UDMP: Unified Delay-Driven Multipath Protocol for AI Clusters

Chengyuan Huang, *Member, IEEE*, Zhengqi Cui, Jun Xu, Zhaochen Zhang, Li Wang, Peirui Cao, Zhongming Ji, Jilei Chen, Dongxu Wang, Shengju Zhang, Lingkun Meng, Ahmed M. Abdelmoniem, *Senior Member, IEEE*, Fu Xiao, *Senior Member, IEEE*, Wanchun Dou, Guihai Chen, *Fellow, IEEE*, Keqiang He, and Chen Tian, *Senior Member, IEEE*

***Abstract*—Distributed AI model training generates bursty, low-entropy elephant flows that challenge existing single-path transport protocols in multi-stage Clos networks, leading to congestion and inefficiency. Multipath transport emerges as a promising solution, leveraging multiple paths to balance traffic and enhance resilience. However, current multipath RDMA solutions suffer from scalability, congestion control, and load-balancing inefficiencies. This paper introduces Unified Delay-driven Multipath Protocol (UDMP), a novel approach that co-designs congestion control and load balancing using network delay as a unified signal. UDMP employs delay-gradient-based congestion control to precisely resolve unavoidable congestion. Moreover, UDMP leverages delay-assisted load balancing to shift traffic across paths with minimal latency adaptively, maintaining throughput when encountering avoidable congestion. A novel Token Pool design integrates these components, eliminating per-path state overhead while achieving fine-grained traffic distribution. Implementations on DPDK and NS3 demonstrate that UDMP achieves up to 2x higher throughput and reduces flow completion times by up to 30% compared to state-of-the-art methods like MPRDMA and QP-Scaling. These results highlight UDMP's effectiveness in meeting the stringent performance requirements of modern distributed AI training workloads.**

Received 15 January 2025; revised 29 October 2025 and 17 December 2025; accepted 22 December 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor B. Ji. Date of publication 29 December 2025; date of current version 12 January 2026. This work was supported in part by the National Key Research and Development Program of China under Grant 2024YFB2900070, in part by the Key Program of Natural Science Foundation of Jiangsu under Grant BK20243053, in part by the National Natural Science Foundation of China under Grant 62502194 and Grant 62325205, in part by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China under Grant JYB2025XDXM103, in part by the U.K. Research and Innovation (UKRI)-Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/X035085/1, and in part by Nanjing University-China Mobile Communications Group Company Ltd. Joint Institute. *(Corresponding authors: Keqiang He; Zhongming Ji.)*

Chengyuan Huang, Zhengqi Cui, Zhaochen Zhang, Li Wang, Peirui Cao, Wanchun Dou, Guihai Chen, and Chen Tian are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: ryanhuang1014@gmail.com; c3485157618@163.com; zhaochenzhang@smail.nju.edu.cn; iswangli@foxmail.com; caopeirui@nju.edu.cn; douwc@nju.edu.cn; gchen@nju.edu.cn; tianchen@nju.edu.cn).

Jun Xu, Zhongming Ji, Jilei Chen, Dongxu Wang, Shengju Zhang, and Lingkun Meng are with China Mobile (Suzhou) Software Technology Company Ltd., Suzhou 215000, China (e-mail: xujun@cmss.chinamobile.com; jizhongming@cmss.chinamobile.com; chenjilei@cmss.chinamobile.com; wangdongxu@cmss.chinamobile.com; zhangshengju@cmss.chinamobile.com; menglingkun@cmss.chinamobile.com).

Ahmed M. Abdelmoniem is with the Queen Mary University of London, E1 4NS London, U.K. (e-mail: ahmed.sayed@qmul.ac.uk).

Fu Xiao is with Nanjing University of Posts and Telecommunications, Nanjing 210023, China (e-mail: xiaof@njupt.edu.cn).

Keqiang He is with Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: kqhe@cs.sjtu.edu.cn).

Digital Object Identifier 10.1109/TON.2025.3649183

***Index Terms*—Distributed AI training, multipath RDMA, delay-driven protocol, congestion control, load balancing, token pool.**

I. INTRODUCTION

DISTRIBUTED AI model training has become a cornerstone for scaling the training of modern large-scale models (*e.g.*, LLMs), enabling efficient utilization of distributed computational and memory resources. Today's AI training clusters predominantly employ multi-stage Clos network architectures [1], [2], [3], [4], which provide extensive network capacity and numerous paths between any pair of computing nodes. However, unlike traditional cloud data center workloads, distributed AI training generates low-entropy, bursty elephant flows [2], [5] driven by hardware accelerators. These low-entropy bursty flows often encounter heavy congestion when relying on single-path transport solutions (*e.g.*, RoCEv2 [6]), primarily due to ECMP hash collisions [7], [8], [9], [10]. There is a fundamental mismatch between single-path transport and AI training traffic patterns, making single-path transport ineffective in such scenarios. Additionally, single-path transports struggle to handle sporadic network device failures, such as link disruptions or switch port flapping. As a result, distributed AI training imposes significant challenges on data center network design, necessitating the development of advanced transport protocols tailored to the demanding performance requirements of distributed AI model training.

Multipath transport offers a promising solution to these challenges. By allowing a source-destination pair to transmit data concurrently across multiple paths, it increases traffic entropy and achieves a more balanced traffic distribution across network links. This maximizes the utilization of the massive capacity offered by multi-stage Clos networks while minimizing the risk of congestion on any single link. Moreover, in the event of a path failure, the remaining paths ensure uninterrupted connectivity, enhancing system resilience and performance. Therefore, multipath transport has the potential to effectively meet the stringent performance demands of modern AI applications [2], [5], [11], [12], where the tail flow completion time often dictates the overall completion time of collective communication tasks.

However, there are multiple challenges to designing and implementing a scalable and efficient multipath transport protocol for AI/ML training clusters. ① The first challenge is the resource constraints on the Network Interface Card (NIC) hardware. Remote Direct Memory Access (RDMA)

is the primary communication protocol [13], [14], [15] in AI/ML training clusters, implemented directly on the NIC to offload packet processing and memory operations from the CPU. This enables low-latency, high-throughput data transfers and direct memory access between source and destination. However, RDMA's hardware-based implementation imposes a scalability challenge. Due to cost and power constraints, the on-chip memory resources are generally limited to a few megabytes [11], [16], [17]. Consequently, multipath RDMA design must remain simple to avoid introducing significant memory overheads to the NIC. ② The second challenge lies in the effective and precise multipath congestion control. When a source node sends data concurrently across multiple paths to a destination node, it must adjust its sending rate based on the type of congestion encountered. For instance, in the case of avoidable hotspots (*e.g.*, ECMP elephant flow collisions), the congestion control mechanism should avoid decreasing the source's sending rate. Instead, it should reroute traffic through less congested or idle network paths. Conversely, if congestion is unavoidable (*e.g.*, incast [18], [19], [20]), the protocol must promptly reduce the sending rate to minimize queuing latency, ensuring efficient data transfer without worsening congestion. ③ The third challenge is adaptive and fine-grained load balancing. Once congestion control determines the appropriate sending rate, the multipath transport's load balancing must dynamically adjust traffic distribution across multiple network paths. For example, in the face of avoidable congestion in the fabric, the source should quickly shift traffic from congested paths to non-congested ones. Further, the granularity of traffic load balancing should be as fine as possible to minimize the risk of switch head-of-line blocking, ensuring low end-to-end latency.

Unfortunately, state-of-the-art multipath RDMA transport protocols [5], [11], [21], [22] fail to address the three key challenges discussed above. We highlight the shortcomings of existing solutions as follows: (1) *QP Scaling*: The most straightforward approach to enabling multipath RDMA transport is QP Scaling [5], [21], [22], which binds multiple Queue Pairs (QPs) between the source and destination NICs to support multipath communication. In this approach, the source node posts messages to multiple QPs, with each QP corresponding to a distinct network path, tracking its state and adapting to real-time congestion. While this method effectively increases traffic entropy and improves load balancing, it incurs significant on-chip memory overhead on the NIC due to the need to maintain per-path states. Given the limited on-chip memory resources, QP Scaling reduces the number of concurrent connections the NIC can support—more QPs result in fewer supported connections. In large-scale training tasks like the AllReduce phase, a single node must establish connections with numerous peers. Consequently, QP Scaling fails to meet the scalability requirements of large-scale training workloads. (2) *MPRDMA*: To address hardware resource challenges in QP Scaling, MPRDMA [11] introduces a multipath RDMA transport using only per-connection states. Instead of maintaining separate control states for each path, MPRDMA uses a single *cwnd* for all paths. This *cwnd* is adjusted dynamically: it increases by $1/cwnd$ when an ACK without an ECN mark is received and decreases by $1/2$ when an ACK with an ECN mark is received. A returning ACK can clock out new packets, provided the *cwnd* allows it. This approach achieves excellent scalability by reducing on-chip memory requirements. However, the absence of per-path states results in poor coordination across paths. Without a path-aware traffic scheduling mechanism, MPRDMA cannot distinguish between avoidable and unavoidable congestion, leading to unnecessary rate reductions for avoidable congestion. Additionally, the time required to shift traffic from congested paths to lighter-loaded ones becomes excessively long (§ II), significantly reducing overall connection throughput.

To overcome these limitations, this paper proposes a Unified Delay-driven Multipath Protocol (UDMP), a low-overhead, high-performance multipath RDMA transport protocol for AI training networks within a data center. UDMP *co-designs congestion control and load balancing and leverages network delay as the unified signal for both connection-level congestion control and path-aware traffic load balancing*. UDMP comprises two core components: ① delay-gradient-based congestion control and ② delay-assisted adaptive load balancing. The congestion control algorithm in UDMP utilizes the gradient of average delay across multiple paths to regulate the sending rate. This design ensures that UDMP maintains high connection throughput in the presence of avoidable congestion, as delay increases on individual paths do not immediately impact the average delay. In contrast, during unavoidable congestion scenarios, such as incast, the perceived average delay rises promptly, enabling UDMP to quickly reduce the sending rate, thereby alleviating congestion and preserving network stability. The load balancing mechanism in UDMP leverages returning ACKs to clock out packets across multiple paths. By exploiting the fact that ACKs from less congested paths return more quickly and exhibit lower end-to-end delays, UDMP's delay-assisted ACK-clocking strategy ensures that these faster ACKs clock out more packets. This approach naturally shifts traffic from highly delayed paths to those with lower delays, optimizing traffic distribution and maintaining low latency. Finally, UDMP integrates connection-level congestion control and path-aware traffic load balancing through an innovative *Token Pool* design. The congestion control algorithm injects tokens (representing the number of packets) into the token pool based on the calculated sending rate, while the load balancing module retrieves tokens from the pool before transmitting packets. By coupling congestion control and load balancing in this manner, UDMP eliminates the need for maintaining per-path control states while enabling adaptive and fine-grained, path-aware traffic load balancing.

We implemented UDMP using both the DPDK framework and the NS3 simulation platform, enabling comprehensive evaluations through testbed experiments and large-scale simulations across diverse network topologies and traffic workloads. The results show that UDMP delivers up to a 2x improvement in throughput and up to a 30% reduction in flow completion time compared to state-of-the-art multipath RDMA transports, MPRDMA and QP-Scaling. In AI model training scenarios, UDMP reduces tail flow completion time by 28% for AlltoAll operations and 70% for AllReduce operations relative to MPRDMA, demonstrating its effectiveness as a high-performance multipath RDMA transport for distributed AI training workloads.

