analysis of job sizes up to 128 GPUs is still highly relevant to large LLM jobs.

The distribution suggests that the majority of the job sizes are the multiples of 8. This is because we mount 8 GPUs to each host and partial host utilization is not recommended. To demonstrate the diversity of the collectives used by different models, we break down collectives by types. As shown in Figure 1b, `AllReduce` and `AlltoAll(v)` dominate in DDP-based models while `AllGather` and `ReduceScatter` are the primitive collectives in FSDP. Message size is the amount of data elements transferred in collective communication operation. We select two different models and observe that the message size distribution varies a lot (as shown by Figure 2). The data volume being transferred as well as the traffic pattern of different models are also varied. This motivates our routing and transport choices explained in the following sections.

Trends in job sizes: At the time of writing, ranking models largely span around 8 - 256 GPUs. As we look toward the future, larger GPU jobs are becoming increasingly common and the GPU hours they consume is gradually trending up. This is due to the upward trend in model size for ranking models, and in a more pronounced manner for LLMs. For example, a large variant of Llama3 was trained on 16,000 GPUs on our RoCE cluster of 24,000 GPUs.[18]

Trends in number of GPUs per collective: Whether it is ranking jobs or LLMs, the number of GPUs per collective operation is not scaling at the same rate as job size. This is due to the usage of multi-dimensional parallelism in deploying large models. This helps limit the number of GPUs in the largest collective to hundreds of GPUs even when running a job that is tens of thousands of GPUs. For this reason, in the rest of the paper we focus on collective operations that involve a GPU size range of 16 - 128.

2.4 Challenges

We faced several challenges in constructing a RoCE network for distributed AI training needs.

Rapid evolution of training model: Rapid evolution of training models necessitates an increase in network bandwidth towards 400Gbps and higher and a scale-out size to tens of thousands of GPUs. While early ranking job sizes are moderate for now, LLM training is known to occupy tens of thousands accelerators at the same time [8] and ranking jobs are growing as well[39]. Constructing a network of this magnitude while maintaining performant inter-GPU communication presents a significant challenge (Section 3).

Low entropy in traffic pattern: The traffic pattern in distributed training exhibits low entropy in the UDP 5-tuple, as shown in Table 2. This is a result of the collective communication discussed in Section 2.2. Typically, one GPU is solely occupied by one training job, whose logical topology is usually sparse. This leads to a significant challenge for evenly distributing the traffic in routing to achieve optimal performance (Section 4).

Different levels of network congestion: Distributed training produces unique full mesh and hierarchical traffic patterns (like Ring or Tree) that both produce different modes of congestion, as shown in buffer occupancy in Table 2. These traffic patterns can be seen in Figure 3. There is no standard best practice for dealing with congestion for such traffic patterns with prior RDMA deployments (Section 5).

[2]LLM jobs are not included.

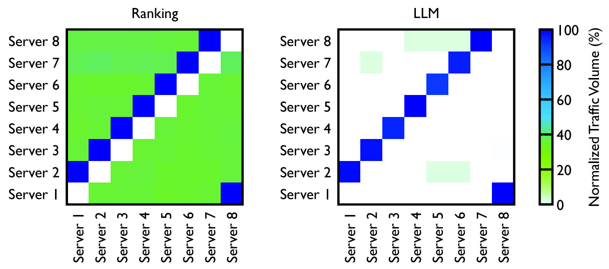


Figure 3: Traffic pattern comparison from experiments

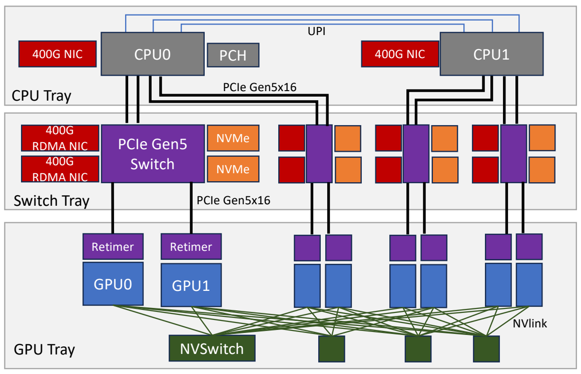


Figure 4: Grand Teton platform

The need for co-tuning: Collective libraries, e.g., NCCL, may deliver sub-optimal, out-of-box performance with RoCE interconnects due to the difference of the developer's environment and production. This necessitates the co-tuning of both collective library and network configurations to achieve optimal performance (Section 6). Low entropy in traffic patterns can result in a few network paths experiencing higher traffic than others resulting in persistent congestion on those paths. Further, even with perfect distribution of traffic, collective traffic patterns like AlltoAll can produce microbursts. While both of these patterns can have negative impact on performance, the exact manifestation and solution approaches to both of these problems are distinct. Hence, we cover solutions to the former problem in (Section 4) and the latter in (Section 5).

3 HARDWARE

In this section, we cover the hardware used for training nodes and the network to set the stage for the software systems that run over them.

3.1 Training Node

The increasing size of training models and datasets has made it infeasible to confine the training process into a single GPU. It also requires increased compute and memory, placing significant demands on the network, thus lending themselves to a specially designed scale up system.

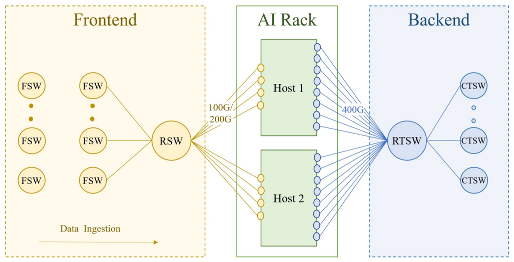


Figure 5: Frontend and Backend networks

The first generation of training node design, ZionEX[20], combines general-purpose CPUs with NVIDIA A100 GPUs. Grand Teton[17], a more recent generation, is based on H100 GPUs. Both ZionEX and Grand Teton utilize a similar system architecture except for GPU NVLink interconnect. Figure 4 depicts Grand Teton's internals. The node is divided into 3 trays - CPU Tray housing 2 CPUs and frontend NICs, Switch Tray housing 4 PCIe Gen5 switches, NVMe storage, as well as 8 RDMA NICs, and GPU Tray housing 8 GPUs. GPUs are fully-connected using NVSwitch[26]. There is a 1:1 mapping between GPUs and NICs. For a training job that uses fewer than 8 GPUs, GPUs communicate with each other within the node without network operation. For larger jobs, RDMA NICs enable GPUDirect technology, so that GPU-to-GPU traffic can bypass host and host memory bottlenecks.

3.2 The Network

The training cluster relies on two independent networks: the Frontend Network (FE) for tasks such as data ingestion, checkpointing, and logging, and the Backend Network (BE) for training, as depicted in Figure 5.

