HyLink: Harnessing PCIe and Dedicated Interconnects for Efficient Collective Communication

Yuezheng Liu
Beihang University
Beijing, China
liuyuezheng@buaa.edu.cn

Menghao Zhang
Beihang University
Beijing, China
zhangmenghao@buaa.edu.cn

Xuebin Song
Beihang University
Beijing, China
soybean@buaa.edu.cn

Juner Shen
Beihang University
Beijing, China
shenje@buaa.edu.cn

Chunming Hu
Beihang University
Beijing, China
hucm@buaa.edu.cn

Xudong Liu
Beihang University
Beijing, China
liuxd@buaa.edu.cn

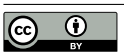
Abstract

Distributed LLM training requires GPUs to cooperate through collective communication primitives. While systems with proprietary high-speed interconnects have been extensively studied, the PCIe bandwidth in GPU clusters remains severely underutilized. Existing collective communication libraries treat the hierarchical PCIe tree as a flat GPU-to-GPU network and relegate the CPU to a passive data relay. Moreover, in systems equipped with dedicated interconnects, the PCIe path is largely overlooked. This paper introduces HyLink, a two-layer collective communication framework, including a PCIe CCL module and a Multipath Orchestrator module. PCIe CCL unlocks underutilized PCIe bandwidth by natively modeling the PCIe hierarchy and elevating the CPU to an active routing and computation node. It provides a CPU-driven Domain-Specific Language (DSL) for multi-stage data flows and an auto-pipelined transfer engine for host-side routing and SIMD-accelerated reduction. Multipath Orchestrator further enables hybrid-link acceleration by splitting each collective operation across the dedicated link and the PCIe bus. It employs a feedback-driven adaptive balancer to dynamically adjust the load ratio to accommodate runtime PCIe interference. Our experiments show that PCIe CCL achieves up to 2.28× the bandwidth of NCCL on PCIe-only NVIDIA GPUs. HyLink improves bandwidth by up to 40% over HCCL on Ascend NPUs, and Multipath Orchestrator achieves 66% higher throughput than static splitting under dynamic PCIe interference.

CCS Concepts

• **Computing methodologies** → **Distributed computing methodologies**; **Machine learning**; • **Networks** → **Data center networks**.

ACM Reference Format:
Yuezheng Liu, Menghao Zhang, Xuebin Song, Juner Shen, Chunming Hu, and Xudong Liu. 2026. HyLink: Harnessing PCIe and Dedicated Interconnects for Efficient Collective Communication. In *The 10th Asia-Pacific Workshop on Networking (APNet 2026), August 06–07, 2026, Singapore, Singapore.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3820441.3820473



This work is licensed under a Creative Commons Attribution 4.0 International License.
APNet 2026, Singapore, Singapore
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2664-4/26/08
https://doi.org/10.1145/3820441.3820473

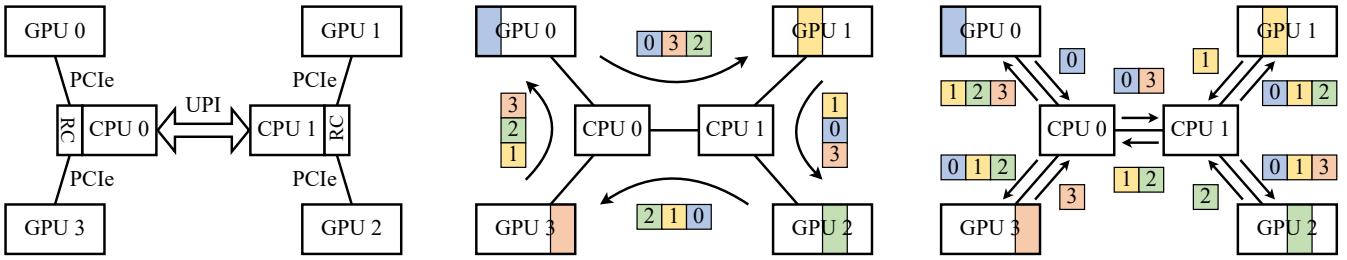
1 Introduction

Over the past several years, the parameter scale of deep learning models, particularly Large Language Models (LLMs), has grown significantly [36]. To address the challenges of training at such scale, various parallel training techniques [22], including Data Parallelism (DP), Pipeline Parallelism (PP), and Tensor Parallelism (TP), have been widely adopted, requiring frequent synchronization of large volume of data via collective communication. Consequently, collective communication has become a primary bottleneck. For example, in Meta's production environment, network overhead accounts for up to 60% of total training time [32].

To alleviate this bottleneck, hardware vendors provide optimized collective communication libraries (CCLs) [1, 10, 23], and numerous studies have optimized routing schemes [3, 4, 9, 21, 28, 34] or hidden latency through communication-computation overlap [6, 7, 31]. However, these efforts predominantly target dedicated high-speed interconnects (e.g., NVLink, xGMI [17], HCCS [11]), overlooking the foundational role of PCIe, which remains critical in two broad scenarios as follows. First, the vast majority of industrial and academic environments still rely on legacy or cost-effective deployments lacking high-speed dedicated links [33]. Second, even in systems with dedicated interconnects, the PCIe tree still carries substantial idle bandwidth that could be leveraged for hybrid-link acceleration.

