LAPS: Latency Aware Packet Spraying on Unequal-cost Multi-path Data Center Networks

Ying Wan, Jinyu Xiao, Haoyu Song, *Senior Member, IEEE,* Zhikang Chen, Yu Jia, Yunhui Yang, Bin Liu, *Senior Member, IEEE,* and Tao Huang, *Senior Member, IEEE*

***Abstract*—Load balancing and congestion control are critical for improving application performance in data center networks. However, the assumption of equal-cost paths no longer holds for newer network topologies and workloads for HPC and AI. The existing flow-based load-balancing schemes fail to adapt to such networks. The random packet spraying is also ill-suited. To address the problem, we propose LAPS, a simple latency-aware packet spraying scheme, to achieve joint load balancing and congestion control regardless of network topology and traffic pattern. As a coherent load-balancing and congestion-control solution, LAPS manages both packet send rate and distribution based on the real-time one-way path delay information. It can be adapted to both TCP and RoCE-based transport protocols and deployed on SmartNICs at a low implementation cost. Evaluations show that LAPS consistently outperforms the other load-balancing and congestion control schemes on unequal-cost multi-path topologies for HPC/AI workloads.**

***Index Terms*—Load balancing, congestion control, data center networks, packet spraying, unequal-cost multi-path, path latency.**

I. Introduction

As a core infrastructure in the age of AI, data center has extended its role from providing cloud services to supporting large model training/inference and distributed machine learning. Data Center Network (DCN) provides the communication support for computing nodes, which is critical in boosting the application performance and reducing the system cost. For large model training, large amounts of intermediate data are frequently exchanged in a large cluster due to different parallelism strategies, and multiple jobs can also run in parallel in a multi-tenant scenario. The resulting traffic surges can easily inundate the network. A slowdown of any flow will hurt the collective communication efficiency. Therefore, in addition to effective traffic scheduling and congestion control, the multi-path opportunities in the network must be exploited to distribute the traffic in conjunction with efficient end-to-end flow congestion control, in order to fully utilize the available network bandwidth and avoid creating hot spots.

Load-balancing (LB): Many multi-path LB schemes have been proposed and some are widely deployed (e.g., ECMP [2] and Flowlet [3]). Most of these schemes assume the equal-cost paths which is indeed the case for the predominant DCN topologies based on multi-rooted trees (e.g., Fat-Tree [4] and Spine-Leaf [5]). Such topologies present a symmetric and layered structure and the multiple candidate paths between each pair of end hosts are equivalent, so the only goal for LB is to distribute the traffic on these paths as evenly as possible, with finer and finer distributing units from flow [2] to flowlet [3] to flowcell [6] to packet [7].

The traffic pattern of distributed large model training exhibits several distinct characteristics [8], [9], [10]: while multiple jobs run in parallel, each job produces a relatively small number of huge, concurrent, and intermittent flows. The low flow entropy causes the flow-based LB ineffective and the bursty data hardly generate any flowlets. These make the packet-based LB more attractive and even inevitable. No wonder Ultra Ethernet Consortium (UEC), dedicated to Ethernet-based AI and HPC optimization, has listed packet spraying as a key feature [11].

However, the current packet spraying schemes assume the equal-cost paths too. Unfortunately, the path asymmetry in bandwidth or length can lead to significant load-unbalancing because the packet spraying only averages the use of output links on a per-switch basis without caring for the path cost. Even the weighted spraying cannot solve the problem due to path/link diversity and traffic dynamics: given the dynamic traffic patterns and multiple unequal-cost paths for each flow, each link of a path may present a fast changing weight for each packet, making the overall load-balancing difficult to achieve through static weights.

Driven by the AI and HPC workloads, some new trends on the DCN topology emerge. First, certain switch-less topologies, such as Torus [12] (an example is shown in Fig. 1(a)), have been widely used which provide multiple possible paths between each pair of nodes, but the load of each link is time-

Manuscript received September 30, 2025; accepted June 13, 2026; approved by IEEE TRANSACTIONS ON NETWORKING Editor Dinesh Bharadia. Date of publication Xxxxxx xx, 202x; date of current version Xxxxxx xx, 202x. The work of Bin Liu was supported by National Key Research and Development Program of China under Grant 2024YFE0203900. The work of Ying Wan and Jinyu Xiao was supported in part by the State Key Laboratory of Internet Architecture under Grant HLW2025ZD08 and Grant HLW2025MS13 and in part by Tencent Rhino-Bird Joint Research Program under Grant JR2026TEG624. An early version of this work [1] was presented in part at the 2nd SIGCOMM Workshop on Networks for AI Computing (NAIC), 2025. That paper has been modified and revised to reflect comments received at the conference. *(Corresponding author: Haoyu Song and Tao Huang.)*

Ying Wan is with the School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China, and also with the Future Advanced Network Laboratory, Nanjing 211189, China (e-mail: wy25@seu.edu.cn).