Frontend network: A training rack is connected to both the FE and BE of the data center network. The FE has a hierarchy of network layers[1] - Rack Switches (RSW), Fabric Switches (FSW), and higher - that house the storage warehouse, which provides GPUs with the necessary input data for training workloads. We ensure that there is enough ingress bandwidth on the rack switch to not hinder the training workload.

Backend network: The BE is a specialized fabric that connects all RDMA NICs in a non-blocking architecture, providing high bandwidth, low latency, and lossless transport between any two GPUs in the cluster, regardless of their physical location. This backend fabric utilizes the RoCEv2 protocol, which encapsulates the RDMA service in UDP packets for transport over the network.

3.3 Evolution

Our BE networks have undergone several transformations. Initially, our GPU clusters used a simple star topology with a few AI Racks connected to a central Ethernet switch. running the non-routable RoCEv1 protocol. This setup had clear limitations in GPU scale and switch redundancy. Therefore, we swiftly transitioned to a fabric-based architecture for extended scalability and higher availability.

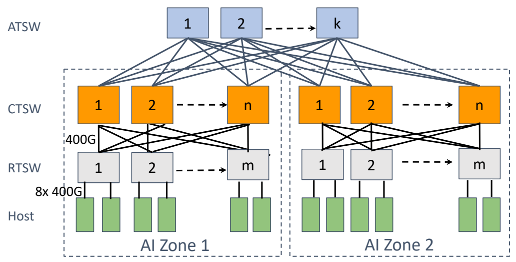


Figure 6: Backend Network Topology

AI Zone We designed a two-stage Clos topology for AI racks, known as an *AI Zone*, as shown in Figure 6. The Rack Training Switch (RTSW), serving as the leaf switch, offers scale-up connectivity for GPUs within the rack using copper-based DAC cables. The spine tier, composed of modular Cluster Training Switches (CTSW), provides scale-out connectivity among all racks in the cluster. The CTSW has deep buffers statically divided over the ports in the chassis. The RTSWs connect to CTSWs via single-mode fiber and 400G pluggable transceivers.

DC-Scale and topology aware scheduling: The AI zones are designed to support a large number of interconnected GPUs in a non-blocking manner. However, emerging AI advancements, such as LLMs, demand a GPU scale larger than what a single AI Zone provides. To accommodate this, we designed an Aggregator Training Switch (ATSW) layer that connects the CTSWs in a data center building, expanding the RoCE domain beyond a single AI Zone. Note, the cross-AI zone connectivity is oversubscribed by design, with network traffic balanced using ECMP. To mitigate the performance bottleneck for cross-AI zone traffic, we enhanced the training job scheduler to find a "minimum cut" when dividing the training nodes into different AI zones, reducing the cross-AI zone traffic and thus collective completion time. The scheduler does this by learning the position of GPU servers in the logical topology to recommend a rank assignment.

3.4 Discussion

The separation between Frontend Network and Backend Network was an early, major design decision in deploying RoCE. This was mostly driven by the fact that we expected the two networks to evolve independently. In addition, separating AI training traffic simplified and accelerated routing and transport design iterations, as the traffic carried by the two networks have very different and often conflicting requirements. Finally, physical separation ensured the ideal network environment for latency-sensitive RoCE operations.

4 ROUTING

The scaling of compute power and network topology discussed above led to the question of how to efficiently balance and route the massive training traffic. Specifically, the AI training workloads had several challenging characteristics: (a) **Low entropy**: Compared to traditional data center workloads, the number and the diversity of flows for AI workloads are much smaller and the flow patterns are usually repetitive and predictable. (b) **Burstiness**: On the time dimension, the flows usually exhibit the "on and off" nature in the time granularity of milliseconds. (c) **Elephant flows**: For each burst, the intensity of each flow could reach up to the line rate of NICs.

We describe below multiple stages of evolution of our routing design to handle these challenges.

4.1 ECMP and Path Pinning

4.1.1 ECMP. We initially considered the widely adopted ECMP, which places flows randomly based on the hashes on the five-tuple: source and destination IP, source and destination UDP port, protocols. However and as expected, ECMP rendered poor performance for the training workload due to the low flow entropy.

We used Max-Mean Ratio (MMR), the flow count of the most loaded link over the average flow count per link, to quantify ECMP imbalance since most flows are the same size within one collective. We observed an average MMR over 1.2 for 1,000 flows on 16 links simulation. In reality, the situation is much worse as shown in Table 2. This imperfect load balancing would result in high probability of collision of large elephant flows, and hence causing severe congestion and slowing down the network throughput and job performance.

4.1.2 Path Pinning. Alternatively, we designed and deployed a path-pinning scheme in the initial years of our deployment. This scheme routed packets to specific paths based on the destination "slice" (the index of the RTSW downlink). This worked well if each rack was fully assigned to the same job and there was no failure in the network. However, this was seldom true. We saw that the rack can be partially allocated to a job, with only one of the two hosts in the rack is using the uplink bandwidth. This fragmented job placement caused uneven traffic distribution and congestion on the uplinks of the particular RTSW and degraded the training performance up to more than 30%. Further, network failures on a uplink or a CTSW caused the affected flows to be unevenly reassigned to other CTSWs by ECMP. Those reassigned flows collided with other existing flows and slowed down the whole training job.

4.1.3 Short-Term and Long-Term Solutions. We mitigated the immediate impact of these flow collisions by upgrading the bandwidth of the RTSW uplinks bandwidth by 2x. Hence we allowed for the RTSW uplink capacity to be 1:2 under-subscribed compared to the RTSW downlink capacity. While this mitigated the immediate performance impact as described later in Section 6.2, this was an expensive solution as it required 2x network capacity. Thus, we recognized this as a short-term mitigation and proceeded to further stages of routing evolution.

4.2 Enhanced ECMP with QP Scaling

We next revisited ECMP with an intent to increase the number of flows for hierarchical collectives through the Queue Pair (QP) Scaling software feature in the collective library.

4.2.1 QP Scaling: The QP Scaling feature helps post messages meant from each NIC-to-NIC flow as multiple flows by posting messages over multiple QPs. We noticed low entropy persisted even with this enabled and a larger number of active Queue Pairs (QPs). Upon debugging, we found that while destination UDP ports remained identical for different QP packets as expected, in some

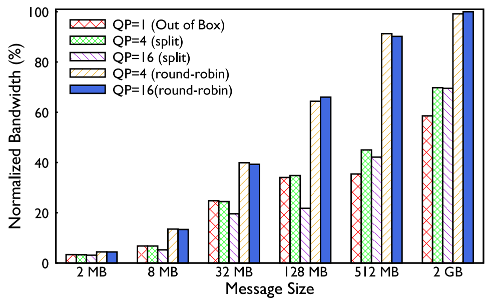
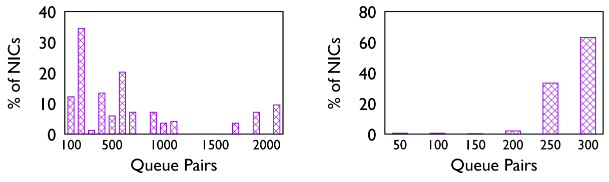


Figure 7: Performance Impact on AllReduce Benchmark: E-ECMP and Queue Pair Scaling



(a) Ranking **(b) LLM**

Figure 8: QPs per NIC in production with QP Scaling

corner cases, so did the source UDP ports, and thus entropy did not increase as we had hoped.