Nevertheless, current CCLs fail to fully exploit PCIe bandwidth due to two architectural limitations. First, existing CCLs mentioned above fail to fully exploit PCIe bandwidth due to a logical-to-physical topology mismatch (Figure 1): they treat the hierarchical PCIe tree as a flat peer-to-peer network, relegating the CPU to a passive data relay. Second, in hybrid interconnect scenarios, current CCLs largely ignore the PCIe path. The prior attempt to harness both paths [30] must disable GPU peer access to force PCIe routing, incurring non-negligible switching overhead. It also relies on a static splitting ratio that cannot adapt to frequent PCIe bus interference, inevitably leading to severe stragglers.

To address these challenges, we present HyLink, a two-layer co-design framework that unlocks underutilized PCIe bandwidth for collective communication, which jointly harnesses dedicated interconnects and the PCIe bus for hybrid-link acceleration. The first layer, PCIe CCL, breaks the flat P2P abstraction by natively modeling the hierarchical PCIe topology and elevating the CPU to an active routing and computation node. PCIe CCL introduces a CPU-driven Domain-Specific Language (DSL) for multi-stage data



(a) PCIe physical topology. (b) Existing CCLs: CPU as passive relay. (c) PCIe CCL: CPU as active router.

Figure 1: Logical-to-physical topology mismatch in PCIe collective communication. (b) and (c) show a 4-GPU AllGather where each GPU holds one chunk. Under the bandwidth-optimal ring algorithm (b), each GPU uploads its chunk 3 times and 6 chunks traverse UPI. With CPU-side routing (c), each GPU uploads only once and only 4 chunks cross UPI, eliminating 2/3 of redundant PCIe uploads and reducing UPI traffic by 1/3.

Table 1: Per-GPU bi-directional bandwidth of the dedicated link (B_d) and PCIe (B_p) across modern accelerators. B_p/B_d is the theoretical upper bound on the additive bandwidth gain of hybrid-link acceleration over dedicated-only execution.

Accelerator	B_d	B_p	B_p/B_d
NVIDIA V100 32G SXM	300 GB/s	32 GB/s	10.7%
NVIDIA A100 80G SXM	600 GB/s	64 GB/s	10.7%
NVIDIA A800 40G SXM	400 GB/s	64 GB/s	16.0%
NVIDIA H100 SXM	900 GB/s	128 GB/s	14.2%
NVIDIA H800 SXM	400 GB/s	128 GB/s	32.0%
AMD Instinct MI300X	896 GB/s	128 GB/s	14.3%
Huawei Ascend 910B	392 GB/s	64 GB/s	16.3%

flows and an auto-pipelined transfer engine for host-side routing and SIMD-accelerated reduction. The second layer, Multipath Orchestrator, transparently intercepts existing CCL calls and splits each collective into two parallel sub-tasks, invoking PCIe CCL for the PCIe path alongside vendor CCLs for the dedicated link, with a feedback-driven adaptive balancer that dynamically adjusts the load ratio at runtime.

We evaluate HyLink on both NVIDIA and Ascend [18] platforms across PCIe-only, hybrid interconnect, and interference scenarios. In PCIe-only scenarios on NVIDIA GPUs, PCIe CCL achieves up to 2.28× the bandwidth of NCCL. In hybrid interconnect scenarios on Ascend NPUs [18], HyLink improves bandwidth by up to 40% over the vendor-provided HCCL using dedicated links alone. Under dynamic PCIe interference, Multipath Orchestrator achieves 66% higher throughput than static splitting strategy, demonstrating robust resilience to runtime PCIe fluctuations.

2 Background and Motivation

2.1 PCIe & Hybrid Architectures

Unlike dedicated interconnects that provide full-mesh or ring topologies, the PCIe fabric in multi-GPU nodes forms a hierarchical topology centered around the CPU [29]. While some high-end vendor-built systems may attach GPUs to non-blocking PCIe switches, a large number of cost-efficient systems connect GPUs directly to the CPU's Root Complex (RC) [16]. In multi-socket configurations, CPUs across different sockets are further interconnected via inter-processor links such as Intel UPI [13]. In such PCIe-dependent systems, the Root Complexes and cross-socket boundaries become unavoidable bottlenecks, introducing complex congestion patterns.

In deployments without dedicated high-speed links, this CPU-centric PCIe hierarchy serves as the sole communication backbone. Even in systems equipped with dedicated interconnects, the PCIe tree carries substantial bandwidth that can supplement collective communication. As Table 1 shows, PCIe consistently delivers a substantial fraction of the dedicated-link bandwidth across vendors and hardware generations, motivating a hybrid-link strategy that simultaneously harnesses both fabrics.

2.2 Limitations of Existing CCLs

PCIe Modeling Limitation. For the PCIe hierarchy itself, existing GPU-centric CCLs (such as NCCL [23]) fail to utilize its bandwidth effectively due to fundamental limitations in their logical abstractions. Traditional collective communication algorithms (such as Ring [5]) and modern chunk-oriented scheduling frameworks and algorithms (such as MSCCL [9] and SCCL [3]) typically abstract the topology as a set of direct GPU-to-GPU edges without modeling the CPU or the PCIe hierarchy as explicit topology levels. This assumption presents a severe limitation when applied to the hierarchical physical reality of PCIe structures. Although TCCL [16] recognizes these PCIe inefficiencies and profiles NUMA-aware costs, it only optimizes the physical ring ordering atop NCCL rather than modeling the PCIe hierarchy as explicit topology levels.

Under this flat abstraction, existing CCLs communicate over PCIe via either PCIe P2P transfers or host-relayed shared memory copies, both of which suffer from inherent inefficiencies. PCIe P2P transfers still traverse the CPU Root Complex in typical server configurations, consuming the same link bandwidth as a host-relayed copy. Moreover, their performance also degrades unpredictably across NUMA boundaries [16]. In the shared memory path, the host is treated merely as a passive data relay, with host memory allocated strictly as intermediate bounce buffers that force each data chunk through unoptimized two-stage copies [23]. In both cases, the computational and routing potential of the host is wasted. Consequently, current CCLs inherently generate massive redundant PCIe or UPI traffic and fail to fully saturate the available hardware bandwidth (Figure 1).