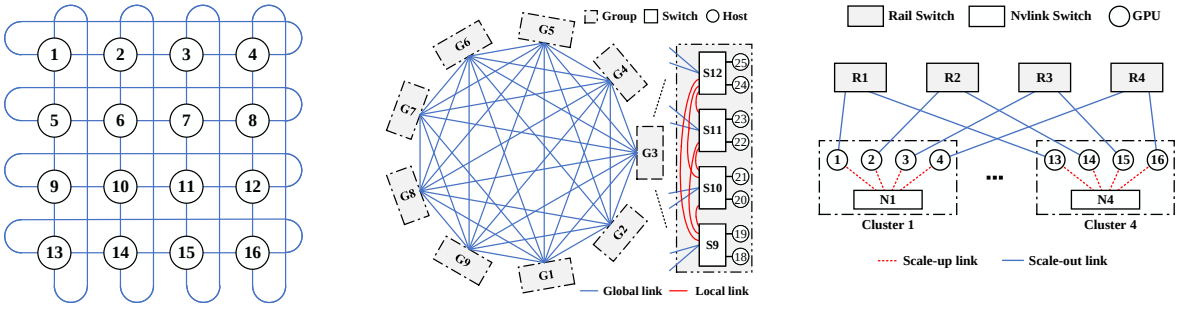
Jinyu Xiao is with the School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China (e-mail: 220255493@seu.edu.cn).

Haoyu Song is with Futurewei Technologies, San Jose, CA 95131 USA (e-mail: shykcl@gmail.com).

Zhikang Chen and Bin Liu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.

Yu Jia and Yunhui Yang are with the China Mobile (Suzhou) Software Technology, Suzhou 215163, China.

Tao Huang is with the Purple Mountain Laboratories, Nanjing 211111, China.



(a) Torus. (b) Dragonfly. (c) Rail.

Figure 1: Typical AI DCN topologies. Switches and servers are represented by squares and circles.

variant caused by changing communication patterns, making the paths with the same length effectively unequal, let alone the other detours. Second, the high-radix, low-diameter networks, such as Dragonfly [13] and Jellyfish [14], are increasingly adopted which require the non-minimal adaptive routing to access the path diversity. Recent AI-oriented DCN architectures further reinforce this trend. For example, UB-Mesh [15] proposes a hierarchically localized nD-FullMesh topology for LLM training and uses All-Path Routing (APR) to exploit path diversity across minimal and non-minimal paths, so as to improve bandwidth utilization. Third, due to the heavy use of GPU/TPU clusters for large model training, heterogeneous interconnection technologies are adopted for intra-cluster scale-up network (e.g., NVLink [16] and UAL [17]) and inter-cluster scale-out network (e.g., IB [18] and UEC [11]). There is ongoing discussion on the convergence of the two networks with the Ethernet-based technologies and protocols [19], which also provides multiple paths between each pair of end nodes in different clusters. In each case above, LB faces the unequal-cost multi-path problem which is challenging for the packet-spraying-based schemes in particular.

Congestion-control (CC): LB and CC are used to being considered as two orthogonal aspects for network performance optimization, and work independently. But in our circumstance, they are entangled and should be considered together for the following reasons.

First, conventional CC algorithms (e.g., DCTCP [20] and DCQCN [21]) take ECN and packet drop as congestion signals to trigger flow rate adjustments. In the case of packet spraying, such signals become unreliable and less meaningful, because an individual packet's experience on just one path can unfairly determine the fate of the entire flow, resulting in frequent and aggressive rate reduction which hurts the Flow Completion Time (FCT). Second, the InfiniBand (IB) transport protocol used by RDMA and some TCP variations require rigid packet delivery ordering in each flow to avoid significant performance degradation. For example, RDMA transport considers any out-of-order (OOO) packet delivery in a transaction as packet drop which triggers an expensive Go-Back-N re-transmission process. Unfortunately, packet spraying makes OOO unavoidable. Moderate OOO can be tolerated by adopting certain techniques [7], [22], [23], [24]. However, packet spraying on unequal-cost paths can lead to a high OOO rate which requires a large reordering buffer, making the implementation impractical. Third, multi-path load-balancing cannot solve the in-cast problem which happens at the end node, and the signals such as ECN cannot differentiate this case from the congestion on paths, making the reaction either too retarded or overly sensitive. Fourth, PFC used by RoCEv2 effectively renders certain links unusable during congestion, but this is not sensible by the end nodes on time, so packets are still dispatched to the congested link, aggravating the situation although other uncongested paths are available.

The gist of this paper is to *enable effective packet spraying to achieve global load-balancing and end-to-end flow congestion-control across arbitrary network topologies and traffic patterns.* We believe the link bandwidth, path length (e.g., hops), and even *local* congestion status (e.g., queue depth) are not effective metrics of path cost for the transient and intermittent traffic from multiple independent jobs. Instead, the realtime end-to-end one-way path latency should be the deterministic factor considered to choose a packet's forwarding path, since the path latency consolidates the effects of all the other factors. Intuitively, packets should always be sprayed on one or more paths with the smallest latencies. Such paths may keep changing, adapting to the current network dynamics. Meanwhile, the measured path latencies can also serve as a more indicative congestion signal. As long as some candidate paths present low latencies, the flow can continue to send with increasing rate; only when all the paths present higher than expected latency, which likely reflects an in-cast scenario, should the flow reduce its sending rate.

To realize it, a NIC-based solution is preferable for the following reasons: (1) it allows the complete coverage of the end-to-end paths and joint consideration of LB and CC, so all potential congestion points including the in-cast are covered; (2) it is applicable to all kinds of networks including the switch-less ones; (3) it can take advantage of the programmable SmartNIC to offload the algorithm, avoiding the scalability concern and the reliance of increasingly unacquirable programmable switches. Thanks to the general purpose processors, a SmartNIC allows more complex computing and sophisticated algorithms than a switch. Although, we still need to overcome several technical hurdles: First, the host needs to compute and maintain candidate paths for each flow. Second, the network needs to sense the real-time path

latency to make timely packet spraying decision. Third, we need an efficient method to direct packets to follow specific end-to-end paths. Fourth, we need a new algorithm to handle the latency-based congestion signals, OOO, and lost packet retransmission. To this end, we propose the Latency Aware Packet Spraying (LAPS) scheme.