4.2.2 Customized Hashing: To account for this, we configured switches to perform Enhanced ECMP (E-ECMP) to additionally hash on the destination QP field of a RoCE packet using the UDF capability of the switch ASIC. This increased entropy, and compared to baseline ECMP without QP scaling, we observed (Figure 7) that E-ECMP along with QP scaling showed performance improvement of up to 40% for the `AllReduce` collective.

4.2.3 Balancing QP Trade-offs: QP buffers are a scarce resource in RDMA NICs, and QP resources are used differently between our Ranking and LLM workloads. Thus, we use QP=4 for Ranking as they already have relatively high entropy due to full-mesh communication involved (e.g., with `AlltoAll`). The result is seen in (Figure 8). We use a QP Scaling factor of 16 for LLM workloads as they involve hierarchical collectives, e.g., `AllReduce`.

4.2.4 Different QP Scaling Methods: We evaluated two QP scaling strategies. The first involved splitting each message meant to be posted over single QP, instead onto multiple QPs resulting in multiple flows. But it also produced smaller message size on fabric as well as multiple ACKs. The second approach involved posting each message to a different queue, in a round-robin fashion. For the NIC message sizes demonstrated in our production with NCCL, we observed the latter to be performing well. This feature has been important for ECMP scalability by increasing the network flows for hierarchical collectives like `AllReduce`.

4.2.5 Motivation to Look Beyond E-ECMP:. While we improved ECMP performance with QP scaling, the underlying probabilistic nature of hashing was a persistent downside of this routing scheme.

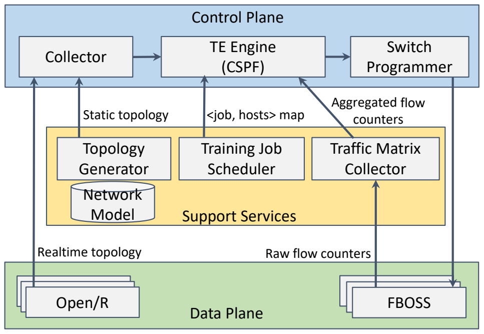


Figure 9: Centralized Traffic Engineering architecture

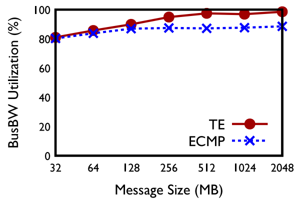
Also, the need to customize the QP scaling factor and methodology based on the workload type, while workable in the short-term, presented long-term operational complexity.

4.3 Centralized Traffic Engineering

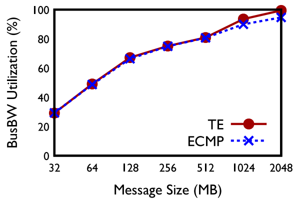
ECMP and path pinning approaches both had limitations from a hardware perspective; thus, we tackled these by developing a centralized traffic engineering (TE) controller, leveraging our previous experience in [5] and [31]. The TE controller dynamically optimizes routing based on real-time workloads and topology input. Figure 9 provides an overview of TE's architecture, demonstrating how it optimizes dynamic path allocation.

4.3.1 Control Plane. The control plane design consists of three components: First, the Collector creates a real-time topology of the end-to-end training cluster. It achieves this by utilizing a topology generator service to bootstrap a static topology from our in-house data warehouse on network model. In addition, it constructs the dynamic topology based on the link states provided by Open/R, our in-house link state routing protocol, which is designed to capture the real-time network state. Second, the TE Engine combines the flow matrix, i.e., the flow bps, from a traffic matrix collector service and job placements from the training job scheduler to derive the traffic matrix, i.e., the byte counters of TE allocated flows. The TE Engine runs a Constrained Shortest Path First (CSPF) algorithm to process the real-time topology and traffic matrix, producing an optimized flow placement periodically, e.g., every 30s. Finally, the Switch Programmer takes the flow placement and translates it into device-specific data plane primitives to enforce the routing decisions.

4.3.2 Data Plane. The data plane operates based on the concept of overriding the default routing policy. Default routes are provided by BGP, ensuring connectivity to all prefixes in the cluster. TE overrides these default routing decisions on RTSWs based on the optimized flow placement, thereby providing the primary routing for the RDMA traffic. TE comprises technologies capable of associating a <source port, destination prefix> tuple with an action of forwarding to a next-hop. Using the source+destination tuple provides finer granularity flow management while using destination prefixes helps keep the scale of these entries manageable in hardware. Specifically, we utilize the Exact Match (EM) table provided by the hardware to override the default routes. BGP-determined



(a) `AlltoAll`



(b) `AllReduce`

Figure 10: Collective benchmark: Normalized Performance Comparison between E-ECMP and TE (128 GPU)

routing decisions serve as a backup for the RDMA traffic when the primary entry is absent due to failures.

4.4 Comparing TE and E-ECMP

Below, we present TE and E-ECMP performance evaluations via a flow-level network simulator, followed by production benchmark results. We then describe the operational complexity of each scheme.

4.4.1 Simulation. The simulation results with production job placement shows that under a non-optimal job scheduling scenario, E-ECMP with 4 QPs per each source-destination pair results in 40% longer than roofline completion time on average. Increasing the QP-scaling to 32 improves performance: worst case is improved from 20% to 52% of roofline. However, the majority of jobs can not achieve the roofline. In comparison, TE with actual demands can achieve 100% utilization when there's enough capacity in network. However, when the link availability is compromised by failures to less than 1:1 subscription ratio, TE can be outperformed by E-ECMP.

4.4.2 Benchmark Evaluation. In a controlled environment, we have observed much more balanced link utilization with TE with a real-world NCCL benchmark [16, 24] compared to E-ECMP on a 16-uplink setup. With E-ECMP, link utilization varies largely: 40-90% of max bandwidth, whereas TE uniformly utilizes 80% of max bandwidth, reducing the worst case scenario. Figure 10 shows that with fixed world size (128 training node), TE outperforms E-ECMP by 5-10% in `AllReduce` and `AlltoAll` collectives.