The rest of the paper is organized as follows: § II introduces the background and motivation for multipath RDMA transport. § III outlines the design of UDMP, while § IV describes its implementation using the DPDK framework. § V presents the evaluation results of UDMP, comparing its performance with

state-of-the-art multipath RDMA transports. § VII discusses related work and § VIII concludes the paper.

II. Background and Motivation

In this section, we first describe AI training networks and their workload characteristics in § II-A. Then, we provide an overview of RDMA and RDMA over Converged Ethernet (RoCE) in § II-B. Finally, we highlight the need for multipath RDMA transport and discuss the challenges of designing and implementing multipath RDMA in § II-C.

A. AI Training Networks and Workload Characteristics

Distributed Model Training. Distributed AI model training has become a cornerstone for scaling the training of modern large-scale models, enabling efficient utilization of distributed computational and memory resources. By partitioning workloads across multiple devices, distributed training facilitates the training of models with billions of parameters, such as large language models (LLMs). Distributed AI model training relies on three primary parallelism methods: data parallelism, model parallelism, and pipeline parallelism, each addressing specific challenges in scaling large models. Data Parallelism [23] replicates the entire model across multiple devices, with each replica processing a unique subset of the input data. After computing local gradients, devices synchronize their updates to ensure model consistency, typically using techniques like all-reduce operations. Model Parallelism [24] divides the model itself across multiple devices, assigning different layers or components of the model to separate nodes. This is particularly beneficial for models that are too large to fit into the memory of a single device. Pipeline Parallelism [25], [26] partitions the model into sequential stages, each processed by a different device, enabling the forward and backward passes to be pipelined.

AI Training Networks. Distributed AI model training requires high-performance networks to ensure low-latency and high-throughput communication. In AI training clusters, each host is typically equipped with 4 to 8 GPUs. These hosts are provisioned with multiple high-speed NICs, ensuring sufficient bandwidth for inter-node communication. The NICs are connected to a non-blocking multi-stage Clos network [1], [2], [5], which supports the massive network capacity required in modern data centers. Furthermore, the multi-stage Clos network provides abundant path diversity between any pair of nodes, enabling robust connectivity and fault tolerance. Open-standard Ethernet-based AI training fabrics have been widely adopted and actively optimized by industry leaders such as AMD, Alibaba, and Meta [2], [5], [27]. In addition to Ethernet-based fabrics, proprietary interconnect technologies like InfiniBand [28], NVLink/NVSwitch [29], and Optical Circuit Switch (OCS)-enabled Inter-Chip Interconnect (ICI) [30] have also been proposed and deployed.

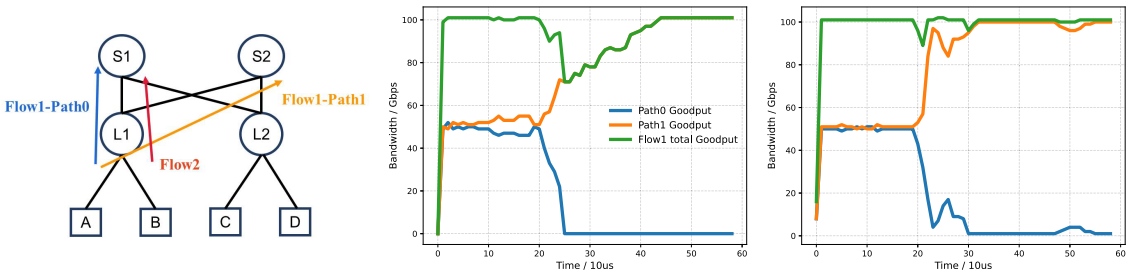
AI Training Traffic Patterns. Compared with workloads running in the cloud, traffic patterns in AI training clusters exhibit two distinct characteristics: (1) *Low Entropy*: Cloud computing workloads generate millions of flows, resulting in a high-entropy traffic profile, where individual flows are typically continuous but low-utilization. In contrast, AI training workloads generate significantly fewer flows, leading to a low-entropy traffic profile that presents unique challenges for effective traffic load balancing in path-rich multi-state Clos networks [5]. (2) *Bursty Elephant Flows*: The collective communication inherent to AI training jobs creates periodic and synchronized traffic bursts. Each bursty flow can saturate the NIC's capacity (*e.g.*, 400Gbps as reported in HPN [2]). These synchronized bursty elephant flows substantially increase peak network utilization, reaching levels of 70-80% compared to the 30-50% typically seen in cloud data center networks [12]. The unique combination of low-entropy and bursty elephant flow traffic patterns in AI training workloads severely limits the effectiveness of traditional hash-based flow-level load balancing schemes, such as Equal-Cost Multi-Path (ECMP) and Weighted-Cost Multi-Path (WCMP) [31]. These schemes are prone to issues like elephant flow collisions [7], [10] and hash polarization [32], which can significantly degrade tail flow completion times. This is particularly problematic for AI training jobs, which are inherently sensitive to tail flow completion times due to their synchronous nature. Many training algorithms, including stochastic gradient descent (SGD), require global synchronization, where all nodes must wait for the slowest node to complete its task before proceeding. Consequently, prolonged flow completion times can cause iteration delays, ultimately increasing training time and reducing overall training efficiency.

Sensitivity to Network Failures. Link and switch failures are not uncommon in large-scale data center networks [33]. For instance, large-scale production AI training clusters have reported 5,000 to 60,000 link flapping incidents per day [2], underscoring the prevalence of such disruptions in real-world environments. AI training jobs use distributed computing frameworks where GPUs across multiple nodes collaborate on a single task, requiring frequent and high-bandwidth communication to synchronize parameters. Network failures such as link flapping and link down delay synchronization and slow down the whole training process. As a result, AI training workloads are highly sensitive to network failures, emphasizing the critical need for robust and adaptive networking solutions to mitigate these disruptions and ensure smooth operation.

B. RDMA and RoCE

Traditional transport protocols like TCP often fall short of meeting the high-throughput and low-latency demands of AI training workloads, leading to suboptimal performance and bottlenecks [34]. In contrast, Remote Direct Memory Access (RDMA) is widely preferred in these scenarios for its superior performance. By bypassing the kernel and CPU during data transmission, RDMA enables direct memory access between nodes, effectively eliminating unnecessary processing overhead. This results in reduced latency and lower CPU utilization—critical factors in AI training environments where GPUs across multiple nodes must exchange large volumes of data, such as gradients and weights, at high speed.

RDMA over Converged Ethernet (RoCE) is commonly adopted in AI training clusters because it enables RDMA functionality over standard Ethernet networks, which are widely deployed in data centers [2], [5], [12], [13], [14], [27]. Unlike InfiniBand, which requires specialized hardware and infrastructure, RoCE integrates seamlessly with existing Ethernet-based networking equipment, reducing deployment costs. Additionally, RoCE supports lossless or near-lossless communication through mechanisms like Priority Flow Control (PFC), ensuring reliable data transmission without packet drops. RoCE's ability to leverage Ethernet while providing RDMA's performance benefits makes it a practical and



(a) Leaf-Spine topology, flow1 starts at 0μs and flow2 starts at 200μs.

(b) Throughput change of flow1 across two paths using MPRDMA.

(c) Throughput change of flow1 across two paths using UDMP.

Fig. 1. UDMP achieves near-optimal congestion-aware multi-pathing, demonstrating faster convergence than MPRDMA without compromising flow throughput.

cost-effective choice for scaling AI training workloads in large-scale data centers.

C. Need for Multipath RDMA Transport

The unique workload characteristics of AI training have important implications for network design: (1) the transport in AI networks must be optimized to effectively load balance low-entropy, bursty elephant flows in path-rich Clos networks; and (2) the sensitivity of AI training workloads to network failures demands a robust transport capable of tolerating failures, minimizing downtime, and ensuring uninterrupted training.

We observe that multipath RDMA transport naturally bypasses network hotspots and failures (*e.g.*, link flapping and link down), thereby improving system throughput and robustness. Furthermore, multipath RDMA transport effectively balances low-entropy, bursty elephant flows generated by GPUs across multiple paths, addressing challenges such as elephant flow collisions [7], [10] and hash polarization [32]. Therefore, multipath RDMA transport emerges as a promising solution to meet the high-throughput, low-latency, and robustness requirements of AI training workloads. However, designing and implementing efficient and practical multipath RDMA presents several challenges.

Challenge #1: Multipath RDMA with Limited NIC On-chip Memory. Protocols like Multi-Path TCP (MPTCP) [35] enable the simultaneous use of multiple network paths to improve throughput and reliability. However, MPTCP maintains per-path congestion windows as part of its congestion control mechanism. While maintaining per-path congestion control states is feasible in MPTCP, as it is implemented in the kernel on the end host, doing so for RDMA is prohibitively expensive. This limitation arises because RDMA is implemented on the NICs and on-chip memory available in RDMA NICs is quite limited (only a few MBs [11], [16], [17]). The on-chip memory overhead of multipath RDMA scales linearly with the number of network paths and the number of connections, both of which increase as AI clusters continue to grow in scale. Recent approaches to scaling RDMA have utilized Queue Pairs (QPs) to enhance RDMA transport [5], [22], [36]. With QP scaling, communication between GPU pairs is conducted across multiple QPs. This design improves traffic entropy and facilitates better load balancing between GPUs, with the collective communication library scheduling RDMA verb calls over QPs created on the source and destination NICs. However, each QP consumes hardware resources, and a large number of QPs can quickly exhaust the NIC's on-chip memory, leading to degraded application performance [37], [38]. Additionally, managing a high number of QPs increases system complexity, making configuration and maintenance more challenging [11], [36].

Challenge #2: Effective and Precise Multipath Congestion Management. Traditional single-path transport protocols (*e.g.*, RoCEv2) manage congestion control by monitoring the congestion level of a single path. The sender adaptively adjusts its sending rate based on congestion feedback, such as ECN notifications or delay signals. In contrast, congestion control in multipath RDMA introduces additional complexities: (1) Multipath RDMA transport must account for congestion levels across multiple paths without maintaining per-path states due to the limited on-chip memory on RDMA NICs; (2) Multipath RDMA transport must precisely handle two types of congestion scenarios, termed *avoidable congestion* and *unavoidable congestion* in this paper, which are prevalent in AI training clusters. As discussed earlier, AI training flows are characterized by synchronized, bursty elephant flows with low entropy. Consequently, classic flow-level load-balancing schemes, such as ECMP and WCMP [31], perform poorly, leading to issues like elephant flow collisions [7], [10] and hash polarization [32]. We define congestion caused by suboptimal load balancing as *avoidable congestion* because it can be fully mitigated via optimized load-balancing techniques. The second type of congestion, *unavoidable congestion*, occurs in scenarios such as incast [18], [19], [20]. Multipath RDMA transport must differentiate between these two types of congestion and respond accordingly. For avoidable congestion, senders should reroute traffic to uncongested paths without reducing their sending rates. However, our findings reveal that the state-of-the-art multipath RDMA transport, MPRDMA, unnecessarily reduces the sending rate for avoidable congestion and experiences slow traffic shifts from congested to non-congested paths (as illustrated in Figure 1). This leads to increased flow completion times and decreased overall efficiency.