Hybrid Interconnect Utilization. To unlock the additional bandwidth offered by the PCIe tree in hybrid interconnect systems,

harnessing both dedicated links and PCIe simultaneously is desirable but faces two key challenges. First, the driver stack typically routes GPU P2P transfers through the dedicated link when one is present, precluding explicit use of the PCIe path. Vendor drivers may allow forcing PCIe routing by disabling GPU peer access at runtime, but this introduces switching overhead that grows with GPU count [30]. Second, the PCIe bus operates as a shared resource and is highly vulnerable to bursts of background traffic. Operations such as model checkpointing, asynchronous data loading, and monitoring log I/O, as well as resource contention from co-located processes, can unpredictably consume PCIe bandwidth, leading to severe performance fluctuations.

Prior work such as Blink [30] partitions data between dedicated links and PCIe using a fixed ratio, suffering from both challenges: it incurs the switching overhead of disabling peer access, and lacks any adaptive mechanism to respond to runtime PCIe interference.

Summary. Existing CCLs exhibit a fundamental structural limitation when deployed on PCIe or hybrid interconnect clusters. To fully unlock the theoretical hardware bandwidth, a novel co-design framework is needed that addresses the limitations identified above. First, for the PCIe topology, it should break the flat GPU-to-GPU abstraction and natively model hierarchical topologies, elevating the CPU as an active computational and routing node to eliminate redundant PCIe traffic. Second, in hybrid interconnect scenarios, it should incorporate runtime adaptive mechanisms to balance load across heterogeneous interconnects, rather than relying on static data partitioning that is vulnerable to PCIe interference.

3 Design

To address the two fundamental issues mentioned above, we propose HyLink. HyLink unlocks the underutilized PCIe bandwidth for collective communication and complements dedicated interconnects with PCIe bandwidth to achieve hybrid-link acceleration.

HyLink employs a two-layer co-design architecture:

- **PCIe CCL**: a collective communication library that breaks the flat P2P abstraction and accelerates PCIe collective communication through hierarchical modeling.
- **Multipath Orchestrator**: a multipath scheduling framework that performs runtime feedback-driven dynamic load balancing between dedicated interconnects and PCIe.

The end-to-end workflow of HyLink operates as follows (Figure 2). Multipath Orchestrator captures the user's collective calls through a transparent interception layer using `LD_PRELOAD` and splits the operation into two parallel sub-tasks based on a runtime splitting ratio. One sub-task is routed through the dedicated link path by invoking vendor CCLs, while the other is directed to the PCIe path via PCIe CCL. These two paths execute concurrently on independent device streams and perform a final synchronization upon completion.

3.1 PCIe CCL

To address the logical-to-physical topology mismatch in existing CCLs, PCIe CCL models collective communication as a multi-stage pipeline routed through a CPU-centric hierarchy, where the CPU is no longer a passive data relay but an active computational and routing node, thereby reducing redundant traffic. On the GPU side,

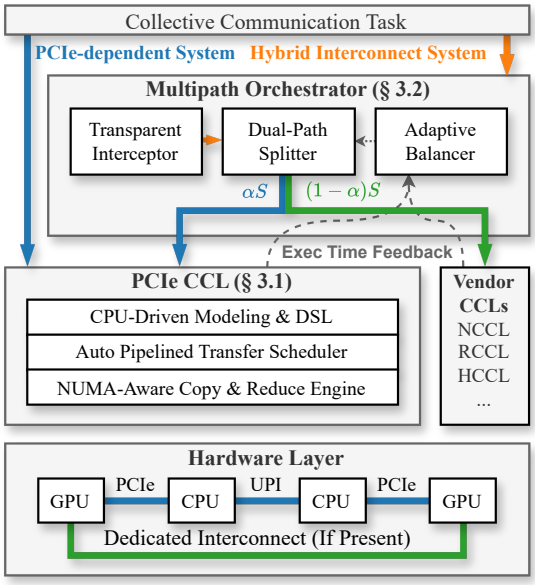


Figure 2: HyLink architecture overview.

PCIe CCL uses a lightweight spin-poll kernel on the user stream to synchronize with CPU-side completion without expensive host-device synchronization calls.

CPU-Driven Collective DSL. To express collective communication algorithms over hierarchical topologies, PCIe CCL introduces a CPU-driven DSL. Unlike existing DSLs designed for flat P2P networks (such as MSCCLang [9]), the PCIe CCL DSL natively models multi-stage data flows and incorporates the CPU as a programmable, active participant. Data movement is explicitly decomposed into four instruction types: `D2H` copies GPU data to host memory; `H2H` routes data across NUMA nodes; `REDUCE` accumulates data from multiple sources on host memory; `H2D` writes the result back to the target GPU. These four primitives are sufficient to express all standard collective communication operations.

Each instruction in the PCIe CCL DSL specifies one of the above instruction types, source and destination devices, chunk indices, and a set of data dependencies. Data dependencies are implemented through a chunk versioning mechanism. Each chunk on every NUMA node maintains an atomic version counter that tracks how many times the chunk has been written or reduced. Logically, an instruction becomes eligible for execution once all its dependent chunk versions have reached their specified values. These versioning semantics naturally accommodate both multi-source accumulation and one-to-many multicast patterns.