4.4.3 Operational Experience with TE and Lessons Learned: We rolled out TE in the ranking-based AI Zones with E-ECMP as a backup routing scheme to handle traffic impacted by failures. We observed that TE was similar to the earlier stage path-pinning routing schemes at effective load balancing, performing well (Section 6.2) as modelled in simulations and measured with benchmarks. However, our simulations and deployment showed that TE is also prone to lower performance when multiple links failures happen in the network. We had originally considered these to be rare cases during simulation, but in practice, they happened more frequently than expected. Further, TE had additional software complexity and manageability overhead. While this was manageable in the AI Zone deployments, we chose not to use TE at DC-scale as this additional complexity/overhead would be even greater given the much-increased size of the network. Computationally, there was also increased load to handle another layer of switches (ATSWs) and the accompanying path diversity. E-ECMP, hence presented a better operational trade-off for our DC-scale clusters. Thus, TE was the primary routing scheme for the majority of clusters targeting ranking workloads, while E-ECMP was the primary routing scheme for DC-scale deployments.

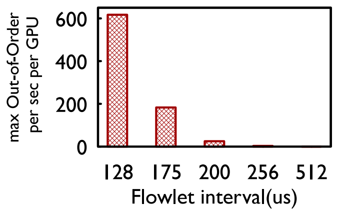


Figure 11: Out-of-order packets under flowlet interval

4.5 Future Direction: Flowlet switching

While TE and E-ECMP comprise our routing approach for different deployments, there are operational trade-offs with each as described earlier.

Flowlet switching[35] is one of the alternative schemes to address issues seen with both of these routing schemes. This technique [35] relies on finding gaps in the flows. When a gap is found in the flow, the flowlet engine makes a decision to reassign the flow to a new ECMP member port based on the load. This is a hardware assisted scheme, supported by the first-hop switch (RTSW), which reacts to changes in the port load at microsecond granularity. In our testing, such a scheme has exhibited superior link load balancing and better performance in case of congestion and multiple-link failure scenarios. Adaptability of this scheme helps it scale with minimal QP scaling for 4 for all workloads. Not needing customized QP scaling on per workload basis mitigates E-ECMP's issues. Hardware support for the scheme helps mitigate the software complexity concerns from TE.

Out-of-order Packets: Flow reassignments can result in out-of-order packets mainly because we resort to path reassignment with active flows. Managing out-of-order packets is an important aspect of this technique and require tuning the flowlet interval. Theoretically this gap needs to be approximately half of the round trip time of the network. Doing so ensures that flow is reassigned to a new path only when previous packets in the same flow have at least reached destination before switching the path resulting in drastic reduction of out-of-order packets. As shown in Figure 11, out-of-order packets go down non-linearly as the interval increases. The out-of-order packets are below 1 packet/second in the 256-512μs flowlet interval range. On the flip side shrinking the flowlet interval leads to more aggressive load balancing. However, there is a diminishing return on performance from more aggressive load balancing. For example in 200G based experimental setup (Figure 11), insignificant difference in performance was found between 256 usec and 512 usec flowlet intervals for A2A collectives. We are able to tune the flowlet interval to achieve optimal load balancing and throughput while incurring negligible to no out-of-order packets.

Load-aware path assignment: For very high interval, flowlet switching tends to be like ECMP where flows are assigned ports statically for the lifetime of the flow. Interestingly, we observed that even in those scenarios, flowlet switching exhibited superior load balancing as compared to ECMP simply because of its picking up initial port based on the port quality (load) rather than hash. So while the path doesn't change for the lifetime of the flow, flowlet mechanism makes better decision to pick the path based on its current port load. In workloads where we do not have long lived flows, we will still benefit from this mechanism.

Status: We found during the initial testing that flowlet switching can achieve similar performance as TE with lower operational cost. Our future roll-out plans with flowlet switching will depend on the signals we get from early deployments. We will cover outcome of such roll-out along with more in-depth trade-offs in future work.

4.6 Discussion

Initially, we had observed low link utilization due to the inefficiency in ECMP as reported in [9] that led to inconsistent performance. To mitigate that, we deployed a temporary over-build of the network. In parallel, we evolved our routing through various stages, including static routing, dynamic routing, and traffic engineering, to achieve consistent and high-performance training. The evolution of routing schemes and resulting positive impact on performance consistency, measured with production data, is presented in Section 6.2. We believe that this will be a continued important research area given the continued evolution of distributed training workloads.

5 TRANSPORT

This section is a deep-dive into our transport design and solutions for challenges relating to network congestion listed in Section 2.4. Section 4 addressed solutions to persistent congestion due to inefficient load balancing. Even with perfect distribution of traffic, collective traffic patterns like `AlltoAll` can cause momentary buffer buildup and micro bursts as shown in Table 2. This phenomenon presents the need for flow control and congestion control for achieving predicable performance by avoiding traffic drops and offering predicable tail latency. We start by describing our initial attempts to leverage DCQCN and then how we pivoted away from DCQCN to manage congestion via receiver-driven admission control at the collective library layer.

5.1 Implementation

We implemented several transport configurations to achieve high bandwidth, low, and predictable latency in our RoCE setup. We collaborated with NIC vendors to support GPUDirect with Linux inbox drivers for better software stack manageability. We used DSCP-based PFC and a single lossless queue across all network switches and NICs to prevent packet drops on Layer 3 network connections. We relied on a go-back-N re-transmit mechanism for rare packet drops due to unhealthy network elements or link flap/down. However, packet drops on acknowledgement packets (ACK or NACK) can cause prolonged Local ACK Timeout (LAT) on the order of milliseconds. Therefore, quickly recognizing and isolating unhealthy network elements and links and carefully tuning LAT duration, are important for predictable training workload performance.

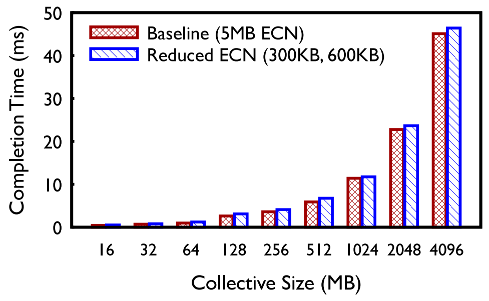


Figure 12: ECN impact on performance: Allreduce completion time(ms) on 32 GPUs comparison with CTSW ECN threshold changes. Baseline uses 5MB as both low and high thresholds. A tighter threshold of 300KB low and 600KB high leads to lower performance.

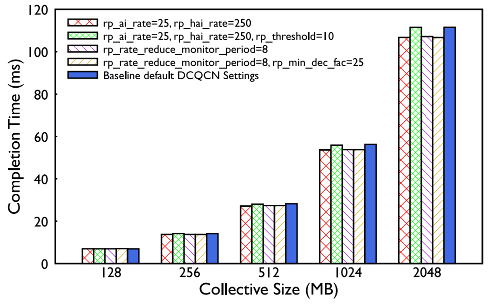


Figure 13: Tuning DCQCN for `AlltoAll` collective

5.2 Congestion Control

While PFC helps avoid congestion drops, hop-by-hop PFC propagation during persistent congestion can lead to head-of-line (HoL) blocking, resulting in flow unfairness and poor performance. We initially deployed DCQCN for our 200G deployments, as recommended by prior RDMA deployments. Our implementation involved a) switches marking in-flight packets with ECN during congestion events, b) receiver NICs generating and sending back Congestion Notification Packets (CNP) to indicate the need to slow down flows upon receipt of marked packets, and c) senders reducing the traffic injection for corresponding flows based on received CNP.