Challenge #3: Adaptive and Fine-grained Traffic Load Balancing. The load balancing functionality of multipath RDMA determines how to distribute the allowed traffic rate, as calculated by congestion control, across multiple paths. To minimize the impact of flow collisions and network failures, load balancing should operate at fine granularity, ideally at the packet level. When congestion control identifies avoidable congestion that does not warrant rate reduction, load balancing must promptly redirect traffic from congested paths to less congested or idle ones. Similarly, in response to temporary congestion caused by microbursts [39], [40] and failures such as link flapping, the load balancing scheme should dynamically reduce the number of packets sent to affected paths.

In summary, multipath RDMA transport needs to achieve adaptive, fine-grained traffic load balancing in path-rich Clos networks, implement effective multipath congestion control strategies that differentiate between avoidable and unavoidable congestion, and minimize on-chip memory overhead.

III. UDMP Design

In this section, we begin by presenting an overview of UDMP's design in § III-A. We then discuss the two core building blocks of UDMP—delay gradient-based congestion control and delay-assisted adaptive traffic load balancing in § III-C and § III-B, respectively. Finally, we dive into additional design details, including the handling of out-of-order packets, retransmission, and new path exploration, in § III-D.

A. Overview

We propose UDMP, a congestion-aware multipath RDMA transport protocol for AI/ML clusters, which seamlessly integrates congestion control (CC) and traffic load balancing (LB) to fully utilize the abundant multipath resources in modern data center fabrics [2], [3], [4].

The first design goal we aim to achieve in UDMP is to avoid keeping per-path states in the NIC's memory. RDMA is implemented on NICs, but the on-chip memory available is limited (i.e., only a few MBs) and expensive. As demonstrated in MPRDMA, Mellanox ConnectX-3 Pro NICs can support only around 200 connections. Introducing per-path states would further diminish the number of connections that RDMA NICs can manage. To address this challenge, UDMP should only maintain a global congestion window or sending rate per connection, irrespective of the number of network paths being used. The second design goal of UDMP is to differentiate between *avoidable* and *unavoidable* network congestion and handle them accordingly. The key distinction lies in how the system responds to these two types of congestion. In scenarios of avoidable congestion, senders do not need to reduce their sending rates. Instead, they can route packets through less congested network paths, effectively bypassing congested areas without sacrificing throughput. The third design goal of UDMP is to avoid the limitations of flow-level traffic load balancing schemes (*e.g.*, ECMP) and adopt a fine-grained, packet-level load balancing strategy. This strategy should enable the system to quickly and adaptively send packets through less congested network paths, improving throughput and reducing latency.

To achieve the aforementioned goals, UDMP co-designs congestion control and multipath traffic load balancing—two critical components for building a high-performance RDMA transport in AI/ML clusters. The core idea of UDMP is to *adaptively clock out packets across multiple paths based on the observed RTTs from returning ACKs, while leveraging the gradient of the average RTT across these paths to determine the optimal sending rate for a connection.* Figure 2 illustrates a high-level overview of UDMP's design.

To simultaneously realize connection-level congestion control and path-level load balancing, we introduce the *Token Pool* as a bridge between them. In UDMP, tokens are generated according to the rate computed by the congestion control module, which defines the overall sending rate of a multipath connection. Each returning ACK then acquires an available token from the pool to trigger the transmission of new packets along the same path, forming an ACK-clocked load balancing.

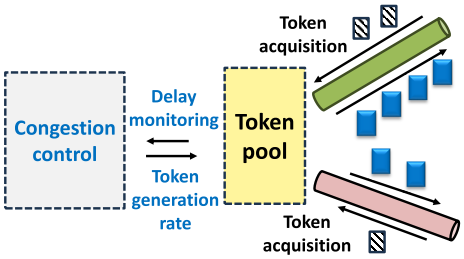


Fig. 2. UDMP design overview.

This design enables fine-grained per-path rate allocation while maintaining global consistency with the connection-level sending rate. In details, (1) For avoidable congestion, we leverage the fact that ACK packets from less congested paths return to the sender faster than those from overloaded paths. By making the fast ACK packets clock out more new packets along the same path, UDMP enables rapid load shifting. This approach allows UDMP to maintain overall throughput by bypassing hotspots. Notably, UDMP leverages the RTTs of returning ACKs to infer end-to-end delays and adaptively distribute traffic across multiple paths, eliminating the need for per-path states and minimizing the on-chip memory overhead on the NICs.

(2) For unavoidable congestion, we utilize the average delay gradient to assess the overall congestion level quickly. When congestion occurs, the RTT of specific paths will increase first, but the overall average RTT will remain stable. At this point, UDMP's adaptive traffic load balancing mechanism will activate, attempting to bypass the hotspot while maintaining throughput. If shifting traffic across paths does not alleviate the congestion, the average delay will rise. Consequently, connection-level congestion control (CC) detects congestion promptly through the gradient and adjusts the overall sending rate accordingly.

Note that UDMP leverages network delay as the unified signal to determine both traffic load balancing and congestion control. This is a valuable feature because delay serves as a universal multi-bit congestion signal [41], [42] and does not require additional configuration of network switches or tuning of ECN thresholds [14], [43].

B. Delay Gradient-Based Congestion Control

1) Delay Gradient: Delay-based congestion control algorithms [41], [42], [44] rely on RTT increases as indicators of network congestion. When RTT rises persistently, these algorithms reduce the sending rate to alleviate congestion. A challenge in extending delay-based congestion control to multipath scenarios lies in handling multiple paths, each of which can independently become a bottleneck and drive up queuing latency. Determining a single baseline RTT for all paths is particularly challenging in such situations. To address this, UDMP leverages delay gradients instead of absolute delay as the congestion control signal. Delay gradients are highly sensitive to variations in queuing delay, enabling effective detection of congestion dynamics without requiring fine-tuning of RTT baselines.

UDMP achieves low latency by responding to delay gradients, effectively reacting to the process of queue buildup rather than waiting to observe whether the queue size has exceeded a threshold. A positive delay gradient, indicated by an increase in RTT, represents queue growth, while a negative gradient

signals queue reduction. By leveraging delay gradients, UDMP can respond to queue buildup proactively without waiting for the queue to fully form. This approach allows us to detect congestion even *earlier* than ECN, as ECN requires the queue to reach a threshold before issuing a signal.

Single-hop model. Now we develop a single-hop model for UDMP to quantify the relationship between delay gradient and queue dynamics. We assume a network with N end hosts, each transmitting data packets to the network at a rate of $y_i(t)$ $(i = 1, \ldots, N)$. The network is abstracted as a single-hop network with a link bandwidth of C, meaning that the output rate is constrained to $\leq C$. Let $q(t)$ represent the queue size in the network. The change in $q(t)$ is determined by the relationship between the network's input and output rates. Thus, we can define the change in $q(t)$ as:

$$\Delta q(t) = \left(\sum_{i=1}^{N} y_i(t) - C \right) \cdot \Delta(t), \tag{1}$$

where $\sum_{i=1}^{N} y_i(t)$ represents the total input rate, and C is the fixed output rate of the link. Here, we assume that the rate changes once every Δt time interval, *i.e.*, the difference between the input and output rates is constant. Consequently, Equation 1 reflects the queue dynamics in a single-hop model, where the queue length increases or decreases as the aggregate input rate exceeds or falls below the link capacity.

Additionally, we establish the linear relationship between queue buildup and RTT variation as follows:

$$\Delta q(t) = \Delta \text{RTT} \cdot C, \tag{2}$$

Combining Equation 1 and Equation 2 yields Equation 3, which links the RTT change directly to the difference between the input and output rates, forming the theoretical foundation for using RTT gradient as a congestion signal.

$$\sum_{i=1}^{N} y_i(t) - C = \frac{\Delta \text{RTT} \cdot C}{\Delta t}. \tag{3}$$

If the RTT increases during the Δt interval, we can infer that the current rate is too high and should be reduced by $\Delta \text{RTT} \cdot C / \Delta t$. Conversely, the rate should be increased accordingly. This equation guides the rate adaptation of the congestion control, shown in line 33 and 35 of Algorithm 1.

2) CC Control Loop: As discussed in § III-A, UDMP maintains connection-level states rather than path-level states. It estimates a connection-level sending rate based on the gradient of the average delay across multiple paths and converts this rate into the number of packets (tokens) allowed for transmission. Then, the adaptive load balancing module (detailed in § III-C) leverages RTTs observed on each path to determine how packets are distributed across multiple paths.

The detailed congestion control (CC) algorithm is presented in Algorithm 1. The high-level logic is as follows: UDMP estimates the sending rate by monitoring the average delay across paths and updating it approximately once per RTT. It uses an exponentially weighted moving average (EWMA) to track the average RTT and leverages its gradient as a congestion signal to adjust the rate, ensuring throughput aligns with the available network bandwidth. Additionally, UDMP maintains a *Token Pool*, which is replenished with tokens (representing the number of packets allowed to send) based on the estimated rate and elapsed time. Packets can only be sent when tokens are available, ensuring throughput matches the estimated rate. Next, we will explain the key components of the control loop.

Algorithm 1 UDMP CC Control Loop

1: $is\_first\_RTT \leftarrow true$
2: $token\_pool \leftarrow BDP/pkt\_size$
3: $bytes\_pool \leftarrow 0$
4: $pkts\_ignored \leftarrow 0$
5: $pkts\_to\_ignored \leftarrow 0$
6: **Procedure** ON_SENDING_PACKET$packet, pathid$
7: **if** $token\_pool \leq 0$ **then**
8: **return**
9: **end if**
10: **if** $is\_first\_RTT = true$ **then**
11: $pathid \leftarrow rand()\%num\_paths$
12: **end if**
13: $token\_pool \leftarrow token\_pool - 1$
14: $packet.path\_id \leftarrow pathid$
15: **Procedure** ON_RECEIVING_ACKack
16: $is\_first\_RTT \leftarrow false$
17: $pkts\_ignored \leftarrow pkts\_ignored + 1$
18: $smooth\_rtt \leftarrow \alpha \times ack.rtt + (1 - \alpha) \times smooth\_rtt$
19: **ALB_ON_RECV**(ack)
20: $bytes\_pool \leftarrow bytes\_pool + rate \times (now - last\_addtoken\_time)$
21: $last\_addtoken\_time \leftarrow now$
22: **if** $smooth\_rtt > T_{high}$ **then**
23: $rate \leftarrow \min\left(rate, linerate \times \left(1 - \frac{smooth\_rtt - base\_rtt}{smooth\_rtt}\right)\right)$
24: **end if**
25: **if** $pkts\_ignored > pkts\_to\_ignore$ **then**
26: $rtt\_diff \leftarrow smooth\_rtt - last\_smooth\_rtt$
27: $gradient \leftarrow rtt\_diff/(now - last\_update\_time)$
28: $last\_smooth\_rtt \leftarrow smooth\_rtt$
29: $last\_update\_time \leftarrow now$
30: **if** $gradient > 0$ **then**
31: $rate \leftarrow rate - |gradient| \times linerate \times \frac{rate}{linerate}$
32: **else**
33: $rate \leftarrow rate + |gradient| \times linerate \times \left(1 - \frac{rate}{linerate}\right)$
34: **end if**
35: $pkts\_ignored \leftarrow 0$
36: $pkts\_to\_ignore \leftarrow inflight\_pkts$
37: **end if**

Initialization Phase: The First RTT. In the first RTT, there are no delay measurements available. As a result, $token\_pool$ is initialized to the Bandwidth-Delay Product (BDP) divided by the packet size (line 2). During this phase, packets are distributed randomly across all paths (lines 11–12).