Figure 3 illustrates a 2-Rank AllReduce program in the PCIe CCL DSL, where the input data is divided into 2 chunks. Each rank first uploads its chunks to distinct host locations via `D2H`. Then, each rank reduces one chunk from the remote host into its local copy via `REDUCE`, which increments the target version to 2. Finally, both ranks read back the fully reduced chunks via `H2D`, gated by the version reaching 2.

Auto-Pipelined Transfer Engine. While the DSL defines the logical data flow, efficiently executing these multi-stage operations requires careful pipelining. In chunk-oriented collective communication, the tree-like topology of PCIe introduces an inherent

```
# Rank 0
D2H     gpu[0]  -> host[0]
D2H     gpu[1]  -> host[1]
REDUCE  host[0] -> host[2]  deps: host[2]>=1
H2D     host[2] -> gpu[0]   deps: host[2]>=2
H2D     host[1] -> gpu[1]   deps: host[1]>=2

# Rank 1
D2H     gpu[0]  -> host[2]
D2H     gpu[1]  -> host[3]
REDUCE  host[3] -> host[1]  deps: host[1]>=1
H2D     host[1] -> gpu[1]   deps: host[1]>=2
H2D     host[2] -> gpu[0]   deps: host[2]>=2
```

Figure 3: 2-Rank AllReduce expressed in the PCIe CCL DSL.

pipeline efficiency problem. When data is relayed between GPUs via the CPU, forming a multi-stage pipeline is necessary to fully saturate the bandwidth of both PCIe links. However, pipeline efficiency is highly dependent on the number of chunks. If the number of chunks is too small, pipeline bubbles become significant. If the number of chunks is too large, it causes a combinatorial explosion that complicates algorithm design and synthesis [28].

In flat P2P topologies, MSCCLang [9] effectively eliminates inter-stage bubbles through fused instructions (such as Recv-Copy-Send). However, this fusion assumes channel-based communication where each stage forwards data along a single channel. In the hierarchical PCIe model, the CPU-side stage may perform multi-source reduction or one-to-many multicast, which cannot be expressed under the fused instruction mechanism.

PCIe CCL solves this problem by decoupling abstraction from execution. The aforementioned *chunks* serve as the basic unit at the algorithm logic layer, while the execution engine automatically partitions each chunk into finer-grained *slices* that are invisible to the algorithm layer. The slice size is an empirically tuned parameter that balances pipeline bubbles against API launch overhead. Each chunk maintains two counters. The *version* is incremented at the start of an instruction to unblock dependent downstream stages, allowing them to enter their execution loops early. The *progress counter* tracks the actual number of slices completed, gating the real data transfer at slice granularity. Each stage in the pipeline atomically advances its progress counter upon completing each slice. Downstream stages poll the upstream counter and begin processing a slice as soon as its dependencies are satisfied. Since these stages run on independent execution contexts, this achieves full-duplex PCIe utilization with fine-grained overlap.

Efficient Host-Side Reduction. In PCIe CCL's hierarchical model, the reduction phase of collective operations (such as AllReduce) is executed by the CPU on the host side. Since host-side reduction is memory-bound, and modern CPU memory bandwidth typically exceeds PCIe bandwidth [2], the throughput of multi-core SIMD-accelerated reduction is sufficient to fully overlap with PCIe transfer latency.

The key challenge is handling multi-source reduction, where several source chunks must be accumulated into the same target. Fully concurrent accumulation causes data races on the target, while strict serialization wastes pipeline opportunities. PCIe CCL addresses this with a slot-based ordered concurrency mechanism: each source is assigned a pipeline slot in arrival order, and slot N may only process a given slice after slot $N-1$ has finished that same slice. This ordering not only prevents data races but also enables parallel execution across slots. Within each slot, multiple SIMD copy-reduce worker threads perform accumulation concurrently to saturate CPU memory channels, and slices are assigned to threads in a round-robin fashion so that earlier slices complete first.

3.2 Multipath Orchestrator

To harness both dedicated links and the PCIe bus simultaneously, Multipath Orchestrator splits collective operations across the two paths and dynamically balances their load at runtime.

Dual-Path Splitting. In hybrid interconnect scenarios, Multipath Orchestrator splits each collective communication operation into two parallel sub-tasks: the *dedicated link path* via vendor CCLs and the *PCIe path* via PCIe CCL. Given a total data size S, the splitter allocates αS to the PCIe path and $(1-\alpha)S$ to the dedicated path, where $\alpha \in [0, 1]$.

The dual paths are coordinated through an event-based fork-join model. Upon intercepting a collective call, Multipath Orchestrator records a synchronization event on the user stream. Both the dedicated link stream and the PCIe stream wait for this event before starting, ensuring input data readiness. The two paths then execute concurrently on independent streams, recording begin/end timing events for performance measurement. Finally, the user stream waits for the end events of both paths before proceeding.

Multipath Orchestrator maintains a shared parameter table that maps collective communication metadata (primitive type, data size, data type) to the splitting ratio α, shared across all ranks for global consistency. After each collective call, the adaptive balancer updates α based on the measured execution times.

Since PCIe links exhibit higher startup latency than dedicated interconnects, the splitter disables the PCIe path entirely when S falls below a configurable threshold, avoiding performance degradation in small-message scenarios.

Feedback-Driven Adaptive Control. After both paths complete, Multipath Orchestrator measures their execution times t_d (dedicated) and t_p (PCIe) using the begin/end device events. The adaptive balancer then adjusts α to equalize t_d and t_p, thereby minimizing the overall latency $\max(t_d, t_p)$.