5.2.1 *Limitations of Tuning DCQCN for Collectives.*

We found DCQCN, the *de facto* congestion control scheme used in RoCE, is less performance effective for training collectives from our experience in 200G and 400G RDMA deployment.

Tuning DCQCN is challenging: Initially, we deployed default DCQCN algorithm with a strict ECN threshold configuration to minimize PFC in our 200G RDMA deployment. However, in practice, we found that while tight thresholds helped avoiding PFC, they significantly reduced the throughput of collective performance in some corner cases. While we do not shared data here for extreme regressions seen in corner cases, Figure 12 shows one of the cases that tighter ECN thresholds hurt collective completion time for 32 GPU `AllReduce` benchmark by a smaller margin (~ 5%).

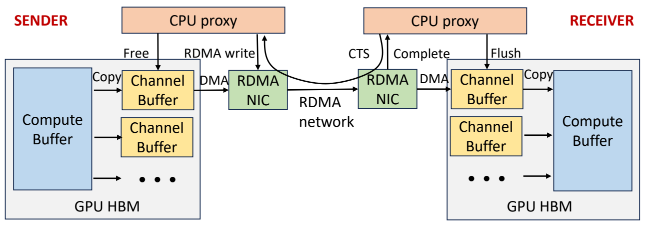


Figure 14: GPU to GPU communication architecture.

We hence reverted to relaxed (5 MB) ECN threshold in CTSW and swept other DCQCN settings in NIC with 128 GPU `AlltoAll`, which exhibited worst case congestion and queue buildup in training. We tried both aggressive and relaxed DCQCN settings. Figure 13 shows examples of how we can get marginally better completion time with a few different relaxed DCQCN setting combinations by a small margin of 3%. But in all of those cases, congestion metrics like PFC became worse by 2-3x, which diminished our intent to adopt DCQCN to avoid HoL blocking and latency inflation.

Thus, we stayed with relaxed ECN marking, allowing for buffer build up in the CTSW, while keeping default DCQCN settings.

Experience with 400G: As we transitioned to 400G deployments, we attempted to further tune DCQCN to adapt to new network speeds and topology. However, with default DCQCN settings and doubled ECN thresholds compared to 200G networks, performance was degraded. Further investigation revealed that DCQCN implementation in firmware has changed, introducing bugs and reduced visibility with problems relating to correct CNP counting.

We proceeded without DCQCN for our 400G deployments. At this time, we have had over a year of experience with just PFC for flow control, without any other transport-level congestion control (but we do use collective-library controls for higher layer congestion management). We have observed stable performance and lack of persistent congestion for training collectives. An indirect positive outcome of this decision is for monitoring purposes, in cases of slow receivers or momentary congestion in the network, we can measure the impact by monitoring NIC pause duration to understand the percentage of time it stopped transmitting or receiving - as opposed to looking at both Congestion Notification Packet metrics and PFC metrics.

5.2.2 *Receiver-Driven Traffic Admission via Collective Library.*

To mitigate the congestion for 400G and beyond, we co-designed the collective library and RoCE transport to enforce receiver-driven traffic admission for better performance. Figure 14 shows that the GPU-to-GPU communication architecture in our production training clusters predominantly uses two-stage copy and receiver-initiated communication via the NCCL collective library. Each GPU's High Bandwidth Memory (HBM) maintains multiple channels for parallel transmission of chunked collective messages. The sender GPU threads first copy data from compute buffer to an available channel buffer. The sender CPU proxy thread can only post an RDMA write request after receiving a clear-to-send (CTS) packet from the receiver, which includes the size and memory information. The receiver's GPU threads then copy the channel buffer contents to the destination compute buffer. Finally, CPU proxy threads on both sides recycle the channel buffer, and the receiver CPU proxy sends another CTS packet once the channel buffer is ready.

PFC	Perftest	`Gather`
Downstream RTSW→CTSW	350K/sec	0
CTSW→Upstream RTSW	183K/sec	0
Upstream RTSW→Sender NIC	10% duration	0

Table 3: Receiver-driven traffic admission under incast

We effectively leverage this mechanism as a receiver-driven traffic admission to limit the amount of in-flight traffic on the network, especially when congestion starts to build up. However, configuring the right setting can be challenging as: 1) the number of channels is limited due to the resource contention on GPU threads with concurrent compute operations; 2) setting the channel buffer size requires a more careful balance between congestion spreading and bandwidth under-utilization than Infiniband due to RoCE's more coarse-grained flow control and possible end-host slowness.

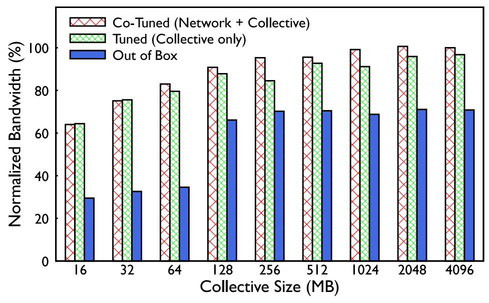
Thus, we took two steps to improve the performance. First, we experimentally determined the right parameter settings for the number of channels and channel buffer size across various training job sizes and collective types. Second, we implemented high priority queuing at switches for CTS packets to expedite the notifications and mitigate potential bandwidth starvation.

To demonstrate the effectiveness of this mechanism, we conducted 16:1 incast experiments on our 400G network with only PFC enabled, as shown in Table 3. We compared a streaming traffic generator Perftest [27] and a recurring NCCL `Gather` collective test. Although `Gather` collective is rarely used in our production training, it generates the most severe incast among other collectives, stressing our mechanism. Both tests ran for five minutes, producing 16:1 incast with the same traffic volume. We observed HoL blocking in perftest. The deep buffer in the CTSW absorbed a significant amount of congestion, but not enough to prevent back pressure from reaching the sending NICs. However, no PFC back pressure was observed in `Gather` collective case. To mimic the slow receiver scenario observed in production, we further restricted the bandwidth of the receiver in `Gather` collective case to only 25% of its capacity. We observed that our mechanism adapts to the throughput changes. While we observed Receiving NIC sending 75% of TX pause duration to RTSW, the RTSW absorbed all of this congestion and did not cause back pressure to the CTSWs.

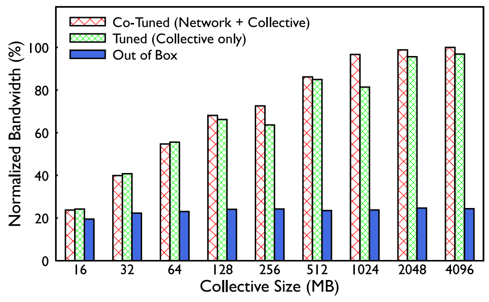
This experiment highlights how receiver-driven traffic admission with the collective library results in controlling network congestion.