The remainder of the paper is organized as follows. Sec. II provides the background. Sec. III describes the architecture and implementation of LAPS. Sec. IV presents the performance evaluation. Sec. V summarizes the related works. Finally, Section VI concludes the paper.

II. Background

When moving packets from A to B with multiple unequal-cost paths, the most sensible choice is to take the fastest way if the current delay information can be acquired in advance, and only if all the paths are congested, shall the sending rate be slowed down to avoid worsening the situation. LAPS follows such a straightforward principle to design its LB and CC mechanisms jointly.

A. Unequal-cost Multi-path

Fig.1 illustrates a few typical network topologies in today's AI data centers. Most of them feature unequal-cost multi-path. In the Dragonfly network, switches and servers are divided into nine identical groups. Four switches in a group are interconnected through six local links to form a full mesh, and nine groups are interconnected through 36 global links to form another level of full mesh. From S0 to S6, the network provides three 3-hop paths, 12 4-hop paths, 35 5-hop paths, 99 6-hop paths, and 338 7-hop paths. In the Torus network, in addition to the set of shortest paths between each pair of nodes, many circuitous paths are possible. In the mixed scale-up and scale-out network, to reach a node in another cluster, a packet can be routed to any node in the same cluster first before being forwarded to the scale-out network; next, the packet can be routed to any node in the target cluster first before being forwarded to the target node.

Different parts of the network can also have different link bandwidth. For example, in the mixed scale-up and scale-out network, the scale-up part usually has higher link bandwidth than the scale-out part. More importantly, the load on each link undergoes rapid and substantial changes. Since each link can be involved in multiple paths for multiple communication pairs, such dynamics are difficult to predict.

B. Path Finding

Although a plethora of choices are available, it is unnecessary and uneconomic to use all possible paths for packet forwarding. Usually, it is sufficient to only consider the top-k paths which include the optimal and some sub-optimal paths in terms of path cost. The choice of k is subject to the network type and scale. For example, in the Dragonfly network in Fig.1(b), the 3-hop and 4-hop paths (15 in total) provide a good balance between path diversity and maintenance cost. The choice of k may also depend on the adjacency degree of two nodes. For example, in a Torus network, more paths should be considered when two nodes are further apart. The k-shortest path algorithm [25] can be used to find all candidate paths. The path calculation can be performed in advance for a fixed topology. It is easy to mark a candidate path invalid dynamically due to switch or link failures on the path.

C. In-band Network Telemetry

In-band Network Telemetry (INT) [26] uses a custom header in application packets to instruct certain network nodes to collect and process specific network data which can infer network performance. INT has become a default feature in commercial switches (e.g., Broadcom [27]). In contrast to out-of-band active measurements, INT has the advantages of low overhead (the measurement data is carried in application packets), low feedback latency (the measurements can be refreshed in one RTT), and high accuracy (it reflects the real experience of the application traffic). LAPS only needs the end-to-end path latency, so we use INT to only collect the packet timestamps on the path head and end nodes. Since we only compare the relative latency difference among the paths between a node pair, there is *no* need for the nodes to synchronize their clocks.

D. Source Routing

LAPS balances the traffic load at path level, requiring the candidate paths to each receiver to be explicitly maintained at a source node, and each packet pinned down on a path. Source Routing (SR) is thus a natural choice to support the non-minimal adaptive routing in unequal-cost multi-path networks. To forward a packet to another node, the head node first figures out the candidate paths based on the packet's destination address, and then inserts a path-defining SR header according to certain path selection criteria. The packet is forwarded to the end node guided by the SR header.

For a DCN with n nodes and k paths maintained for each node pair, an end node needs to maintain a path selection table with $(n-1)k$ entries at most. In reality, each node only needs to maintain the candidate paths to the nodes with which it needs to communicate. Such information is acquirable from the job scheduler in a multi-tenant environment. The paths can be dynamically calculated or loaded once a new job is scheduled. Similarly, after a job is done, the unused paths can be revoked to save the memory resource.

E. Multi-path Packet Spraying

We consider the path latency as the dynamic path cost. The packets should be sprayed to different paths accordingly. We use a general Softmax function to model the packet distribution strategy as shown in Equation 1.

$$P(d_i) = \frac{e^{-\beta d_i}}{\sum_{i=1}^{k} e^{-\beta d_i}}, \forall i \in [1, ..., k] \tag{1}$$

In the equation, d_i is path i's measured latency, and $P(d_i)$ is the probability that path i is selected to send a packet. The non-negative real-value parameter β is used to adjust the

probability distribution. When $\beta = 0$, the strategy degenerates into RPS [7]: the packets are evenly distributed to all the candidate paths regardless of their latency. Clearly, this will lead to serious load unbalancing. On the other hand, increasing the value of β will make the probability distribution more concentrate toward the paths of the smallest latencies. When β is large enough, the Softmax function degenerates into an ArgMax function, which means only the path(s) with the minimum latency will be chosen for packet forwarding. The fastest-path-only strategy has its merits: it reduces the path churning for each flow with the best effort, so the OOO delivery problem is mitigated. However, such a strategy may also cause fast fluctuation of path latency and introduce instability to the network. It is interesting to find the optimal β value as the trade-off for best overall performance.