Delay Gradient Calculation. UDMP relies on NIC timestamps to accurately measure the RTT of each ACK returned from different paths and uses EWMA to update the average RTT (line 20). Then, the gradient of the averaged RTT can be calculated as the difference between the current EWMA-smoothed RTT and the previous smoothed RTT, divided by the elapsed time (lines 28–29). To accurately reflect the congestion level across multiple paths, UDMP calculates the gradient of average RTT samples at intervals of

approximately one RTT (lines 27–38). This interval is determined by observing whether the number of received ACKs equals the inflight packets (line 38).

Sending Rate Calculation. Next, UDMP updates the sending rate based on the gradient of the average RTT. If the gradient of the average RTT is negative or zero, it indicates unused network bandwidth, allowing the endpoint to increase the sending rate. To ensure fairness among flows, we follow the principle that flows with higher sending rates should receive a smaller portion of the remaining bandwidth, leaving more bandwidth for flows with lower sending rates. Accordingly, the remaining bandwidth is allocated proportionally based on the ratio of the current sending $rate$ to the link bandwidth $linerate$, *i.e.*, $1 - rate/linerate$, as described in line 35 of Algorithm 1. Conversely, if the gradient is positive, indicating congestion, UDMP decreases the sending rate using the same $rate$ to $linerate$ ratio, *i.e.*, $rate/linerate$ as shown in line 33 of Algorithm 1. As demonstrated in Appendix A, UDMP's control law ensures fairness between flows.

Detecting Severe Congestion. The gradient-based algorithm effectively controls throughput to approach the total available network bandwidth during the queue buildup phase. However, when the queue stabilizes at a high fixed level, the gradient may remain near zero or exhibit minimal variation. For instance, in scenarios like heavy incast, where the queue persists at a high level, the risk of packet loss increases. We introduce an RTT threshold, T_{high}, to address this issue as an upper limit for acceptable queue levels. Unlike gradient calculations, the T_{high} condition is evaluated for every ACK (line 24) rather than once per RTT, ensuring a quick response to congestion scenarios like incast. By setting T_{high} sufficiently high, we can confidently identify severe congestion and take corrective action to reduce the sending rate (line 25).

Bytes Pool and Token Pool Update. The bytes pool is updated at line 22 of Algorithm 1. Essentially, $bytes\_pool$ is incremented by the product of the estimated sending rate and the time gap since tokens were last added. This ensures that the bytes pool accurately reflects the allowable data transmission rate. $token\_pool$ is updated in Algorithm 2 by converting bytes from $bytes\_pool$ into packets in the $token\_pool$. Finally, $token\_pool$ is decremented by one for each data packet that is sent into the network (line 14).

Algorithm 2 UDMP Adaptive LB Pseudocode

```
1:  Procedure ALB_ON_RECV(ack)
2:  if ack.rtt < smooth_rtt and bytes_pool ≥ 2 × pkt_size then
3:      bytes_pool ← bytes_pool − 2 × pkt_size
4:      token_pool ← token_pool + 2
5:      ON_SENDING_PACKET(new_packet, ack.path_id)
6:      ON_SENDING_PACKET(new_packet, ack.path_id)
7:  else if bytes_pool ≥ pkt_size and ack.rtt < smooth_rtt + base_rtt then
8:      bytes_pool ← bytes_pool − pkt_size
9:      token_pool ← token_pool + 1
10:     ON_SENDING_PACKET(new_packet, ack.path_id)
11: end if
```

C. Delay-Assisted Adaptive Load Balancing

To achieve traffic load balancing across paths without maintaining per-path states, we design a delay-assisted packet scheduling mechanism. This mechanism leverages the ACK clocking principle, where ACK packets returning to the sender trigger the transmission of new packets. ACK packets from less congested paths typically return faster due to lower end-to-end delay. As such, the return speed of an ACK packet inherently reflects the congestion level of the path it traversed. Leveraging this property, we employ the connection-level token pool, which is managed by the congestion control algorithm (introduced in Section III-B), to coordinate packet scheduling across all paths.

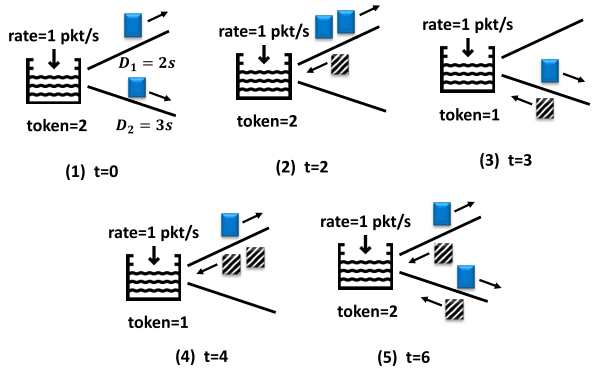


Fig. 3. Traffic distribution across two paths with the coordination of the token pool.

When the token pool has available tokens, an incoming ACK can consume one token and clock out a certain number of new data packets (one or two, depending on the ACK's RTT) onto the same path. This design ensures that faster-returning ACKs, which correspond to less congested paths, have a greater chance of acquiring tokens before the pool is exhausted. Consequently, more packets are injected into less congested paths, naturally shifting the load from overloaded to underutilized paths and achieving load balancing.

The detailed algorithm is presented in Algorithm 2. Specifically: (a) When the incoming ACK's RTT is smaller than the average RTT (calculated across multiple paths), the ACK can clock out up to two packets, provided the token pool has sufficient tokens; (b) If the ACK's RTT is between the average RTT and the sum of the average RTT and the base RTT, the ACK is allowed to trigger at most one packet; (c) If the ACK's RTT exceeds the sum of the average RTT and the base RTT, no new packets are sent to this path. This approach ensures that paths with lower RTTs receive more traffic while overloaded paths with higher RTTs are gradually relieved, promoting balanced load distribution. It is worth noting that, similar to MPRDMA, UDMP requires per-packet ACKs. As reported by MPRDMA, this approach introduces minimal bandwidth overhead (less than 4%) compared to conventional RDMA protocols.

Figure 3 illustrates an example of the token pool's operation, where the token injection rate is one packet per second, and there are two paths with RTTs of 2 seconds (path 1) and 3 seconds (path 2), respectively. At second 0, packets are sprayed randomly across the two paths. At second 2, the incoming ACK from path 1 arrives and triggers the transmission of two data packets due to its shorter RTT. At second 3, the incoming ACK from path 2 triggers the transmission of one data packet. At second 4, the token pool has only one token remaining, so the ACK from path 1 clocks out one data packet. At second 6, the returning ACKs from both paths clock out one packet each,

and the process repeats. Statistically, the traffic distribution ratio between the two paths is $L_1/L_2 = (1+2+1)/(1+1) = 4/2 \approx D_2/D_1 = 3/2$, where L_1 and L_2 represent the traffic on paths 1 and 2, and D_1 and D_2 are their respective RTTs. As this example illustrates, the delay-assisted adaptive load balancing scheme naturally shifts traffic to less congested paths.

D. Other Design Details

Handling Out-of-Order (OOO) Packets. Multipath transmission introduces the packet reordering issue at the receiver side. (1) Storing OOO packets in hardware for reordering requires high on-chip memory, which is costly in RDMA NICs. To overcome this, we adopt a strategy similar to SRD [45], placing received packet data directly into host memory and delegating the handling of out-of-order packets to the MPI layer. The original RDMA header already includes the address in each packet for WRITE and READ operations, so the receiver can place the data accordingly. However, for SEND/RECV operations, additional information is needed to determine the memory address where the data should be placed. Many prior studies have already implemented SEND/RECV schemes without buffering out-of-order packets in NIC's on-chip memory, *e.g.*, MPRDMA [11] and SRNic [17] proposed to carry the target address in the header in different ways, allowing the packet to be placed directly into the application buffer without the need for caching or reordering on the receiver NIC. The extension adds 8 and 20 bytes to the header for SEND and WRITE operations, resulting in slight throughput reductions of 0.7% and 1.8%, respectively.

(2) To track OOO packets and ensure data integrity, UDMP employs a small on-chip bitmap, typically sized to one BDP, with minimal memory overhead. However, delay variations among different paths of a connection can cause severe packet reordering, consuming excessive bitmap space and potentially leading to packet drops. To mitigate this, UDMP adopts an OOO-aware path selection strategy similar to that of MPRDMA. In multipath RDMA transports, packet reordering primarily results from latency imbalances across network paths. MPRDMA addresses this issue using ECN-based congestion control combined with an out-of-order path selection that prunes slower paths and favors those with similar delays. Specifically, when an ACK arrives, the sender checks whether its SACK PSN is lower than snd_ool. If so, the sender reduces the congestion window ($cwnd$) by one, and this delayed ACK triggers a packet retransmission on the other paths. Here, $snd\_ool = snd\_ooh - \Delta$, where snd_ooh denotes the highest PSN acknowledged so far, and Δ—a tunable parameter smaller than the bitmap size—determines MPRDMA's tolerance to OOO packets. Similarly, in UDMP, an ACK whose PSN exceeds the bitmap window is prevented from clocking out new packets, effectively mitigating the OOO degree. The key difference between UDMP and MPRDMA lies in how they handle OOO packets. In UDMP, an OOO packet does not directly reduce the congestion window ($cwnd$). Instead, the transmission opportunity lost on the affected path is implicitly redistributed to other active paths, since the overall $cwnd$ remains unchanged. This design makes UDMP more aggressive in exploiting the aggregate bandwidth and allows it to maintain a high connection-level sending rate for a longer period, which is particularly advantageous for bandwidth-intensive AI training workloads. Although such aggressiveness may slightly increase the OOO degree, it remains well controlled by the delay-aware load balancing (for the normal conditions) and OOO-aware path selection (for the abnormal cases). Consequently, the OOO degree stays fully manageable within the BDP-sized bitmap across diverse network scenarios, as elaborated in Section V-B.

Retransmission Design. Due to multipath transmission, we must differentiate between packet loss and out-of-order arrival to avoid unnecessary retransmissions. We adopt a similar retranmission design as MPRDMA, where the receiver maintains a fixed-size bitmap to track the out-of-order status of packets. The size of the bitmap is fixed to the size of one Bandwidth-Delay Product (BDP). For packets with sequence numbers that exceed the bitmap, we simply discard them and send NACKs to the sender to trigger retransmission. Additionally, based on our observations, UDMP's delay-driven adaptive load balancing scheme (§ III-C) ensures traffic distribution across paths is balanced, and it is rare for the degree of out-of-order packets to exceed the size of the bitmap (§ V). This design helps mitigate unnecessary retransmissions while effectively managing packet reordering.

New Path Exploration. Periodic probing of new paths is essential to identify potentially better routes. Specifically, for each RTT, the sender sets a new source port for the packet with a probability of p, instead of transmitting along the path indicated by the ACK. By probing new paths periodically, the system can explore alternative routes and adapt to changes in network conditions, potentially improving overall performance and load balancing.

IV. Implementation

This section introduces our implementation of UDMP in DPDK and analyzes its memory overhead.