5.2.3 *Absorbing In-Network Congestion.*

Our AI zone is built on a 2-stage Clos architecture (Section 3.2). While we used switches with shared and shallow buffer architecture for our RTSWs, we used CTSWs with deep buffers. We leveraged these large buffers by statically carving them per port on the ingress. This large per-port buffer helped absorb any ephemeral congestion and ensured a port facing back pressure to reduce HoL blocking across ports. Given the high radix of our spine switches, we considered this non-blocking architecture an additional safety layer towards minimizing congestion.



(a) AlltoAll



(b) AllReduce

Figure 15: Network/Collective Co-Tuning performance

5.3 Discussion

Congestion control have been a focal point of research in RDMA networks [15, 43]. DCQCN has been the "gold standard" for storage-focused networks. However, our experience with distributed AI training workloads provides a different perspective on tailoring the congestion control algorithms. Despite turning off DCQCN and multiple instances of RTSW sending PFC to a deep-buffer CTSW, we have not encountered a scenario over the last 4 years where production AI training traffic causes the CTSW to send PFCs to RTSWs persistently. Specifically, it's worth evaluating the feasibility of operating without transport-level congestion control at all. Our current solution depends on careful coordination between the collective communication library and the network, and may depend on the relative throughput between GPU and network, which may not be applicable to all scenarios. We encourage the research community to put more focus on this topic.

6 EXPERIENCES

6.1 Co-tuning Network and Collective

Our journey towards achieving close to roof-line communication performance for AI Training workloads involved tuning how collective library, transport and network layers interact together. We have been able to improve the out-of-the-box RoCE performance by more than 2x in many cases and even higher as shown in Figure 15.

We observed NCCL may deliver sub-optimal out-of-the-box performance due to the difference of the developer's environment vs. our production environment. Some of the assumptions from the developer environment include: very low RTT latency (<10μs), an adaptive routing mechanism, a non-blocking topology without over-subscription. Those assumptions are shown in sub-optimal choices across the architecture, including a two-stage copy in posting smaller messages; a receiver-initiated architecture that relies on control messages (CTS) in the critical path; and limited logical topology choices that accumulate latency at large gangs, e.g., Ring for `AllGather`. In terms of adaptability, NCCL used in our production environment learns and adapts to server topology, but it has no default awareness of network topology.

Higher unloaded RTT: In our production, we observed higher unloaded RTT (22μs) due to CTSW switches using a Virtual Output Queuing (VOQ) architecture and hence requiring credit information exchange from egress to ingress queue. This high latency necessitated tuning the message size posted to be larger as well as ensuring more outstanding data on the network. This involved tuning channel buffer size and message size that is posted to the network to be just large enough to achieve optimal performance. For latency-sensitive collectives like `AllGather`, `ReduceScatter`, increasing the message size as well as interim buffer size improved performance.

Rendezvous message performance impact: The Receiver-driven communication architecture relies on rendezvous messages such as Clear to Send (CTS) and Acknowledgements (ACK). In production, we observed that congestion buildup has delayed these messages on the return path. We instrumented NCCL to measure the average delay the sender is waiting for such packets. We reduced this delay from P90 of 43 us to 4 us. Changing the QoS priority of CTS messages was implemented as a collective library change. For ACK packets, we used RTSW ASIC features to modify the DSCP marking to land them in a different priority.

Small messages: NCCL [25] accounts for optimal performance for small collective sizes by following 2 strategies: 1) Different logical topologies to implement the same collective, e.g., Tree vs Ring, 2) Different low-latency or high-bandwidth protocols, e.g., Simple, LL128, LL (low latency), to deal with memory barriers when crossing GPU memory to the PCIe / RDMA boundary or vice-versa. Each topology or protocol may be optimal for smaller or larger collective size. However, the tuning of when to use a particular logical topology or protocol is calculated by a static tuning model that assumes low fabric latency (both loaded and unloaded). This results in unfavorable trade-offs and poor performance. We explain the latency trade-offs of logical topology and protocols below.

Our strategy to achieve optimal performance at smaller or medium collective size is to tweak the tuning algorithm to choose recursive logical topology (Tree) or one that posts larger messages on the network (Rail-based Alltoall) as well as protocols (LL128) at larger collective sizes than the default NCCL choice. Posting larger messages helps reduce the number of CTS and completion messages.

CTSW latency: While the above tuning and changes ensure the collective performance is optimal for our baseline RTT, we also reduced the RTT from CTSWs that leverage a VOQ architecture. This involved tuning a more aggressive credit allocation from egress to ingress port, both in terms of initial allocation as well as the ramp curve of credits. For hierarchical collectives specifically that are latency-sensitive, we noticed up to 15% performance improvement for small sizes.

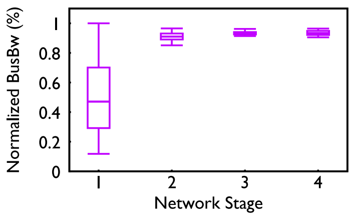


Figure 16: Performance in various Network Stages

Other enhancements included tuning NIC PCIe credits and relaxed ordering; network-topology-aware rank assignment; and eliminating causes for zero-byte messages sent by the NIC that are not supported by Ethernet switches but are supported by InfiniBand. All of these changes helped ensure our RoCE fabric provides optimal AI Training performance, in spite of distinct artifacts such as high fabric RTT.

6.2 Impact of Routing and Topology

We examine the evolution of one of our ZionEX AI Zones (200G based) through stages over time, from a routing and topology perspective. Figure 16 depicts the performance of each stage (time period) over tens of thousands of jobs run over years, quantified by normalized `AllReduce` collective bandwidth[24].

The Stage 1 Backend Network has a topology with 1:1 subscription ratio from RTSW-to-CTSW, while Stage 2 has a topology with 1:2 under-subscription. Both Stage 1 and Stage 2 use static routing with traffic collisions. We observe a significant performance increase with additional capacity when migrating from Stage 1 to Stage 2. The lower and inconsistent performance in Stage 1 is attributed to the static routing traffic collisions as described earlier. While the performance impact was resolved with Stage 2's under-subscription, the solution came with investing 2x network infrastructure.

Stage 3 shows performance when we migrated our network setup to be controlled by the traffic engineering while keeping under-subscription to 1:2. You can observe similar performance with Stage 2 but with a tighter band. In Stage 4, we further reduced under-subscription ratio to 1: 1.125 to reclaim CTSW hardware without any negative impact to performance. We didn't reduce the under-subscription all the way to 1:1 in order to allow for buffer for up to two link failures.

6.3 Observability Tools

As we operate these RoCE backend networks, we require observability over all network components (fabric, routing, transport) and the collectives in order to quickly troubleshoot failed or slow workloads.