We adopt a proportional control strategy with a dead zone. Let $r = t_d/t_p$ denote the time ratio between the two paths. When r falls within a narrow band around unity ($1-\gamma < r < 1+\gamma$), the two paths are considered balanced and α is kept unchanged. When the PCIe path is the bottleneck ($r \leq 1-\gamma$), α is halved to aggressively offload traffic back to the dedicated link. When the dedicated path is the bottleneck ($r \geq 1+\gamma$), α is scaled proportionally to the observed imbalance with a damping factor $\lambda < 1$ to prevent overshooting. The complete adaptation algorithm is given in Algorithm 1.

4 Implementation and Evaluation

We implement HyLink as two shared libraries. PCIe CCL is implemented as a standalone collective communication library with ~6200 lines of C++ code. Multipath Orchestrator is implemented as a multipath scheduling layer with ~4600 lines of C++ code. The two layers are decoupled by design: PCIe CCL can be used independently in pure PCIe environments, while Multipath Orchestrator orchestrates dual-path parallelism in hybrid interconnect scenarios. The source code is publicly available [12].

Algorithm 1 Multipath Orchestrator Adaptive Control for α

Require: Data size S, threshold $S_{\min}$, margin $\gamma \in (0, 1)$, damping $\lambda \in (0, 1)$

if $S < S_{\min}$ **then** ▷ Small-message fallback
  Use dedicated link only; **return**
end if
$r \leftarrow t_d / t_p$
if $r \leq 1 - \gamma$ **then** ▷ PCIe is bottleneck
  $\alpha \leftarrow \alpha / 2$
else if $r \geq 1 + \gamma$ **then** ▷ Dedicated link is bottleneck
  $\alpha \leftarrow \alpha \cdot r \cdot \lambda$
end if
$\alpha \leftarrow \text{clamp}(\alpha,\ 0,\ 1)$

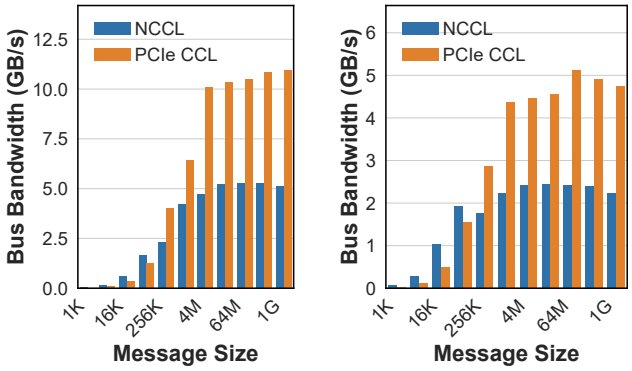
Table 2: NCCL tuning parameters swept in the benchmark.

Parameter	Values
NCCL_PROTO	Default, Simple, LL, LL128
NCCL_ALGO	Default, Ring, Tree
NCCL_P2P_DISABLE	Default, 0, 1

We evaluate HyLink through three sets of experiments: (1) microbenchmarks of PCIe CCL against NCCL in PCIe-only environments, (2) microbenchmarks of Multipath Orchestrator demonstrating the bandwidth gains of hybrid-link acceleration, and (3) a stress test validating the adaptive balancer's resilience to dynamic PCIe interference. In experiments (2) and (3), the adaptive balancer uses empirically tuned parameters γ=0.05, λ=0.97, and $S_{\min}$=1 MB.

PCIe CCL Benchmark. We evaluate PCIe CCL on 4 NVIDIA GeForce RTX 3090 GPUs connected solely via PCIe (i.e., no NVLink bridge is used), comparing against NCCL [23] on AllGather and AllReduce across varying message sizes. The host has 2 AMD EPYC 7642 sockets and 512 GB of DDR4-3200 memory (8 channels per socket, ~205 GB/s per socket), forming 2 NUMA domains. Each socket is directly connected via PCIe Gen 4 ×8 to 2 of the 4 GPUs, with no PCIe switch in the path. Because NVIDIA consumer-grade GPUs do not officially support PCIe P2P and we do not employ any P2P workaround (e.g., the tinygrad driver hack), all inter-GPU traffic is staged through host DRAM in this setup. On platforms where PCIe P2P is enabled, the headroom over NCCL may be smaller. For NCCL, we sweep several influential environment variables (Table 2) and report the maximum bandwidth across all combinations; other parameters are left default and auto-tuned by NCCL. As shown in Figure 4, PCIe CCL achieves up to 2.28× the bus bandwidth of NCCL for AllGather and up to 2.16× for AllReduce.

Multipath Orchestrator Benchmark. We evaluate Multipath Orchestrator on 2 and 4 Ascend 910 NPUs interconnected via both HCCS dedicated links and the PCIe bus on AllGather across varying message sizes, comparing against HCCL [10] using HCCS only. Our testbed is a partial-topology node rather than the full 8-NPU Ascend 910B configuration. Because HCCS is a switchless full-mesh fabric, each NPU in this setup uses only a subset of the seven HCCS link pairs, and the effective B_p/B_d here is correspondingly higher than the per-NPU aggregate listed in Table 1. As shown in Figure 5, Multipath Orchestrator achieves up to 1.4× the bus bandwidth of HCCL.



(a) AllGather Performance (b) AllReduce Performance

Figure 4: PCIe CCL vs. NCCL bus bandwidth for AllGather and AllReduce.