A. Prototype

While UDMP is hardware-friendly and can be fully implemented on a programmable NIC, *e.g.*, by leveraging on-chip processing logic in FPGA-based NICs to realize congestion control and load balancing [11], [46], we currently develop a software-based prototype to validate its functionality due to the unavailability of such hardware.[1] We implemented a multipath transport with the send and receive APIs in Linux 5.4.0 using DPDK 22.11 to verify the correctness and effectiveness of UDMP. The core logic of UDMP, including load balancing and congestion control, is implemented in approximately 200 lines of C code, demonstrating its lightweight design. Multipath packet transmission is achieved by rewriting the destination port field in the UDP header. Since the NICs in our testbed do not support accurate hardware timestamps, we used the end-host CPU's Time Stamp Counter (TSC) to record packet transmission and reception times. This approach enables precise RTT measurement for returning ACKs while minimizing the impact of non-congestion-related delay variations.

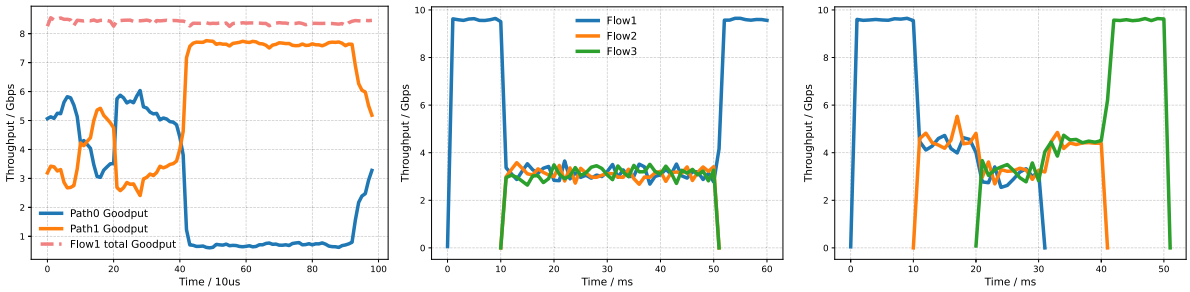
B. Memory Overhead Analysis

Table I summarizes the per-connection states required by UDMP. Similar to MPRDMA, UDMP's carefully designed congestion control and load balancing mechanisms for multipath transmission introduce minimal on-chip memory

[1]We leave the development of a hardware-based prototype for future work.

TABLE I
CONNECTION-LEVEL MEMORY OVERHEAD OF UDMP

UDMP	Variable	Size (B)	*MPRDMA*	Variable	Size (B)
Congestion Control & Load Balance	Avg delay	8	Congestion Control	cwnd & inflate	8
	Last time & delay	8	OOO-aware path selection	snd_ooh & Δ	5
	IgnorePkts & TolgnorePkts	2		L	1
	TokenPool	4	RTT measurement	srt, rttvar, rtt seq	12



(a) Avoidable congestion scenario. **(b)** Fast convergence evaluation of UDMP. **(c)** Fairness evaluation of UDMP.

Fig. 4. Flow throughput time series in avoidable and unavoidable (*i.e.*, incast) congestion scenarios using UDMP.

overhead, which is maintained at the connection level and does not increase with the number of network paths used by the transport. It is worth noting that Table I only shows the overhead associated with UDMP's congestion control and load balancing functionalities, while the overhead for retransmissions and other components remains consistent between UDMP and MPRDMA.

V. EVALUATION

In this section, we present the evaluation results of UDMP. Our study includes both testbed experiments and large-scale simulations, comparing UDMP's performance across various topologies and workload scenarios with state-of-the-art multi-path RDMA transport solutions.

A. Testbed Experiments

Setup. Our testbed consists four servers. Each server is equipped with an Intel Xeon E5-2650 CPU @ 2.20GHz, 256GB RAM. Servers are interconnected via a Tofino switch and Mellanox ConnectX-5 NICs on the servers. The link is configured at 10Gbps, with the switch utilizing the default shared buffer mechanism, and the base RTT is 14μs. Currently, with P4 programming, we can configure the Tofino switch into a two-tier spine-leaf topology.

Avoidable congetsion scenario. To evaluate the effectiveness of UDMP in avoiding congestion, we assess how UDMP handles avoidable congestion by dynamically shifting traffic from congested paths to non-congested ones without compromising flow throughput. Specifically, we measure the real-time throughput of the affected flow across different paths. We replicate the elephant flow collision scenario shown in Figure 1, initiating two flows at 0μs and 400μs, respectively, and display the total throughput of flow 1, along with its real-time throughput across both paths. As illustrated in Figure 4a, flow 2 starts at 400μs and collides with flow 1 on path 0 (*i.e.*, L1 to S1 in Figure 1), causing flow 1 to quickly reduce its throughput on path 0 and shift it to path 1 (*i.e.*, L1 to S2 in Figure 1), all while maintaining stable overall throughput. The experiment confirms that UDMP can effectively identify avoidable congestion scenarios and promptly load balances traffic from congested paths to non-congested ones without sacrificing flow throughput.

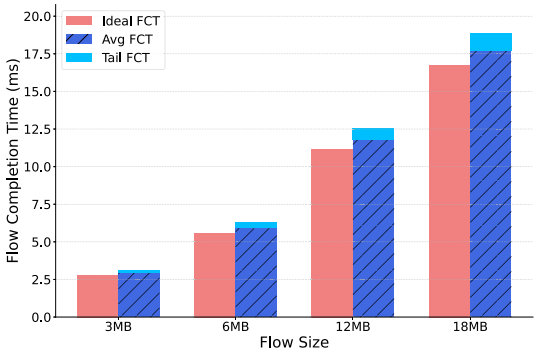
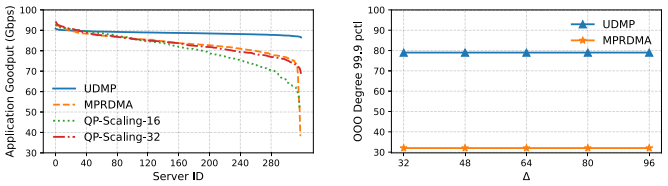


Fig. 5. The FCT of UDMP under the AllReduce workload with varying flow sizes.

Unavoidable congestion scenario. We adjust the network topology to create an incast test with an incast degree of 3, evaluating UDMP's congestion control mechanism under unavoidable congestion scenarios. Initially, a sender starts a long-lived flow at time 0ms, followed by two additional senders transmitting data to the same receiver at time 10ms for 40ms. As shown in Figure 4b, UDMP quickly converges to a fair share of bandwidth when incast flows compete on the same bottleneck link. Next, we modify the traffic pattern: three senders initiate short flows lasting 30ms each at time 0ms, 10ms, and 20ms, respectively, to the same receiver. The results in Figure 4c demonstrate that UDMP ensures a fair allocation of network bandwidth for all flows.

AllReduce workload. In the same testbed, we use UDMP to run AllReduce traffic, a representative communication pattern in AI training, to evaluate its behavior under realistic traffic dynamics. Specifically, four servers form a ring-based communication group and execute three rounds of AllReduce operations. We vary the flow sizes and measure both the average and tail FCTs. As shown in Figure 5, the average and tail FCTs achieved by UDMP are very close to the ideal



(a) Flow throughput statistics. (b) Out-of-Order Degree.

Fig. 6. Comparing UDMP with existing multipath RDMA transports using the permutation workload.

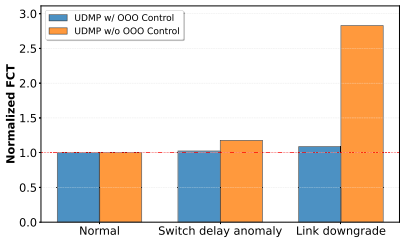


Fig. 7. Out-of-order control under different network failures with UDMP.

baseline assuming full link utilization (without computation overhead), and the average and tail performance deviation remains within 5% and 10%, respectively. This result demonstrates that our software prototype can effectively handle realistic traffic dynamics.

B. Large-Scale Simulations

Setup. In addition to DPDK-based testbed experiments, we implemented UDMP in the NS3 simulator and conducted large-scale simulations to evaluate its performance. For the simulations, we adopted a fat-tree topology with 320 hosts. The topology includes five pods interconnected by 16 core switches, with each pod containing 4 aggregation switches, 4 edge switches, and 16 servers. Host-to-edge switch links are configured at 100 Gbps, while inter-switch links operate at 400 Gbps to create a non-blocking topology. Switch buffers are set to 32 MB, aligning with buffer sizes commonly found in modern Ethernet switching chips. The propagation delay of each link is set to 1μs.

We compare the performance of UDMP with state-of-the-art multipath RDMA transport solutions, including MPRDMA and QP-Scaling. Specifically, we evaluate QP-Scaling-16 and QP-Scaling-32, where traffic between the source and destination servers is split across 16 and 32 QPs, respectively. Average and tail flow completion times are used as the primary metrics for performance comparison.

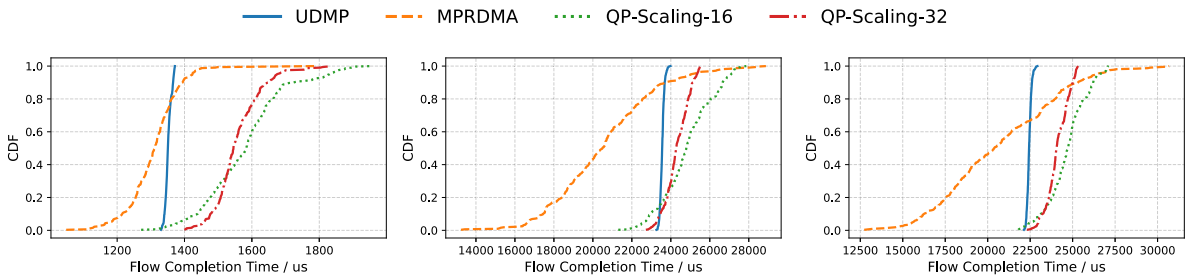
Permutation workloads in non-blocking networks. We employ the permutation traffic pattern to evaluate UDMP's effectiveness in traffic load balancing and congestion control. In the permutation traffic pattern, each sender in the network sends data to one unique receiver, and every receiver receives data from exactly one unique sender, forming a one-to-one mapping between senders and receivers. This pattern ensures that all nodes are engaged in communication simultaneously, with no overlap in their pairings. In an ideal scenario with perfect or near-perfect load balancing, the congestion control mechanism should not throttle the sending rate of any flow. As such, this workload is ideal for demonstrating the efficacy of the load balancing mechanism and the interaction between congestion control and load balancing.

Figure 6a illustrates the throughput of permutation flows under various multipath RDMA transport solutions. UDMP achieves near-optimal performance, with each flow's throughput approaching the theoretical maximum. This indicates that UDMP's congestion control and adaptive load balancing work in tandem to distribute traffic uniformly across network paths, effectively managing avoidable congestion and minimizing its impact on throughput. In terms of tail throughput at the 99th percentile (P99), UDMP outperforms other solutions, achieving improvements of 2x, 1.7x, and 1.3x over MPRDMA, QP-Scaling-16, and QP-Scaling-32, respectively. These results validate UDMP's design and highlight its advantages over state-of-the-art solutions.