F. Congestion Control & Lost Recovery

Since the packet spraying preference is always on the fastest path(s), low latency on these paths implies the flow sending rate can be increased. When all the paths show a higher than expected latency, it is likely a congestion at the path end (i.e., in-cast) happens, suggesting a flow rate reduction. Latencies as the collective congestion signal is more expressive and stable than ECN in our network scenario.

We like to retain the "lossless" feature of the DCN by enabling the functions such as PFC, but the harsh action of PFC may seriously affect the performance due to HoL blocking and pause propagation. Fortunately, the LB and CC of LAPS can effectively reduce the PFC probability. Before PFC is triggered, the queue buildup is already reflected in the link latency increase, making the paths using this link unfavorable for the active flows so the congestion on the link can be mitigated.

The latency-based packet spraying is also helpful to reduce the OOO rate due to the moderate path stickiness, so a smaller reorder buffer is needed on the receiver NIC. However, OOO is unavoidable so we must differentiate it with packet drop events for efficient retransmission.

A key observation is that packets distributed on the same path are delivered in order if not lost. If their ACKs also follow the same path, then the ACKs should be received in order. Hence, if the ACKs received for a path skip an expected sequence number, it can be considered that the corresponding packet is lost, and a retransmission can be issued immediately. Meanwhile, if the sending window is already saturated and the ACK for the first packet is not received after the RTT for a path, the packet is also retransmitted. The receiver may receive more than one copy of a packet due to mistaken retransmissions. The duplicated packets are ACKed and dropped.

III. Architecture & Implementation

A. Architecture Overview

Fig. 2 illustrates the architecture of LAPS where all the major functions are implemented on the SmartNIC or DPU associated with the end hosts. The source node sprays packets across selected paths with a higher probability towards paths with lower latency. Intermediate switches forward the packets

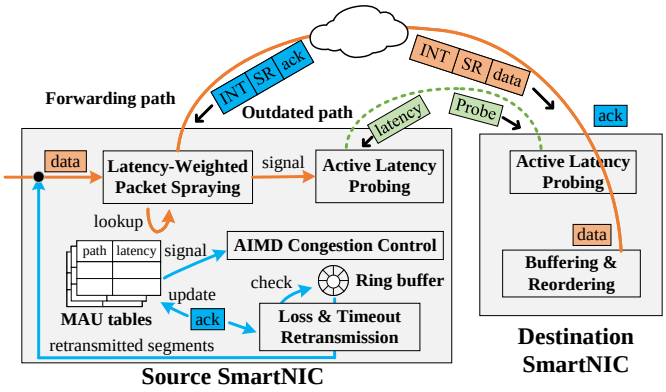


Figure 2: The architecture of LAPS.

according to the path information embedded in the SR header. The destination node acknowledges the received data segment by sending Acknowledge packet (ACK) along the reversed path acquired from the SR header. The ACK also embeds the latest forwarding path latency for the source to refresh its path latency information. At the source node, if the latency of certain path is outdated, which may be due to the lack of packets sent on that path or no returned ACK for it for a while, it sends out a probe packet along that path to obtain its latency. The source node adjusts a flow's sending rate using the "Additive Increase and Multiplicative Decrease" (AIMD) approach based on the real-time latency measurements of all its candidate paths for congestion control. The source node detects packet loss through ACKs and performs selective retransmissions. If no ACK is received for a sent packet within a timeout period, a retransmission of the missing packets on this path is triggered.

B. Packet Format

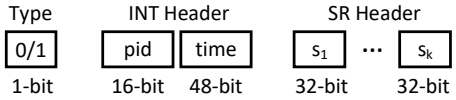


Figure 3: Custom packet header format in LAPS.

To support the above functions, LAPS embeds a custom header in each packet, which consists of three parts[1]:

- **Type**: The 1-bit `type` differentiates the normal data/ACK packets from the probe/ACK packets. The source node sets this flag when sending a probe packet and the destination node keeps it set in the responding ACK. The probes and their ACKs are generated and terminated at NIC.
- **INT**: The INT header contains two fields, `pid` and `time`. For a data packet, `pid` and `time` record the forwarding path ID and the sent time from the source, respectively. For an ACK packet, `pid` is copied from the corresponding data packet, and `time` represents the one-way path latency calculated from the receiver's local time and `time` carried in the data packet. `time` ranges from 0 to 2^{48} microseconds.

[1]It is easy to adapt such information to standard protocols such as SRv6, if needed. We omit the discussion for simplicity.

We use a 16-bit field to identify 65,536 global paths which is sufficient for most networks. It is straightforward to extend the width of `pid` for ultra-large networks[2].

- **SR**: The header contains a series of 32-bit fields $\texttt{s}_1, \texttt{s}_2, \ldots$, in which $\texttt{s}_i$ identifies the i-th node on the forwarding path[3]. A path does not need to enumerate all the nodes on it hop by hop. A few key anchor points are sufficient if the shortest path between any two adjacent anchor points is supposed to be taken for the path.

Assuming a path is specified by two or three anchor nodes, which is sufficient for most DCNs (e.g., Dragonfly and Fat-Tree), the additional overhead is about 20 bytes per packet, accounting for 1.3% for 1500-byte IP packets.