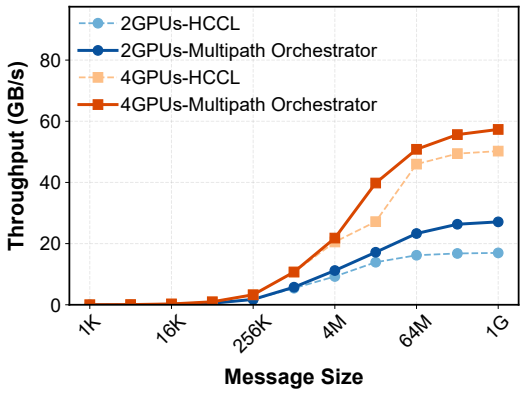


Figure 5: Multipath Orchestrator vs. HCCL throughput for AllGather.

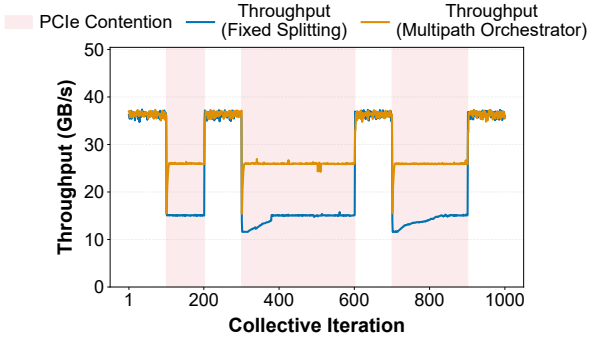


Figure 6: Resilience to Dynamic PCIe Interference

Resilience to PCIe Interference. We evaluate the resilience of Multipath Orchestrator's adaptive strategy on 2 NVIDIA V100 GPUs by simulating checkpointing-induced PCIe contention: periodic bursts of background PCIe traffic are injected via `cudaMemcpyAsync` during running. The baseline uses a fixed splitting ratio α, similar to Blink [30], pre-tuned to be optimal under zero interference, while Multipath Orchestrator dynamically adjusts α at runtime. The adaptive balancer converges within 5 iterations. As shown in Figure 6, under sustained interference, Multipath Orchestrator achieves 66% higher throughput than the fixed-ratio baseline.

5 Discussion

Data Layout in Multipath Orchestrator. The buffer splitting strategy in Multipath Orchestrator leverages element-wise semantics like AllReduce: each output element depends only on the corresponding input elements across all ranks, so the buffer can be partitioned at any offset and both paths produce results that are already correctly placed in the output buffer. For primitives like AllGather, whose output places each rank's input contiguously, splitting the send buffer causes each path to gather only a contiguous sub-buffer of each rank's data, producing partial results that must be interleaved to form the correct output. In our current AllGather implementation, each path gathers its assigned sub-buffer independently, and an additional device-side memory copy is performed afterwards to merge the partial results into the correct output layout. Candidate solutions to eliminate this copy include leveraging programmable collective frameworks such as MSCCL [9] to write outputs directly to correct offsets, or extending the dedicated CCL with a strided-buffer primitive, which would further reduce execution time and GPU memory consumption. We leave these primitive-aware splitting strategies as future work.

Scaling to Multi-Node Clusters. HyLink currently targets intra-node collective communication; extending it to multi-node clusters is feasible for both layers. First, PCIe CCL's CPU-centric model naturally extends to multi-node settings: inter-node network links can be modeled as an additional level in the hierarchy alongside NUMA nodes and PCIe buses, and the CPU-driven DSL can express cross-node data movement without structural changes. Second, for Multipath Orchestrator, enabling dual-path splitting for cross-node collectives is less effective, because NIC DMA traffic shares PCIe bandwidth with the hybrid PCIe path. Fortunately, this limitation is mitigated in practice, since prevailing training paradigms decompose parallelism into non-overlapping dimensions (TP, PP, DP) [22], where TP communication is typically confined within a single node and the three types of collectives do not execute concurrently. Multipath Orchestrator can therefore selectively accelerate TP with full hybrid-link benefits, without NIC contention from PP or DP.

6 Related Work

Collective Communication Algorithms. Extensive research has been devoted to optimizing collective communication algorithms. Classic designs such as Ring [5] and Double Binary Tree [15] trade off between bandwidth utilization and latency. For heterogeneous interconnects, BlueConnect and Themis [8, 25] decompose AllReduce along hierarchical topology levels to balance load across links. Other work targets specific physical topologies such as torus, mesh, and reconfigurable optical networks [19, 27, 32]. Since hand-crafted algorithms struggle to generalize across diverse topologies, automated synthesis has attracted growing attention. Blink [30] formulates bandwidth-optimal collectives as maximum-weight spanning tree packing. SCCL [3] first performs joint latency-bandwidth Pareto optimization via SMT solving but faces exponential blowup at larger scales. Subsequent work [4, 21, 28, 34] improves scalability through search space pruning, alternative mathematical formulations, and topology symmetry exploitation. HyLink remains orthogonal to these vendor CCL optimizations and could benefit from these techniques in the future.

Collective Communication Libraries. The systems community has built production-grade libraries integrating algorithm selection, tuning, and transport optimization. Vendor CCLs such as NCCL [23], RCCL [1], and HCCL [10] bundle predefined algorithms with heuristic selection mechanisms. AutoCCL [35] applies coordinate descent for automated parameter tuning. TCCL [16] profiles NUMA-aware PCIe costs to find optimal ring orderings for PCIe clusters. MSCCLang [9] provides a chunk-oriented DSL and low-level asynchronous primitives for expressing and executing collective algorithms on GPUs, and ResCCL [20] introduces primitive-level scheduling to maximize link utilization across micro-batches. At the transport level, recent work [26, 37] explores multi-NIC bandwidth aggregation and flexible RDMA transport. However, these libraries and frameworks model the topology as flat GPU-to-GPU edges; HyLink breaks this abstraction by natively modeling the PCIe hierarchy with the CPU as an active participant.