Then, following the convention in MPRDMA, we evaluate the packet Out-of-Order Degree (OOD) resulting from multipath transmission. OOD is defined as the difference between two packet sequence numbers (PSNs). Each packet header includes a unique PSN, and the receiver tracks these sequence numbers from incoming packets to calculate the OOD. In our evaluation, the bitmap size is set to infinite to accommodate the maximum OOD. All paths are configured with RED marking parameters (P_{max}, K_{min}, K_{max}) = (1.0, 36 packets, 36 packets), and the BDP size is approximately 108 packets. As outlined in Algorithm 2, UDMP evaluates path congestion levels based on latency, and ACKs returned from paths whose RTTs exceed the sum of a base RTT and the smoothed RTT are restricted from clocking out new packets on those paths. Figure 6b depicts the 99.9th percentile of OOD for various Δ values. UDMP demonstrates a higher OOD compared to MPRDMA due to its less stringent criteria for identifying slower paths. However, the OOD in UDMP remains within the BDP size, enabling packet tracking and allowing for fast retransmissions with a fixed bitmap overhead.

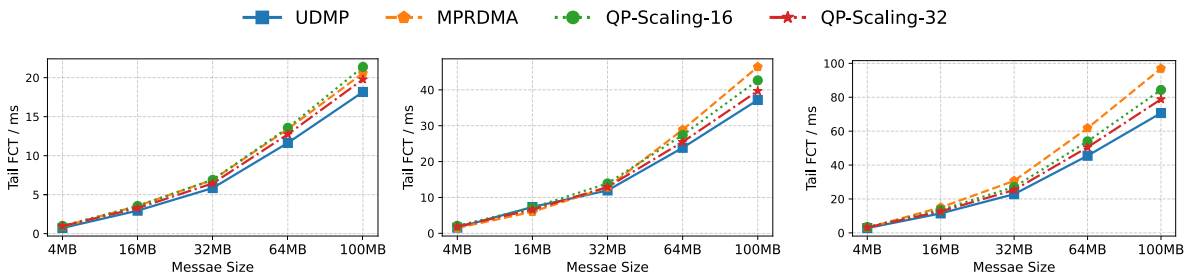
Moreover, to test the OOO control capability of UDMP under different network failures, we set the OOO control threshold Δ of the OOO-aware path selection to 64 and create two network failure scenarios: (1) *Switch delay anomaly*, where the switch processing rate drops and causes the RTT in one link to spike dramatically, increasing from 1μs to 1ms; (2) *Link degradation*, where the capacity of certain paths is instantly reduced from 100 Gbps to 40 Gbps to emulate link downgrades caused by auto-negotiation failures. As shown in Figure 7, under normal conditions, the integrated load balancing and congestion control of UDMP effectively bound the OOD, resulting in the same normalized FCT for UDMP with and without the OOO-aware path selection dedicated for OOO control. In the switch delay anomaly scenario, enabling OOO-aware path selection increases the FCT by only about 3%, while disabling it leads to a 20% increase, demonstrating that it effectively mitigates performance degradation even when the load balancing function is impaired. In the severe link degradation scenario, UDMP with OOO-aware path selection experiences less than 10% increase in FCT, whereas without the dedicated OOO control, the FCT increases by up to $2\times$ compared with the normal condition. These results confirm that the OOD in UDMP remains well bounded under adverse network conditions, ensuring low retransmission rates and stable throughput in real-world environments.

Permutation workloads in over-subscribed networks. Network over-subscription can occur in data center networks



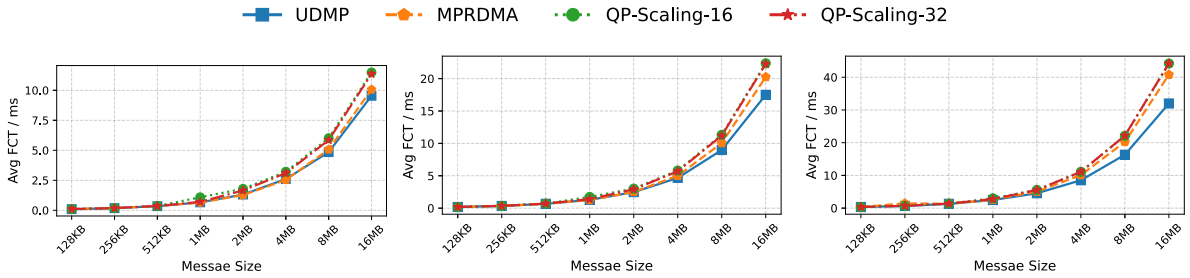
(a) Permutation workload 2:1 oversub - 8 MB 320 Nodes - 100Gbps. **(b)** Permutation workload 4:1 oversub - 32 MB 320 Nodes - 100Gbps. **(c)** Permutation workload 8:1 oversub - 64 MB 320 Nodes - 100Gbps.

Fig. 8. Flow completion time for different permutation workloads on fat-tree topologies with different oversubscription ratios.



(a) Permutation workload 2:1 oversub - 320 Nodes - 100Gbps. **(b)** Permutation workload 4:1 oversub - 320 Nodes - 100Gbps. **(c)** Permutation workload 8:1 oversub - 320 Nodes - 100Gbps.

Fig. 9. Tail (max) flow completion time for different message sizes on fat-tree topologies with different oversubscription ratios.



(a) 8:1 incast, average message completion time. **(b)** 16:1 incast, average message completion time. **(c)** 32:1 incast, average message completion time.

Fig. 10. Incast performance for varying message sizes and incast degrees.

as a cost-saving measure, such as by reducing the number of switches and links. In this experiment, we modify the full-bisection-bandwidth, non-blocking 3-tier fat-tree topology to create 2:1, 4:1, and 8:1 oversubscription scenarios. In these configurations, the total bandwidth from the aggregation switches to the core switches is reduced to 1/2, 1/4, and 1/8 of the total bandwidth from the hosts to the ToR switches. We use the permutation pattern to generate traffic workloads, setting the permutation flow message size to 8 MB, 32 MB, and 64 MB for the 2:1, 4:1, and 8:1 oversubscription ratios, respectively. This experiment evaluates the effectiveness of the joint optimization of congestion control and multipath load balancing mechanisms in UDMP. Figure 8 shows the CDF of flow completion time under various fabric oversubscription ratios. We observe that, compared to existing solutions, UDMP achieves more uniform flow completion times. Figure 9 presents the tail (max) flow completion times for varying flow message sizes. The tail flow completion time in UDMP is reduced by 15% to 28% compared to MPRDMA and QP-Scaling.

Incast workloads. We also conduct incast experiments, evaluating 8-to-1, 16-to-1, and 32-to-1 incast scenarios with 8 different flow message sizes, ranging from 4MB to 100MB. Figure 10 shows the average flow completion time for UDMP, MPRDMA, and QP-Scaling across these incast scenarios, with varying message sizes. The average flow completion time of UDMP is reduced by 18% to 30% compared to state-of-the-art methods. This improvement is attributed to UDMP's delay gradient-based congestion control mechanism, which is more sensitive to delay variations and offers a more precise, adaptive method for adjusting flow transmission rates in response to congestion levels.

AI training workloads. Finally, we evaluate the performance of UDMP and other multipath RDMA transports using AlltoAll and AllReduce communication patterns, which are prevalent in distributed AI training workloads.

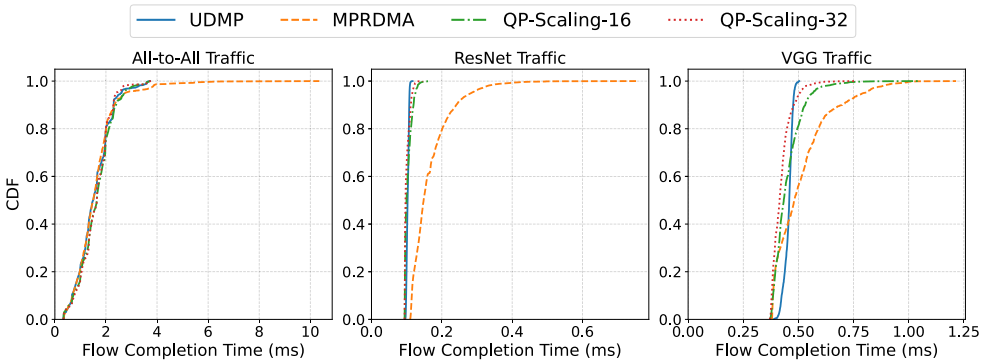


Fig. 11. The CDF of FCTs for different schemes under various AI training workloads.

The experiments simulate model training scenarios on a fat-tree topology with 320 hosts, partitioned into 20 groups performing identical collective operations. This setup mimics scenarios where multiple training jobs are running concurrently in the data center fabric. Figure 11 presents the CDF of FCTs for different schemes under various AI training workloads. Both ResNet and VGG use AllReduce communications, but they differ in the computation times, resulting in varying traffic volumes for the two models. Specifically, the time interval between consecutive flows from a node differs in these two models. Due to the synchronization nature of distributed AI training, the tail flow completion time (FCT) within each group determines the overall job completion time. For instance, in a job consisting of 100 AllReduce flows per iteration, the next iteration cannot start until all 100 flows have completed. Hence, even a single slow flow delays the entire iteration, and such slowdowns can accumulate over multiple iterations, significantly extending the total training time. As a result, AI training workloads are particularly sensitive to tail flow completion times. Our experiments show that, UDMP achieves the relatively shortest tail FCTs across all three collective communication patterns, demonstrating its effectiveness in reducing training time. Specifically, for AllToAll communication, UDMP reduces tail FCTs by 28% compared with MPRDMA. Because AllToAll traffic, characterized by a large number of flows, efficiently utilizes all available paths, leading to fewer hotspots and comparable FCTs between UDMP and QP-Scaling. In contrast, AllReduce traffic employs fewer flows and experiences more severe hotspot contention. For ResNet and VGG workloads, UDMP reduces tail FCTs by 70% and 47% compared to MPRDMA, and by 20% and 28% compared to QP-Scaling-32, respectively. These results demonstrate UDMP's effectiveness in mitigating congestion and enhancing training efficiency.

VI. Discussion

Per-Packet ACKs Overhead. Per-packet ACKs are indeed fundamental to UDMP, as they enable accurate RTT sampling and thus precise delay-based control. This cost is inherent to any RTT-driven congestion control mechanism. The bandwidth overhead introduced by ACK packets (less than 4% in our evaluation) is comparable to that of existing RTT-based schemes, such as Timely [42], Swift [41], and DX [47], which also rely on frequent RTT measurements. Therefore, the overhead is not unique to UDMP but reflects a common design requirement for achieving fine-grained delay feedback.

Delay Noise. As clarified in Section IV-A, our current implementation is a software prototype, whereas the intended deployment of UDMP is on hardware (*e.g.*, SmartNICs), which is our future work. Prior studies, including DX [47], have shown that hardware timestamping provides high-fidelity and low-jitter RTT measurements, effectively eliminating the noise introduced by shared CPU resources. Therefore, in the envisioned hardware deployment, UDMP's RTT-based decisions would be significantly more stable and robust.

Potential for Increased Out-of-Order Packets. Although UDMP may introduce a higher degree of out-of-order packets, our design already incorporates dedicated handling mechanisms to keep reordering within a manageable range (elaborated in Section III-D). In particular, the OOO-aware path selection described in Section III-D restricts slower paths from clocking out new packets and thus prevents excessive reordering. Finally, as shown in Section V-B, these mechanisms effectively maintain the OOO degree at an acceptable level.