C. Match-Action Tables

LAPS maintains and uses two tables:

- **Path Search Table (PST)**: `PST` uses the packet's destination address as the key for Exact Matching (EM) to obtain the IDs of candidate forwarding paths.
- **Path Information Table (PIT)**: Indexed by path ID, `PIT` maintains the path information. A path can be labeled as invalid using the `valid` field, possibly due to temporary network failure, to be excluded from consideration. `baseVal` represents the baseline latency of an uncongested path. `time` and `realVal` represent the latest measurement time and the measured latency, respectively. `anchors` indicates the anchor points defining the path.

DstIP	Pids
ip1	p1, p2

Pid	valid	time	baseVal	realVal	anchors
p1	1	50	15	15	s_a, s_b, ...
p2	1	42	10	8	s_a, s_c, ...

Figure 4: Two EM match tables in LAPS.

On each NIC, LAPS stores at most n-1 EM entries in `PST` for a network of n hosts. Additionally, LAPS consumes $O(nk)$ EM entries in `PIT` to store path information, where k is the number of paths between any pair of hosts. Current NICs can easily support EM tables with millions of entries, so the memory resource consumption of LAPS is acceptable even for large-scale networks.

D. Latency Measurement

LAPS measures path latency with two methods:

- **INT**: For each data packet, the sender sets the `pid` and `time` fields of the INT header; the receiver calculates the path latency as Δt = Now()-`time` and uses the INT header of the corresponding ACK to send the result back. Upon receiving it, the sender updates `realVal` and `time` of the entry `PIT`[`pid`] to Δt and Now(), respectively.
- **Active Probing (AP)**: LAPS times out the PIT entry for a path `pid` if the time elapsed since the last update exceeds 2·`realVal`. AP is used to refresh the entry `PIT`[`pid`]. In this case, a probe packet is sent on the path to obtain its latest status. Meanwhile, `PIT`[`pid`].`time` is set to Now(), and `PIT`[`pid`].`realVal` is temporarily doubled which not only reduces the probing frequency on congested or failed paths, but also ensures that such paths are not selected to forward packets.

[2]A path can also be uniquely identified by the source node ID plus a path ID which has only a local scope for the node.

[3]Other facilitating fields such as path length and node pointer are omitted for simplicity. Node address can also be compressed to lower overhead.

E. Latency-Weighted Packet Spraying

To send a packet, LAPS first looks up `PST` and `PIT` to determine the latest status of all candidate paths. Then, based on the probabilities calculated by Equation 1, the packet is distributed to a path.

Algorithm 1 details the latency-weighted packet spraying process. Specifically, invalid paths are given zero weight. If all paths `p` appear congested, i.e., `p.realVal>p.baseVal`, the CC of LAPS will kick in to reduce the sending rate (see Section III-F). AP may also be triggered during path evaluation.

When the receiver needs to send an ACK for a received data packet, it simply uses the reversed path indicated by the SR header of the packet. The decision is based on the following considerations: (i) It avoids path lookups at the receiver. (ii) The small ACK packets do not cause load imbalance even if distributed unevenly. (iii) Network devices are configured to give ACKs higher priority than data packets so they will not experience queuing delay. (iv) The fact that an ACK takes the reverse path of the corresponding data packet enables efficient loss detection and fast retransmission (see Section III-G).

Algorithm 1: Latency Weighted Packet Spraying

```
Input: pkt
paths = [PIT[pid] for pid in PST[pkt.dip].pids]
weights = [0 for p in paths]
for i in [0, weights.size()) do
    if paths[i].valid then
        if Now()-paths[i].time>2·paths[i].realVal then
            ActiveProbing(paths[i])
            paths[i].realVal = 2 · paths[i].realVal
            paths[i].time = Now()
        weights[i] = exp(-β· paths[i].realVal)
fwdPath = SelPathByWeight(paths, weights)
AddHdrAndTxData(pkt, fwdPath)
```

F. Congestion Detection and Control

LAPS does not use ECN for congestion detection. Instead, it adjusts the sending rate by comparing the real-time one-way path latency with the baseline latency of each path. For each path, the baseline latency is initialized as the uncongested transmission delay of data packet plus a small queueing margin equivalent to five packets at each switch along the path. This margin allows LAPS to tolerate transient queue buildup caused by short microbursts and avoids unnecessary rate reduction when the path is only temporarily backlogged. On the one hand, LAPS decreases the traffic distribution on paths with increasing latencies, which mitigates their congestion without

Algorithm 2: Congestion Detection and Control

```
Input: ack
PIT[ack.intHdr.pid].realVal = ack.intHdr.time
PIT[ack.intHdr.pid].time = Now()
paths = [PIT[pid] for pid in PST[ack.sip].pids]
T = Max { p.baseVal | p ∈ paths }
t = Min { p.realVal | p ∈ paths }
if t > T and nxtDec > Now() then
    expRate = curRate, curRate = curRate·1/2
    nxtDec = Now() + 2 · t
    incStage = 0
else if t ≤ T and nxtInc > Now() then
    incStage = incStage + 1
    if incStage > N then
        expRate = 2 · expRate
    curRate = (expRate + curRate) · 1/2
    nxtInc = Now() + 2 · t
```

Algorithm 3: Loss Detection and Recovery

```
Input: ack
pid = ack.intHdr.pid
realVal = ack.intHdr.time
outStdSegs = rBuf[pid]
it = outStdSegs.begin()
while it != outStdSegs.end() and ack.seq != it.seq do
    lostSegs.append(it.seq)
    it = outStdSegs.remove(it)
outStdSegs.pop_front()
ExecEventAfter(ReTx(pid), 2·realVal)
Function ReTx(pid):
    outStdSegs = rBuf[pid]
    lostSegs.append(outStdSegs)
    outStdSegs.clear()
```

reducing the flow send rate. However, when traffic is heavy, or during in-cast at the receiver node, the latencies of all paths for a flow tend to converge to the largest baseline latencies of these paths T. At this point all paths have equal weight, which makes it no longer possible to mitigate congestion by adjusting traffic distribution. Hence, LAPS starts to reduce the flow send rate.