Communication-Computation Co-Optimization. Another line of work hides communication latency through communication-computation overlap and fusion. CoCoNet [14] provides a unified DSL for semantics-preserving transformations, enabling transformations such as fusion and overlap. Centauri [6] exposes finer-grained overlap opportunities by partitioning collective communication according to primitive type, hardware topology and workload structure. Concerto [7] formulates communication scheduling as a resource-constrained scheduling problem and uses automatic decomposition to create overlap opportunities for critical communication. Chimera [24] focuses on hybrid parallel LLMs and reduces redundant data movement by reordering operators and fusing adjacent communication operators. These techniques are orthogonal to HyLink, which optimizes the collective primitive itself.

7 Conclusion and Future Work

In this paper, we identify the underutilized PCIe bandwidth in GPU clusters for collective communication, and design HyLink, a two-layer framework that unlocks this bandwidth by natively modeling the PCIe hierarchy and harnessing dedicated interconnects and the PCIe bus. Our evaluation shows the promising potential in accelerating collective communication on both PCIe-only and hybrid interconnect systems.

While our evaluation validates the core design on multiple platforms, several directions merit further exploration. We plan to further optimize performance, integrate an automatic synthesizer for topology-aware collective schedule generation atop the DSL primitives, extend data layout optimization and multi-node scaling, and adapt to and evaluate on a broader range of hardware platforms and larger-scale clusters.

Acknowledgments

We thank our shepherd and anonymous APNet reviewers for their valuable comments. This work is supported in part by the National Key Research and Development Program (No. 2024YFB4505604), the National Natural Science Foundation of China (No. 62402025), and Huawei-BUAA Joint Lab. Menghao Zhang is the corresponding author.

References

[1] AMD. 2018. *ROCm Collective Communication Library (RCCL).* https://github.com/ROCm/rccl

[2] AMD. 2023. *4th Gen AMD EPYC Processor Architecture.* White Paper. AMD. https://www.amd.com/content/dam/amd/en/documents/products/epyc/4th-gen-epyc-processor-architecture-white-paper.pdf Accessed: 2026-03-08.

[3] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2021. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 62–75. https://doi.org/10.1145/3437801.3441620

[4] Jiamin Cao, Shangfeng Shi, Jiaqi Gao, Weisen Liu, Yifan Yang, Yichi Xu, Zhilong Zheng, Yu Guan, Kun Qian, Ying Liu, Mingwei Xu, Tianshu Wang, Ning Wang, Jianbo Dong, Binzhang Fu, Dennis Cai, and Ennan Zhai. 2025. SyCCL: Exploiting Symmetry for Efficient Collective Communication Scheduling. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) *(SIGCOMM '25)*. Association for Computing Machinery, New York, NY, USA, 645–662. https://doi.org/10.1145/3718958.3750499

[5] Ernie Chan, Robert van de Geijn, William Gropp, and Rajeev Thakur. 2006. Collective communication on architectures that support simultaneous communication over multiple links. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, New York, USA) *(PPoPP '06)*. Association for Computing Machinery, New York, NY, USA, 2–11. https://doi.org/10.1145/1122971.1122975

[6] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. 2024. Centauri: Enabling Efficient Scheduling for Communication-Computation Overlap in Large Model Training via Communication Partitioning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 178–191. https://doi.org/10.1145/3620666.3651379

[7] Shenggan Cheng, Shengjie Lin, Lansong Diao, Hao Wu, Siyu Wang, Chang Si, Ziming Liu, Xuanlei Zhao, Jiangsu Du, Wei Lin, and Yang You. 2025. Concerto: Automatic Communication Optimization and Scheduling for Large-Scale Deep Learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 198–213. https://doi.org/10.1145/3669940.3707223

[8] Minsik Cho, Ulrich Finkler, David Kung, and Hillery Hunter. 2019. BlueConnect: Decomposing All-Reduce for Deep Learning on Heterogeneous Network Hierarchy. In *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia (Eds.), Vol. 1. 241–251. https://proceedings.mlsys.org/paper_files/paper/2019/file/0c8abcf158ed12d0dd94480681186fda-Paper.pdf

[9] 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

[10] Huawei. 2024. *Huawei Collective Communication Library (HCCL).* https://gitee.com/ascend/cann-hccl

[11] Huawei Technologies Co., Ltd. 2024. *Atlas 800 Training Server Maintenance and Service Guide (Model 9000, Air Cooling) 19 - Technical Specifications.* Huawei Enterprise Support. https://support.huawei.com/enterprise/en/doc/EDOC1100141951/43611d26/technical-specifications Accessed: 2026-03-02.

[12] HyLink authors. 2026. *Source Code for HyLink.* https://github.com/Networked-System-and-Security-Group/HyLink

[13] Intel Corporation. 2022. *Intel® Xeon® Processor Scalable Family Technical Overview.* Technical Overview 673025. Intel Corporation. https://www.intel.com/content/www/us/en/developer/articles/technical/xeon-processor-scalable-family-technical-overview.html Accessed: 2024-05-20.

[14] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. 2022. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 402–416. https://doi.org/10.1145/3503222.3507778

[15] Sylvain Jeaugey. [n. d.]. *Massively Scale Your Deep Learning Training with NCCL 2.4.* NVIDIA Corporation. https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/

[16] Heehoon Kim, Junyeol Ryu, and Jaejin Lee. 2024. TCCL: Discovering Better Communication Paths for PCIe GPU Clusters. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 999–1015. https://doi.org/10.1145/3620666.3651362

[17] Jayacharan Kolla, Pedram Alizadeh, and Gilbert Lee. 2025. Understanding RCCL Bandwidth and xGMI Performance on AMD Instinct MI300X. AMD ROCm Blogs. https://rocm.blogs.amd.com/software-tools-optimization/mi300x-rccl-xgmi/README.html Accessed: 2026-02-25.