6.3.1 Job-Facing Network Errors. RDMA is very sensitive to network issues and hence it affects GPU training efficiency. In order to quickly examine the RDMA network condition behind a training workflow, we built telemetry systems to automatically collect RDMA hardware counters across the network switch, NIC, PCIe switches and GPUs. We identified three important counters to identify network issues: (1) Out-of-Sequence (OOS): The number of packets out of sequence as perceived by the NIC. This helped us identify and root-cause many instances of packet drops due to unhealthy network switches as well as NIC hardware bugs. There were instances of packet drops not reported anywhere else, but this transport metric showed the existence of the problem. (2) Link Flap Counters: This can indicate both hardware and software flaps reported by the NIC. (3) Local Ack Timeouts (LAT): The number of times a Queue Pair's ACK timer expired for QPs at the sender side. Expiration can impact performance regressions and on occasion job failures. With RoCET, these counters are directly reported to users when their jobs fail and serve as a strong indication that some components within the network may be misbehaving.

6.3.2 Operator-Facing Network Errors. We have safeguard mechanisms built into our network where we not only monitor and detect anomalies but also perform automated mitigation to fix many of these issues.

PFC Watchdog. This is enabled on both RTSW and CTSW devices to catch any long-duration PFC pause (>200ms) which could be because of deadlocks or a bad NIC which continues to send PFC frames.

Buffer thresholds and Congestion Drops. We have monitoring around buffer utilization on RTSW devices. Alarms are raised if buffer utilization goes beyond 80% which points to either persistent congestion or a bug. In practice, we have not breached this threshold. We also monitor packet drops due to congestion. Since we have a lossless network, these drops are uncommon and observed mostly due to misconfiguration.

Reachability. We have several in-house tools periodically checking for the health and connectivity of the fleet by sending pings to various nodes to detect either aliveness or abnormal loss and delay in the network.

6.4 Troubleshooting Examples

In this section, we share a few incidents that demonstrate the complexity of operating such RoCE networks.

6.4.1 Performance Baseline. We observed a performance regression detected during one of our cluster turn-ups. We found no congestion metric that we were monitoring that was out of the baseline. Further investigation revealed that the only difference was the software image we had in our CTSWs. After downgrading the CTSW images to be the same as the one used in the baseline, the numbers aligned again. We engaged the vendor to understand the delta between the two images, and it was highlighted that there was a change in the internal packet scheduling which led to a higher port-to-port latency for that device. This incident showcased the need for constant latency monitoring in our backend networks, both loaded (with congestion) and unloaded (when the network is not saturated).

Implication This issue showcased the need to measure and monitor loaded and unloaded latency to ensure we catch regressions due to various factors. We now use a mix of synthetic traffic generation and library instrumentation to capture this and establish a baseline.

6.4.2 *Dynamic Nature of AI Training Job.*

We observed several CTSWs reporting RoCE traffic drops in a training cluster which we migrated to support the integration of the TE controller. Migration consisted of: reconfiguration of the CTSWs, changes in the QoS settings, routing changes in the RTSWs, and PCIe credit upgrade for NIC. Only a few switches reported traffic drops post migration, but the aggregate drop volume was enough to disrupt some jobs.

Upon investigation, we traced the problem to an intricate interaction of multiple parallel events. The primary cause of the drop was a bug from an outdated assumption about the SRAM buffer size in the CTSWs, which led to tail-drop of traffic when more than half of the buffer was populated. The overflowing of the SRAM buffer only occurred under a certain combination of aggravating factors: (1) a firmware upgrade that enabled more aggressive RoCE packet transmission, (2) a training job with bursty traffic pattern, and (3) the introduction of H100 GPUs with greater computing resources and thus higher I/O requirements.

Implication: This incident showcased the importance of thorough validation and performance regression checks before deploying changes. It is important to note that in the problem space of RoCE for AI training jobs, we must contend with the dynamic nature of training jobs. A significant portion of training jobs is experimental, so predicting traffic patterns and therefore conducting thorough end-to-end testing is challenging. This incident emphasized the importance of having effective detection and alerting systems to quickly identify and respond to issues.

7 RELATED WORKS

Deployment experience of RDMA networks: Traditionally, RDMA networks had been deployed mostly for storage applications for its reduced CPU overhead and network latency [3, 6, 9, 19]. There are many efforts to address several issues in deploying RDMA at scale, such as congestion control [15, 43], reliability [9, 10, 28, 38], and performance isolation [40]. To the best of our knowledge, we are the first to present challenges and experience in deployment RDMA networks for large-scale AI training, including the reassessment of topology, routing, and operation aspects.

Improving networks for AI: There are many works proposed to improve the networks for AI workloads. For instance, existing works propose new network topologies [36, 37] to align with traffic demand and characteristics in distributed training. BytePS [12] leverages spare CPU and bandwidth resources to accelerate distributed training, while many efforts exploit network bandwidth to accelerate collective communication [4, 30, 32]. [29] proposes a novel scheme to combine congestion control and scheduling to accelerate training. In this paper, we showed how we focused on improving RDMA for AI training by presenting deployment experience and best practices in operating AI network at large-scale.

Networks and AI co-design: Prior works illustrated the benefits of co-designing network communication and AI training. For instance, TPU clusters are optimized with 3D parallelism for training large models [13], while Neo is paired with ZionEX with 4D parallelsim for optimizing communications for large-scale DLRM training [20]. Some works propose to automate model parallelism [34, 42]. In this paper, we have shared similar principles to co-design RDMA networks with the understanding of distributed training workloads from the speed, scale, and collective communication.

8 CONCLUSION

The design and operation of large-scale RoCE networks for distributed AI training workloads have evolved to meet the increasing demands of computational density and scale. By segregating Frontend and Backend networks, employing various routing schemes, and optimizing collective traffic patterns, we have been able to build a performant and reliable network infrastructure. These designs and insights underline the importance of deeply understanding the training workload and translating these implications into network component design, ultimately contributing to the advancement of distributed AI training infrastructure.

ACKNOWLEDGEMENT

Many current and former people in the Network Infrastructure team at Meta have contributed to productionizing RoCE networks for AI training over the years. In particular, we would like to acknowledge Srinivas Sridharan, Petr Lapukhov, Jose Leitao, and Brandon Taylor. This work is a close collaboration with our partners in Meta's AI Production Engineering, AI and Systems Co-design, and AI Hardware Systems teams. We are also indebted to Omar Baldonado, our shepherd Kun Tan, and the anonymous SIGCOMM reviewers for their comments and suggestions on earlier drafts.

REFERENCES

[1] Anubhavnidhi Abhashkumar, Kausik Subramanian, Alexey Andreyev, Hyojeong Kim, Nanda Kishore Salem, Jingyi Yang, Petr Lapukhov, Aditya Akella, and Hongyi Zeng. 2021. Running BGP in Data Centers at Scale. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Virtual, 65–81. https://www.usenix.org/conference/nsdi21/presentation/abhashkumar

[2] Amazon. 2024. Elastic Fabric Adapter. https://aws.amazon.com/hpc/efa/.