New Path Exploration Overhead. As noted in Section III-D, periodic path exploration is a common design choice in multipath transport protocols, *e.g.*, MPRDMA [11], and is necessary for discovering better routes under dynamic network conditions. Although changing source ports may momentarily perturb ECMP hashing and cause short-lived imbalance, the impact is minimal because probing happens infrequently (*e.g.*, every few seconds) and thus occupies only a negligible fraction of typical flow lifetimes in datacenter environments, whose mean FCTs are on the order of several to tens of milliseconds.

Bitmap Overhead. UDMP is designed for AI training networks within a datacenter environment, whose RTT is usually below one millisecond. In such settings, although higher link bandwidth increases the BDP (usually tens to hundreds of KiBs) and thus the bitmap size (tens to hundreds of bits per flow), the corresponding NIC hardware resources (*e.g.*, on-chip SRAM capacity) also scale across generations. For example, the number of concurrent connections supported by NVIDIA RNICs has increased from hundreds in ConnectX-3 to thousands in ConnectX-6, reflecting the steady growth of available on-chip memory. This trend suggests that maintaining a BDP-sized bitmap remains feasible even as bandwidth increases.

Multi-Tenant or Heterogeneous Workloads Evaluation. UDMP is designed specifically for AI training clusters, where traffic is dominated by synchronized, bursty elephant flows [48], typically with sizes on the order of tens of megabytes or more. These flow characteristics align naturally with UDMP's aggressive load-balancing and delay-driven congestion control mechanisms, which are optimized for high-bandwidth,

long-lived transfers rather than short RPC-style workloads. While mixed traffic scenarios are less common in dedicated AI training fabrics, exploring such heterogeneous environments is an interesting direction for future work and may require additional design considerations.

VII. Related Work

Traffic Load Balancing. ECMP is the de facto load balancing scheme used in many production data centers due to its simplicity and scalability—it hashes flows onto network paths with equal weights. WCMP [31] extends ECMP by assigning different weights to paths. Flow-level load balancing schemes such as ECMP and WCMP suffer from issues like elephant flow collision [7], [8], [9], [10], [49] and hash polarization [32]. CONGA [8] and LetFlow [9] advocate for load balancing at the flowlet granularity in switches, where flowlets are identified based on inactivity gaps in a flow. Presto [10] proposes load balancing on 64KB flowcells and addresses packet reordering issues in the Linux GRO layer. DRILL [49] uses the queue lengths of local switch ports and randomized algorithms to perform packet-level load balancing in switches in a distributed manner. Host-based PLB [50] randomly changes the paths of flows experiencing congestion and repaths after idle periods to minimize packet reordering.

Datacenter Congestion Control. DCTCP [43] is a data center transport protocol that leverages Explicit Congestion Notification (ECN) to achieve low latency and high throughput by dynamically adjusting the sending rate based on network congestion levels. DCQCN [14] combines ECN and Priority Flow Control (PFC) to achieve fairness, high throughput, and low latency for RDMA over Converged Ethernet (RoCE). TIMELY [42] proposes to use RTT as the congestion signal to achieve low latency and high throughput in data center networks without relying on ECN, while Swift [41] extends this approach by decoupling fabric and host congestion and using a simple target end-to-end delay instead of RTT gradients to control flow sending rate.

Multi-path Transport. MPTCP [35], [51] enhances TCP by enabling a single connection to use multiple paths concurrently, thereby increasing reliability and throughput while efficiently utilizing available network resources. MPRDMA [11] introduces the first multi-path RDMA transport protocol, aimed at boosting throughput, reducing latency, and enhancing robustness against network failures. STrack [12] employs ECN for adaptive packet-level load balancing and leverages RTT as the signal for congestion control. SRD [45] introduces a novel (yet proprietary) transport mechanism for HPC/ML clusters, combining packet-level traffic load balancing with tailored congestion control and loss recovery schemes to manage congestion and ensure reliable data delivery. Like UDMP, it delegates the handling of out-of-order packets to the MPI layer. Lastly, NDP [52] re-architects data center transport by employing packet-level ECMP in symmetric Clos networks, incorporating payload cutting [53], and adopting a receiver-driven congestion control scheme to reduce latency in data center networks.

VIII. Conclusion

Distributed AI training workloads present significant challenges to traditional single-path RDMA transports. Existing multipath RDMA transport protocols, such as QP-Scaling and MPRDMA, face scalability and performance limitations, hindering their ability to meet the demands of large-scale AI clusters. To address these issues, this paper introduces Unified Delay-driven Multipath Protocol (UDMP), a novel approach designed to overcome challenges in scalability, congestion control, and load balancing. Using delay as a unified signal, UDMP integrates two key components: delay-gradient-based congestion control and delay-assisted load balancing. This co-design ensures high throughput, low latency, and robustness against congestion and failures. Furthermore, UDMP features a novel Token Pool design that seamlessly bridges congestion control and load balancing while eliminating per-path state overhead, significantly enhancing scalability for distributed AI training at scale. Experimental results demonstrate UDMP's superior performance, achieving up to 2x better throughput and reducing flow completion times up to 30% compared to state-of-the-art multipath RDMA transport solutions.

Appendix
Fairness Analysis

A. UDMP CC Fairness Proof

The AIMD algorithm benefits from the fact that all flows either increase rate with the same amount or decrease rate with the same ratio. And Poseidon [54] has shown that the MIMD (Multiplicative Increase Multiplicative Decrease) algorithm ensures fairness by satisfying the inequality (5). For instance, given two flows A and B with rates a and b, satisfying $a < b$, MIMD ensures:

$$\frac{a}{b} < \frac{b \cdot U}{a \cdot U} < \frac{b}{a}. \tag{4}$$

where U is the update coefficient. Specifically, $b \cdot U$ and $a \cdot U$ represent the updated flow rates after applying the adjustments of the CC algorithm.

Our method builds upon this principle and can be expressed as follows:

- MD (Multiplicative Decrease):

$$b \cdot U = b(1 - g), \tag{5}$$

- MI (Multiplicative Increase):

$$b \cdot U = b + g(l - b), \tag{6}$$

where g represents the delay gradient, and l is the link's speed. Obviously, the MD phase keeps the rate ratio constant.

To validate the MI approach converges to fairness, we start by analyzing the inequality:

$$\frac{a}{b} < \frac{b \cdot U}{a \cdot U} \implies \frac{a}{b} - \frac{b \cdot U}{a \cdot U} < 0. \tag{7}$$

Breaking it down, the calculation is as follows:

$$\frac{a}{b} - \frac{b \cdot U}{a \cdot U} = \frac{a}{b} - \frac{b + g \cdot (l - b)}{a + g \cdot (l - a)}. \tag{8}$$

Simplifying further:

$$\frac{a}{b} - \frac{b \cdot U}{a \cdot U} = \frac{g \cdot l \cdot (a - b) + (a^2 - b^2)(1 - g)}{gb(l - a) + ab}. \tag{9}$$

Since $a < b$, the numerator $g(a - b) + (a^2 - b^2)(1 - g)$ is negative, proving that:

$$\frac{a}{b} - \frac{b \cdot U}{a \cdot U} < 0. \tag{10}$$

Similarly, the inequality to be proven is:

$$\frac{b}{a} > \frac{b \cdot U}{a \cdot U} \implies \frac{b}{a} - \frac{b \cdot U}{a \cdot U} > 0. \tag{11}$$

Breaking it down, the calculation is:

$$\frac{b}{a} - \frac{b \cdot U}{a \cdot U} = \frac{b}{a} - \frac{b + g \cdot (l - b)}{a + g \cdot (l - a)}. \tag{12}$$

Simplifying further:

$$\frac{b}{a} - \frac{b \cdot U}{a \cdot U} = \frac{g \cdot l \cdot (b - a)}{a^2 + g \cdot (l - a) \cdot a}. \tag{13}$$

Since $b > a$, the numerator $g(b - a)$ is positive, proving that:

$$\frac{b}{a} - \frac{b \cdot U}{a \cdot U} > 0. \tag{14}$$

Therefore, the MI method will gradually converge to a fair state, while the MD does not increase or decrease fairness between flows. In the end, the whole approach is fair. In conclusion, both the MD and MI approaches satisfy the fairness constraints, ensuring a fair allocation of network bandwidth.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Oct. 2008.

[2] K. Qian et al., "Alibaba HPN: A data center network for large language model training," in *Proc. ACM SIGCOMM Conf.* New York, NY, USA: Association for Computing Machinery, Aug. 2024, pp. 691–706, doi: 10.1145/3651890.3672265.

[3] A. Greenberg et al., "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 51–62.

[4] A. Singh et al., "Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 183–197, 2015.

[5] A. Gangidi et al., "RDMA over Ethernet for distributed training at meta scale," in *Proc. ACM SIGCOMM Conf.*, Aug. 2024, pp. 57–70.

[6] InfiniBand Trade Association. (2014). *Supplement to InfiniBand TM Architecture Specification Volume 1 Release 1.2.1 Annex A17: RoCEv2*. [Online]. Available: https://web.archive.org/web/20200917012109/https://cw.infinibandta.org/document/dl/7781

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.*, 2010, pp. 89–92.

[8] M. Alizadeh et al., "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 503–514.

[9] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 407–420.

[10] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 465–478, 2015.

[11] Y. Lu et al., "Multi-path transport for RDMA in datacenters," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement.*, 2018, pp. 357–371. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/lu

[12] Y. Le et al., "STrack: A reliable multipath transport for AI/ML clusters," 2024, *arXiv:2407.15266*.

[13] C. Guo et al., "RDMA over commodity Ethernet at scale," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 202–215.

[14] Y. Zhu et al., "Congestion control for large-scale RDMA deployments," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 523–536, Sep. 2015.

[15] W. Bai et al., "Empowering Azure storage with RDMA," in *Proc. 20th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2023, pp. 49–67.

[16] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *Proc. 16th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2019, pp. 1–16.

[17] Z. Wang et al., "SRNIC: A scalable architecture for RDMA NICs," in *Proc. 20th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2023, pp. 1–14.

[18] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. 1st ACM Workshop Res. Enterprise Netw.*, Aug. 2009, pp. 73–82.

[19] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data-center networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 345–358, Apr. 2013.

[20] J. Zhang, F. Ren, and C. Lin, "Modeling and understanding TCP incast in data center networks," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1377–1385.

[21] X. Jia et al., "Turbo: Efficient communication framework for large-scale data processing cluster," in *Proc. ACM SIGCOMM Conf.*, Aug. 2024, pp. 540–553.

[22] F. Tian, Y. Zhang, W. Ye, C. Jin, Z. Wu, and Z.-L. Zhang, "Accelerating distributed deep learning using multi-path RDMA in data center networks," in *Proc. ACM SIGCOMM Symp. SDN Res. (SOSR)*, Oct. 2021, pp. 88–100.

[23] S. Li et al., "PyTorch distributed: Experiences on accelerating data parallel training," 2020, *arXiv:2006.15704*.

[24] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*.

[25] Y. Huang et al., "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 103–112.

[26] D. Narayanan et al., "PipeDream: Generalized pipeline parallelism for DNN training," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, Oct. 2019, pp. 1–15.

[27] U. E. Consortium. (2024). *The New Era Needs a New Network*. Accessed: Dec. 25, 2024. [Online]. Available: https://ultraethernet.org/

[28] Nvidia. (2024). *Nvidia Dgx Superpod*. Accessed: Dec. 25, 2024. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-superpod/

[29] Nvidia. (2024). *Nvlink and Nvlink Switch*. Accessed: Dec. 25, 2024. [Online]. Available: https://www.nvidia.com/en-us/data-center/nvlink/

[30] N. Jouppi et al., "TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, Jun. 2023, pp. 1–14.