[18] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. 2021. Ascend: a Scalable and Unified Architecture for Ubiquitous Deep Neural Network Computing : Industry Track Paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 789–801. https://doi.org/10.1109/HPCA51647.2021.00071

[19] Dongkyun Lim and John Kim. 2025. TidalMesh: Topology-Driven AllReduce Collective Communication for Mesh Topology . In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE Computer Society, Los Alamitos, CA, USA, 1526–1540. https://doi.org/10.1109/HPCA61900.2025.00114

[20] Tongrui Liu, Chenyang Hei, Fuliang Li, Chengxi Gao, Jiamin Cao, Tianshu Wang, Ennan Zhai, and Xingwei Wang. 2025. ResCCL: Resource-Efficient Scheduling for Collective Communication. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) *(SIGCOMM '25)*. Association for Computing Machinery, New York, NY, USA, 55–70. https://doi.org/10.1145/3718958.3750514

[21] Xuting Liu, Behnaz Arzani, Siva Kesava Reddy Kakarla, Liangyu Zhao, Vincent Liu, Miguel Castro, Srikanth Kandula, and Luke Marshall. 2024. Rethinking Machine Learning Collective Communication as a Multi-Commodity Flow Problem. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia) *(ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 16–37. https://doi.org/10.1145/3651890.3672249

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

[23] NVIDIA. 2015. *NVIDIA Collective Communication Library (NCCL).* https://github.com/NVIDIA/nccl Accessed: 2024-05-21.

[24] Le Qin, Junwei Cui, Weilin Cai, and Jiayi Huang. 2025. Chimera: Communication Fusion for Hybrid Parallelism in Large Language Models. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 498–513. https://doi.org/10.1145/3695053.3731025

[25] Saeed Rashidi, William Won, Sudarshan Srinivasan, Srinivas Sridharan, and Tushar Krishna. 2022. Themis: a network bandwidth-aware collective scheduling policy for distributed training of DL 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, 581–596. https://doi.org/10.1145/3470496.3527382

[26] Zhenghang Ren, Yuxuan Li, Zilong Wang, Xinyang Huang, Wenxue Li, Kaiqiang Xu, Xudong Liao, Yijun Sun, Bowen Liu, Han Tian, Junxue Zhang, Mingfei Wang, Zhizhen Zhong, Guyue Liu, Ying Zhang, and Kai Chen. 2025. Enabling efficient GPU communication over multiple NICs with FuseLink. In *Proceedings of the 19th USENIX Conference on Operating Systems Design and Implementation* (Boston, MA, USA) *(OSDI '25)*. USENIX Association, USA, Article 6, 18 pages. https://dl.acm.org/doi/10.5555/3767901.3767907

[27] Daniele De Sensi, Tommaso Bonato, David Saam, and Torsten Hoefler. 2024. Swing: Short-cutting Rings for Higher Bandwidth Allreduce. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1445–1462. https://www.usenix.org/conference/nsdi24/presentation/de-sensi

[28] 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

[29] Debendra Das Sharma. 2024. PCI-Express: Evolution of a Ubiquitous Load-Store Interconnect Over Two Decades and the Path Forward for the Next Two Decades. *IEEE Circuits and Systems Magazine* 24, 2 (2024), 47–61. https://doi.org/10.1109/MCAS.2024.3373556

[30] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Nikhil Devanur, Jorgen Thelin, and Ion Stoica. 2020. Blink: Fast and Generic Collectives for Distributed ML. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 172–186. https://proceedings.mlsys.org/paper_files/paper/2020/file/cd3a9a55f7f3723133fa4a13628cdf03-Paper.pdf

[31] Guanhua Wang, Chengming Zhang, Zheyu Shen, Ang Li, and Olatunji Ruwase. 2024. Domino: Eliminating Communication in LLM Training via Generic Tensor Slicing and Overlapping. arXiv:2409.15241 [cs.DC] https://arxiv.org/abs/2409.15241

[32] 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

[33] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 945–960. https://www.usenix.org/conference/nsdi22/presentation/weng

[34] William Won, Midhilesh Elavazhagan, Sudarshan Srinivasan, Swati Gupta, and Tushar Krishna. 2024. TACOS: Topology-Aware Collective Algorithm Synthesizer for Distributed Machine Learning. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 856–870. https://doi.org/10.1109/MICRO61859.2024.00068

[35] Guanbin Xu, Zhihao Le, Yinhe Chen, Zhiqi Lin, Zewen Jin, Youshan Miao, and Cheng Li. 2025. AutoCCL: Automated Collective Communication Tuning for Accelerating Distributed and Parallel DNN Training. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 667–683. https://www.usenix.org/conference/nsdi25/presentation/xu-guanbin

[36] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2025. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] https://arxiv.org/abs/2303.18223

[37] Yang Zhou, Zhongjie Chen, Ziming Mao, ChonLam Lao, Shuo Yang, Pravein Govindan Kannan, Jiaqi Gao, Yilong Zhao, Yongji Wu, Kaichao You, Fengyuan Ren, Zhiying Xu, Costin Raiciu, and Ion Stoica. 2025. An Extensible Software Transport Layer for GPU Networking. arXiv:2504.17307 [cs.NI] https://arxiv.org/abs/2504.17307