[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. 2023. Empowering Azure Storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 49–67. https://www.usenix.org/conference/nsdi23/presentation/bai

[4] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. 2023. MSCCLang: Microsoft Collective Communication Language. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 502–514. https://doi.org/10.1145/3575693.3575724

[5] Marek Denis, Yuanjun Yao, Ashley Hatch, Qin Zhang, Chiun Lin Lim, Shuqiang Zhang, Kyle Sugrue, Henry Kwok, Mikel Jimenez Fernandez, Petr Lapukhov, Sandeep Hebbani, Gaya Nagarajan, Omar Baldonado, Lixin Gao, and Ying Zhang. 2023. EBB: Reliable and Evolvable Express Backbone Network in Meta. In *Proceedings of the ACM SIGCOMM 2023 Conference* (, New York, NY, USA,) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 346–359. https://doi.org/10.1145/3603269.3604860

[6] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian,

Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. 2021. When Cloud Storage Meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Virtual, 519–533. https://www.usenix.org/conference/nsdi21/presentation/gao

[7] Google. 2024. Fastsocket. https://cloud.google.com/kubernetes-engine/docs/how-to/nccl-fast-socket.

[8] Google. 2024. Google Cloud demonstrates the world's largest distributed training job for large language models across 50000+ TPU v5e chips. https://cloud.google.com/blog/products/compute/the-worlds-largest-distributed-llm-training-job-on-tpu-v5e.

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

[10] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. 2019. Tagger: Practical PFC Deadlock Prevention in Data Center Networks. *IEEE/ACM Transactions on Networking* 27, 2 (2019), 889–902. https://doi.org/10.1109/TNET.2019.2902875

[11] IBTA. 2024. RMDA over Converged Ethernet. https://www.roceinitiative.org/.

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

[13] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 82, 14 pages. https://doi.org/10.1145/3579371.3589350

[14] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch distributed: experiences on accelerating data parallel training. *Proc. VLDB Endow.* 13, 12 (aug 2020), 3005–3018. https://doi.org/10.14778/3415478.3415530

[15] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 44–58. https://doi.org/10.1145/3341302.3342085

[16] Mingyu Liang, Wenyin Fu, Louis Feng, Zhongyi Lin, Pavani Panakanti, Shengbao Zheng, Srinivas Sridharan, and Christina Delimitrou. 2023. Mystique: Enabling Accurate and Scalable Generation of Production AI Benchmarks. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 37, 13 pages. https://doi.org/10.1145/3579371.3589072

[17] Meta. 2024. Grand Teton. https://www.opencompute.org/documents/grand-teton-intel-based-cpu-tray-specification-v1-0-pdf.

[18] Meta. 2024. Introducing Meta Llama 3: The most capable openly available LLM to date. https://ai.meta.com/blog/meta-llama-3/.

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

[20] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie (Amy) Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, KR Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Ajit Mathews, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. 2022. Software-Hardware Co-Design for Fast and Scalable Training of Deep Learning Recommendation Models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 993–1011. https://doi.org/10.1145/3470496.3533727

[21] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. arXiv:1906.00091 [cs.IR]

[22] NVIDIA. 2024. Collective Operations. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html.

[23] NVIDIA. 2024. InfiniBand Networking Solutions. https://www.nvidia.com/en-us/networking/products/infiniband/.

[24] NVIDIA. 2024. NCCL Tests. https://github.com/NVIDIA/nccl-tests.

[25] NVIDIA. 2024. NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl.

[26] NVIDIA. 2024. NVLink. https://www.nvidia.com/en-us/data-center/nvlink/.

[27] PERFTEST PACKAGE. 2023. perftest. https://enterprise-support.nvidia.com/s/article/perftest-package.

[28] Kun Qian, Wenxue Cheng, Tong Zhang, and Fengyuan Ren. 2019. Gentle flow control: avoiding deadlock in lossless networks. In *Proceedings of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, New York, NY, USA, 75–89. https://doi.org/10.1145/3341302.3342065

[29] Sudarsanan Rajasekaran, Manya Ghobadi, Gautam Kumar, and Aditya Akella. 2022. Congestion control in machine learning clusters. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks* (Austin, Texas) *(HotNets '22)*. Association for Computing Machinery, New York, NY, USA, 235–242. https://doi.org/10.1145/3563766.3564115

[30] Joshua Romero, Junqi Yin, Nouamane Laanait, Bing Xie, M. Todd Young, Sean Treichler, Vitalii Starchenko, Albina Borisevich, Alex Sergeev, and Michael Matheson. 2022. Accelerating Collective Communication in Data Parallel Training across Deep Learning Frameworks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1027–1040. https://www.usenix.org/conference/nsdi22/presentation/romero

[31] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. 2017. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 418–431. https://doi.org/10.1145/3098822.3098853

[32] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 593–612. https://www.usenix.org/conference/nsdi23/presentation/shah

[33] Srinivas Sridharan, Taekyung Heo, Louis Feng, Zhaodong Wang, Matt Bergeron, Wenyin Fu, Shengbao Zheng, Brian Coutinho, Saeed Rashidi, Changhai Man, and Tushar Krishna. 2023. Chakra: Advancing Performance Benchmarking and Co-design using Standardized Execution Traces. arXiv:2305.14516 [cs.LG]

[34] Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, Xi Luo, Dheevatsa Mudigere, Jongsoo Park, Misha Smelyanskiy, and Alex Aiken. 2022. Unity: Accelerating DNN Training Through Joint Optimization of Algebraic Transformations and Parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 267–284. https://www.usenix.org/conference/osdi22/presentation/unger

[35] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 407–420. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini

[36] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. 2023. How to Build Low-cost Networks for Large Language Models (without Sacrificing Performance)? arXiv:2307.12169 [cs.NI]

[37] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 739–767. https://www.usenix.org/conference/nsdi23/presentation/wang-weiyang

[38] Xinyu Crystal Wu and T. S. Eugene Ng. 2022. Detecting and Resolving PFC Deadlocks with ITSY Entirely in the Data Plane. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. IEEE Press, London, United Kingdom, 1928–1937. https://doi.org/10.1109/INFOCOM48880.2022.9796798

[39] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, Yinghai Lu, and Yu Shi. 2024. Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations. arXiv:2402.17152 [cs.LG]

[40] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. 2022. Justitia: Software Multi-Tenancy in Hardware Kernel-Bypass Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1307–1326. https://www.usenix.org/conference/nsdi22/presentation/zhang-yiwen

[41] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *Proc. VLDB Endow.* 16, 12 (aug 2023), 3848–3860. https://doi.org/10.14778/3611540.3611569

[42] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 559–578. https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin

[43] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 523–536.