[31] J. Zhou et al., "WCMP: Weighted cost multipathing for improved fairness in data centers," in *Proc. 9th Eur. Conf. Comput. Syst.*, Apr. 2014, pp. 1–14.

[32] Y. Xu et al., "Hashing design in modern networks: Challenges and mitigation techniques," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2022, pp. 805–818.

[33] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proc. ACM SIGCOMM Conf.*, Aug. 2011, pp. 350–361.

[34] P. Gervasi. *The Evolution of Data Center Networking for AI Workloads*. Accessed: Dec. 2025. [Online]. Available: https://www.kentik.com/blog/the-evolution-of-data-center-networking-for-ai-workloads

[35] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. 8th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2011, pp. 99–112.

[36] S. K. Monga, S. Kashyap, and C. Min, "Birds of a feather flock together: Scaling RDMA RPCs with flock," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, Oct. 2021, pp. 212–227.

[37] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2016, pp. 437–450.

[38] A. Kalia, M. Kaminsky, and D. G. Andersen, "FaSST: Fast, scalable and simple distributed transactions with two-sided RDMA datagram RPCs," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement.*, 2016, pp. 185–201.

[39] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proc. Internet Meas. Conf.*, Nov. 2017, pp. 78–85.

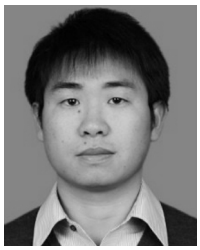
[40] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Observing and mitigating micro-burst traffic in data center networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 98–111, Feb. 2020.

[41] G. Kumar et al., "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 514–528.

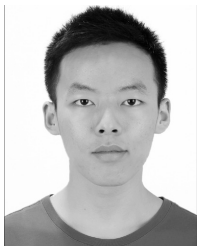
[42] R. Mittal et al., "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, Sep. 2015.
[43] M. Alizadeh et al., "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf.*, Aug. 2010, pp. 63–74.
[44] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 329–342.
[45] L. Shalev, H. Ayoub, N. Bshara, and E. Sabbag, "A cloud-optimized transport protocol for elastic and scalable HPC," *IEEE Micro*, vol. 40, no. 6, pp. 67–73, Nov. 2020.
[46] M. T. Arashloo, A. Lavrov, M. Ghobadi, J. Rexford, D. Walker, and D. Wentzlaff, "Enabling programmable transport protocols in high-speed NICs," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, Feb. 2020, pp. 93–109. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/arashloo
[47] C. Lee, C. Park, K. Jang, S. Moon, and D. Han, "Accurate latency-based congestion feedback for datacenters," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, Jul. 2015, pp. 403–415. [Online]. Available: https://www.usenix.org/conference/atc15/technical-session/presentation/lee-changhyun
[48] W. Li et al., "Understanding communication characteristics of distributed training," in *Proc. 8th Asia–Pacific Workshop Netw.* New York, NY, USA: Association for Computing Machinery, Aug. 2024, pp. 1–8, doi: 10.1145/3663408.3663409.
[49] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro load balancing for low-latency data center networks," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 225–238.
[50] M. A. Qureshi et al., "PLB: Congestion signals are simple and effective for network load balancing," in *Proc. ACM SIGCOMM Conf.*, Aug. 2022, pp. 207–218.
[51] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM Conf.*, Aug. 2011, pp. 266–277, doi: 10.1145/2018436.2018467.
[52] M. Handley et al., "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 29–42.
[53] P. Cheng, F. Ren, R. Shu, and C. Lin, "Catch the whole lot in an action: Rapid precise packet loss notification in data center," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2014, pp. 17–28.
[54] W. Wang et al., "Poseidon: Efficient, robust, and practical datacenter CC via deployable INT," in *Proc. 20th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2023, pp. 255–274.

Chengyuan Huang (Member, IEEE) received the B.Eng. and Ph.D. degrees from Beijing University of Posts and Telecommunications in 2015 and 2021, respectively. From 2021 to 2023, he was a Post-Doctoral Researcher with Purple Mountain Laboratories, Nanjing, China. He is currently an Assistant Researcher with the Department of Computer Science and Technology, Nanjing University. His research interests include data center networks, software-defined networking, and distributed systems.

Zhengqi Cui received the B.S. degree from the School of Computer Science and Technology, Northeastern University, China, in 2024. He is currently pursuing the M.S. degree with the School of Computer Science and Technology, Nanjing University, China. His research interests include networked system for large language model and transport protocol.



Jun Xu received the B.S. degree from Jinan University, Guangzhou, China, in 2006, and the Ph.D. degree from the University of Science and Technology of China (USTC), Hefei, China, in 2012. Since 2014, he has been with China Mobile China Mobile (Suzhou) Software Technology Company Ltd., China, as a Senior Researcher. His research interests include datacenter networks, edge intelligence, and machine learning systems.

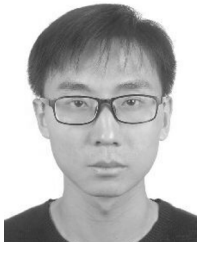


Zhaochen Zhang received the B.S. degree from the Department of Software Engineering, Huazhong University of Science and Technology, China, in 2021. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Nanjing University, China. His research interests include datacenter networks.

Li Wang received the B.S. degree from the Department of Computer Science and Technology, Nanjing University, China, in 2024, where he is currently pursuing the Ph.D. degree. His research interests include heterogeneous computing and programmable networks.

Peirui Cao received the bachelor's degree from Southeast University in June 2015, the master's degree from Beihang University in March 2018, and the Ph.D. degree from Shanghai Jiao Tong University in June 2024. He is currently a Research Assistant Professor with the School of Computer Science, Nanjing University (NJU). He is committed to closing the gap between theory and real network systems. His research interests include data center networks, large-scale network simulators, and network bottleneck analysis.

Zhongming Ji received the B.E. degree in communication engineering from Hefei University of Technology, Hefei, China, in 2017, and the Ph.D. degree in information and communication engineering from the University of Science and Technology of China (USTC), Hefei, in 2022. He is currently a Technical Researcher with China Mobile (Suzhou) Software Technology Company Ltd. His current research interests include datacenter networks, cloud computing, and edge computing.



Jilei Chen received the M.E. degree in computer software and theory from Jiangsu University of Science and Technology, Zhenjiang, China, in 2016. Since 2016, he has been with China Mobile China Mobile (Suzhou) Software Technology Company Ltd. His current research interests include cloud computing, computing power networks, and artificial intelligence.

Dongxu Wang received the B.E. degree in computer science and technology and the M.E. degree in computer application from Harbin Institute of Technology, Harbin, China, in 2005 and 2007, respectively. Since 2015, he has been with China Mobile (Suzhou) Software Technology Company Ltd. His current research interests include cloud computing, computing power networks, and software–hardware collaboration.

Shengju Zhang received the B.E. degree in software engineering from Wuhan University of Technology, Wuhan, China, in 2007, and the M.E. degree in computer software and theory from Nanjing University, Nanjing, China, in 2010. Currently, he is with China Mobile (Suzhou) Software Technology Company Ltd. His current research interests include computing power networks, operating systems, virtualization, and heterogeneous computing.

Lingkun Meng received the B.E. degree in computer science and technology from the University of Jinan, Jinan, China, in 2012, and the M.E. degree in software engineering from Zhejiang University, Hangzhou, China, in 2014. Since 2014, he has been with China Mobile (Suzhou) Software Technology Company Ltd. His current research interests include cloud computing, edge computing, computing power networks, and artificial intelligence.

Ahmed M. Abdelmoniem (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from The Hong Kong University of Science and Technology (HKUST), Hong Kong, in 2017. He is currently an Associate Professor with the School of Electronic Engineering and Computer Science, Queen Mary University of London, U.K. He also leads the SAYED Systems Group. He held the positions of a Research Scientist with KAUST, Saudi Arabia, and a Senior Researcher with Huawei's Future Networks Laboratory (FNTL), Hong Kong. He is a principal investigator and a co-investigator on several national and international research projects, funded mainly by grants more than U.S. $1.7 million. He has published numerous (more than 135) papers in top venues and journals in distributed systems, computer networking, and machine learning. His current research interests are optimizing systems supporting distributed machine learning, federated learning, and cloud/data-center networking, emphasizing performance, practicality, and scalability. He was awarded the prestigious Hong Kong Ph.D. Fellowship from the RGC of Hong Kong in 2013.

Fu Xiao (Senior Member, IEEE) received the Ph.D. degree in computer science and technology from Nanjing University of Science and Technology, Nanjing, China, in 2007. He is currently a Professor and a Ph.D. Supervisor with the School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing. His research papers have been published in many prestigious conferences and journals, such as IEEE INFOCOM, IEEE ICC, IEEE IPCCC, IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON MOBILE COMPUTING, *ACM Transactions on Embedded Computing Systems*, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. His research interests include the Internet of Things and mobile computing. He is a member of the IEEE Computer Society and the Association for Computing Machinery.

Wanchun Dou received the Ph.D. degree in mechanical and electronic engineering from Nanjing University of Science and Technology, China, in 2001. From April 2005 to June 2005 and from November 2008 to February 2009, he visited the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, as a Visiting Scholar. He is currently a Full Professor with the State Key Laboratory for Novel Software Technology, Nanjing University. Up to now, he has chaired three National Natural Science Foundation of China projects and published more than 60 research papers in international journals and international conferences. His research interests include workflow, cloud computing, and service computing.

Guihai Chen (Fellow, IEEE) received the B.S. degree in computer software from Nanjing University in 1984, the M.E. degree in computer applications from Southeast University in 1987, and the Ph.D. degree in computer science from The University of Hong Kong in 1997. He had been invited as a Visiting Professor with the Kyushu Institute of Technology, Japan; The University of Queensland, Australia; and Wayne State University, USA. He is currently a Distinguished Professor with Nanjing University. He has published more than 350 peer-reviewed papers, and more than 200 of them are in well-archived international journals such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, IEEE/ACM TRANSACTIONS ON NETWORKING, and *ACM Transactions on Sensor Networks*, and also in well-known conference proceedings, such as HPCA, MOBIHOC, INFOCOM, ICNP, ICDCS, CoNext, and AAAI. His research interests include parallel computing, wireless networks, data centers, peer-to-peer computing, high-performance computer architecture, and data engineering. He has won nine paper awards, including the ICNP 2015 Best Paper Award and the DASFAA 2017 Best Paper Award.

Keqiang He received the B.E. degree in software engineering from Xidian University in 2009, the M.E. degree in computer science and technology from Tsinghua University in 2012, and the Ph.D. degree in computer science from the University of Wisconsin–Madison in 2017. He is currently an Associate Professor with Shanghai Jiao Tong University. His research interests include data center networking, networking for AI, AI for networking, and machine learning systems.

Chen Tian (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China, in 2000, 2003, and 2008, respectively. He is currently a Professor with the State Key Laboratory for Novel Software Technology, Nanjing University, China. Previously, he was an Associate Professor with the School of Electronics Information and Communications, Huazhong University of Science and Technology, China. From 2012 to 2013, he was a Post-Doctoral Researcher with the Department of Computer Science, Yale University. His research Interests include data center networks, network function virtualization, distributed systems, internet streaming, and urban computing.