As shown in Algorithm 2, LAPS uses a simple AIMD approach to adjust the flow rate. When a flow starts, both the current and expected sending rates are initialized to the maximum sending rate allowed by the NIC rate limiter. When receiving an ACK, the sender first parses the INT header and updates the corresponding PIT entry. Then, it detects if all the valid paths have higher latency than T. If true, the sender sets the flow's expected rate to the current rate and halves the current rate; otherwise, the flow's current rate is linearly increased by half of the difference between the expected rate and the current rate. Rate adjustment is only allowed once within a 2T time window to prevent over-adjustment before the effects are perceived by the sender.

The algorithm allows unused bandwidth to be explored. For example, if the flow's expected rate expRate is $2a$, and its current rate curRate starts from a and keeps increasing for N stages. At this point, curRate is close to $2a$ and the increment of curRate per stage is only $\frac{a}{2^N}$. To explore further, LAPS doubles expRate to $4a$ to increase the rate by $\frac{4a-2a}{2} = a$. If this causes congestion, LAPS halves curRate from $3a$ to $\frac{3a}{2}$ and makes $3a$ as the new expRate for subsequent linear increases. In our evaluation, we set N=5 by default, which provides a balance between bandwidth probing speed and the risk of rate oscillation.

After N consecutive linear increases (i.e., IncStage==N) and expRate has been doubled, IncStage is not reset. Consequently, if doubling expRate does not cause congestion, it will continue to be doubled during the next increase, which can be referred to as the Hyper Rate Increase (HAI) phase. IncStage is only reset to zero when decreasing rate due to congestion. This approach allows rapid probing to find the maximum available bandwidth.

G. Loss & Timeout Retransmission

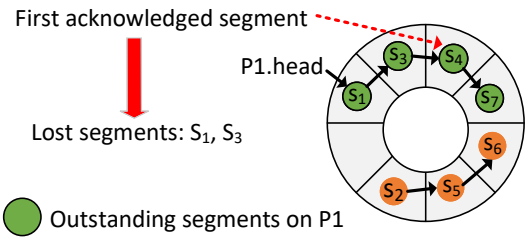


Figure 5: The ring buffer for detecting lost packets.

Even in lossless Ethernet, packet loss may happen due to malfunctioning hosts, switches, or links. Multi-path packet spraying makes it difficult to differentiate between OOO delivery and packet drops. To address this challenge, LAPS designs a simple yet effective packet loss detection and selective retransmission mechanism.

As described in Section III-E, LAPS makes an ACK packet use the reverse path of the acknowledged data packet. Therefore, any OOO ACK on a path implies likely packet loss, and the sender can immediately selectively resend the deemed lost packet. To achieve this, LAPS maintains a ring buffer at the sender for each flow, recording the sequence numbers of data packets sent along each path that have not yet been acknowledged. Upon receiving a data packet, the receiver sends an ACK to selectively acknowledge the current data segment. The sender, based on the ACK packet, not only knows the acknowledged data segment but also the transmission path pid from the INT fields. The sender then indexes the corresponding ring buffer area using pid and compares the unacknowledged data segments sent along this path until it finds the currently acknowledged segment x. Consequently, any data segments before x are deemed lost. LAPS will immediately retransmit these lost data segments in sequence before sending new data.

To handle the tail drops where there is no ACK for sender to trigger the loss retransmission, LAPS implements a fast timeout retransmission mechanism. Upon sending a data packet along the path p, sender resets a timer to start retransmitting all the out-standing packets on p if no ACK are returned after time 2*p.realVal. Each time an ACK is received, the sender resets the timer according to the latest p.realVal.Algorithm 3 details the retransmission process of LAPS due to packet loss or timeout.

H. Implementation

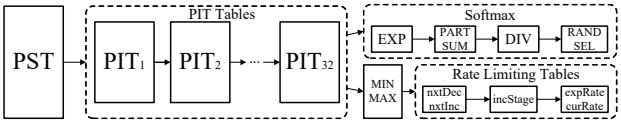


Figure 6: Pipelined implementation of LAPS.

To verify the feasibility of LAPS on an FPGA-based NIC, we implement a prototype on the Xilinx Alveo U280 Accelerator Card by leveraging the open-source projects FastRMT [28] and Corundum [29]. Algorithm 1 and Algorithm 2 can be deployed on the FPGA, but Algorithm 3 requires a ring buffer and associated hardware for each flow, which exceeds the resources available on the FPGA. Instead, Algorithm 3 can be adapted to run on multi-core processor-based SmartNICs such as AMD Pensando [30] and Intel IPU [31], given the lenient timing constraints for lost packet detection.

Overall, LAPS can be realized as a pipeline composed of a series of Match-Action Units (MAU) which fit in the classic RMT model [32], [33]. Unfortunately, programming Algorithm 1 and Algorithm 2 in P4 [34] presents a significant challenge. The algorithms involve multiple accesses to `PIT`, loops, and the Softmax function calculation, none of which are supported by P4. We need to overcome these challenges in an FPGA-based implementation.

Decomposing `PIT`. In an RMT pipeline, each table can only be accessed once. However, Algorithms 1 and 2 both need to access `PIT` for each `pid` found in `PST`. If the entire `PIT` is placed in a single SRAM, it would need to be read from the SRAM until the lookups for all `pids` are completed, creating a bottleneck in the pipeline. To address this issue, we decompose `PIT` into 32 sub-tables in our prototype. We retrieve 32 `pids` at once from `PST` and then look them up in parallel across the sub-tables and potentially write back to one of them (lines 1-2, Algorithm 2). Except for the valid `pids` in `PST`, the other `pid` fields are configured as 0, and the `valid` field of the 0th entry in each `PIT` sub-table is always set to 0, with the `realVal` field always set to `inf` to ensure it is not selected by packet spraying. Through control plane management, we ensure that no two `pids` mapped to the same `DstIP` appear in the same `PIT` sub-table.

Transforming loops into parallel circuits. Algorithm 1 iterates through each path to determine whether a probe packet is needed. For simplicity, only one probe packet for one randomly selected path among all the outdated paths is sent. This reduces the active probing frequency and delays the latency updates for some paths, but it has little impact on the overall performance. Meanwhile, calculating the path weight for packet spraying is unavoidable, so the Softmax calculation is performed in parallel on all valid paths. Similarly, Algorithm 2 iterates through each path to calculate the maximum value T of `baseVal` and the minimum value t of `realVal`. The maximum and minimum values are computed hierarchically through two groups of tree-structured comparators.

Implementing the Softmax function. Given the 48-bit timestamp in LAPS, implementing the Softmax function in Algorithm 1 would consume a significant amount of FPGA resources. We employ a series of techniques to reduce its resource consumption. Since $-\beta d_i < 0$, we have $e^{-\beta d_i} \in (0, 1)$, which has an integer part of 0. Therefore, we use fixed-point numbers instead of floating-point numbers. The bit width of the fixed-point numbers is compressed to 16 bits. In addition, we precompute e^{β} to avoid the multiplication between β and `paths[i].realVal`, and apply the Basic-Split technique [35] to convert the calculation into lookup tables, additions, and divisions.

IV. Evaluation

We conduct extensive simulations using NS3 [36] to compare LAPS with the following schemes: flow-based LB (ECMP, PLB [37]), Flowlet-based LB (LetFlow, Conga), and ConWeave [38]. CONGA is also based on INT and SR, but it is designed for equal-cost paths and works on flowlets. We reuse the implementations of ECMP, PLB, LetFlow, Conga, and ConWeave from prior work [39], [40], [38], [41]. We also implement the LAPS prototypes on a Xilinx UltraScale+ FPGA card, Alveo U280 [42]. All the open-sourced code is available at [43].

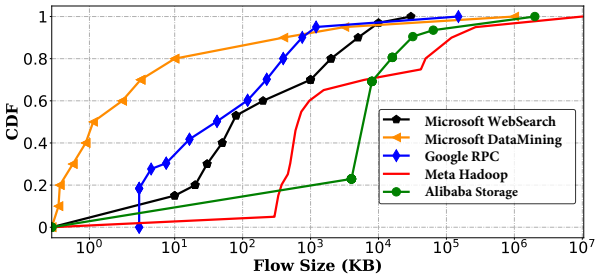


Figure 7: The traffic distribution of workloads.

A. Testbed

Topologies: We run simulations on two typical DCN topologies: Dragonfly (Fig. 1(b)) and Rail (Fig. 1(c)), which are widely used for high performance computing and distributed training. The Dragonfly network comprises nine groups with 36 switches and 144 servers, and the Rail network comprises eight clusters with eight Rail switches, eight intra-node Nvlink switches, and 64 servers. In the Dragonfly network, not only the minimal path but also a set of sub-optimal paths are considered: for inter-group traffic, each group is used as a midpoint to bridge the two sections of the shortest paths from the source server and to the destination server; for intra-group traffic, each switch of the same group is used as a midpoint. Thus, there are three and eight candidate paths for intra-group and

inter-group communication, respectively. In the Rail network, GPUs within the same cluster communicate directly through the Nvlink switch that connects them. For inter-cluster GPU communication, the data is routed through all Rail switches. The number of available paths thus equals the number of Rail switches (8). This setup allows us to comprehensively evaluate the effectiveness of our load-balancing and congestion-control strategies under varying network conditions.

Table I: Per-Host PST and PIT entry counts

Topology	Dragonfly		Rail	
Category	Intra-Group	Inter-Group	Intra-Cluster	Inter-Cluster
# PST entries	3+3×4	8×4×4	7	7×8
# PIT entries	3×1 + (3×4)×3	(8×4×4)×8	7×1	(7×8)×8

Table I summarizes the per-host sizes of `PST` and `PIT` in the evaluated Dragonfly and RailOnly topologies. In LAPS, each source host s maintains one `PST` entry for each destination host d, and the `PST` entry stores the IDs of the candidate paths to d. Therefore, the number of `PST` entries is 143 for the 144-server Dragonfly topology and 63 for the 64-GPU RailOnly topology. The number of `PIT` entries equals the total number of candidate paths referenced by all `PST` entries of the source host. In Dragonfly, if d is attached to the same switch as s, LAPS maintains one direct path. If s and d are in the same group but attached to different switches, LAPS maintains three candidate paths: the minimal-hop direct path and two paths through the other two switches in the group. If s and d are in different groups, LAPS maintains eight candidate paths, including one direct inter-group path and seven paths through the other groups as intermediate groups. Thus, each host maintains $3\times1+(3\times4)\times3+(8\times4\times4)\times8$=1,063 `PIT` entries in Dragonfly. Similarly, each GPU in RailOnly has seven intra-cluster destinations with one candidate path per destination and 7×8 inter-cluster destinations with eight candidate paths per destination, resulting in $7\times1+(7\times8)\times8$=455 `PIT` entries.

It is worth noting that the above path construction only specifies the key waypoints that distinguish different candidate paths. These waypoints are exactly the anchors that need to be encoded in the SR header. The intermediate nodes on the shortest-path segment from the source to the first anchor, between two adjacent anchors, and from the last anchor to the destination do not need to be explicitly carried in the SR header. Therefore, even for the longest inter-group path considered in the evaluated 144-server Dragonfly topology, the SR header only needs to specify one anchor. Besides, the anchor count is independent of the number of servers under the given topology construction and path selection rule.

Workloads: We adopt five representative real-world data center workloads to generate the test traffic: Web Search (WS) [44] and Data Mining (DM) [45] from Microsoft, Remote Procedure Call (RPC) [46] from Google, Hadoop (HDP) from Meta, and Cloud Storage (STR) from Alibaba [47]. The characteristics of the workloads are shown in Fig. 7. The five workloads are all heavy-tailed, with the top 5% of flows accounting for 46.1%, 97.9%, 90.7%, 81.8%, and 77.1% of the total traffic for WS, DM, RPC, HDP, and STR, respectively. We also test the performance of different load balancing algorithms under the distributed training of large language models LLAMA-2 with 7B parameters [48], during which each GPU processes different mini-batch data but shares the same model parameters. At the end of each training step, the gradients from each GPU are aggregated, and the model parameters are updated across all GPUs. Therefore, we evaluate the algorithms in two scenarios: (1) one GPU sends 64MB of data to every other GPU, and (2) every other GPU sends 64MB of data to a single GPU.

Table II: Three used typical communication patterns

Pattern	**(Sender, Receiver)**
All2all	(sv[i], sv[j])
AllReduce	(sv[i], sv[1])
AllScatter	(sv[1], sv[j])

Patterns: We select three representative communication patterns: All2all, AllReduce, and AllScatter in Table II, where sv[i]/g[i] represents the i-th server/GPU. For All2all, the servers/GPUs send traffic to all the other servers/GPUs; For AllReduce, all the other servers/GPUs send traffic to one server/GPU; For AllScatter, one server/GPU sends traffic to all the other servers/GPUs.

Parameters: All the links are set to 100Gbps. The DropTail queue for each interface is set to 150KB ($\sim$100 packets) and the (shared) buffer size of each switch is set to 32 MB. Following typical industry deployments, except for LAPS and PLB, we utilize RoCEv2 [49] as the transport protocol and DCQCN [21] for congestion control and retransmission. PFC is enabled by default. PLB is implemented with DCTCP [41]. The flowlet timeout in LetFlow and CONGA is set to 50us to balance the flow splitting and reorder avoidance according to [50] and [51]. Unless otherwise stated, all parameters are set to their default values as specified in [39], [40]. We also test the algorithm performance under TCP in which SACK is used to handle the packet reordering, and under equal-cost paths in Fat-Tree and Spine-Leaf topologies. The results show that LAPS consistently outperforms the other algorithms in these cases as well. Due to space limitations, we primarily present the results on RDMA scenarios with unequal-cost paths.

Metrics: We compare LAPS with the other LB schemes in terms of average FCT of all flows and the FCT of the 99th percentile (P99) flow under different network topologies, application workloads, communication patterns, and load ratios. In addition, we evaluate the degree of packet OOO, which reflects the reordering pressure at the receivers. Further, we test the retransmission performance of different algorithms by setting a packet corruption probability of 0.0001% on each link. Finally, we analyze the resource consumption on Alveo U280 to show the hardware cost.

B. Simulation Results

Fig. 8$\sim$17 illustrate the FCTs of the LB algorithms under different settings, in which (X, Y, Z) indicates the combination of the topology X, the workload Y, and the communication pattern Z.

Average FCT. The first subfigures in Fig. 8$\sim$17 illustrate the average FCT of all flows. LAPS consistently outperforms the other algorithms in all the cases. At 80% load in (Rail, DM, All2all) shown in Fig. 9, LAPS improves the average FCT

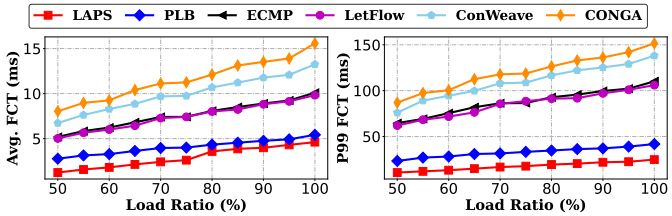


Figure 8: The FCTs on (Rail, WS, All2all).

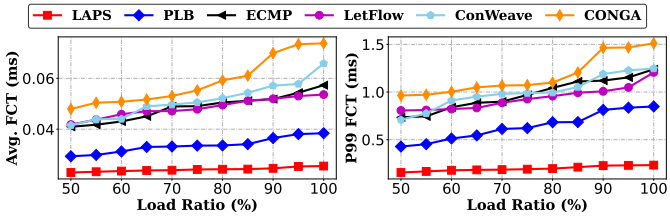


Figure 11: The FCTs on (Rail, HDP, All2all).

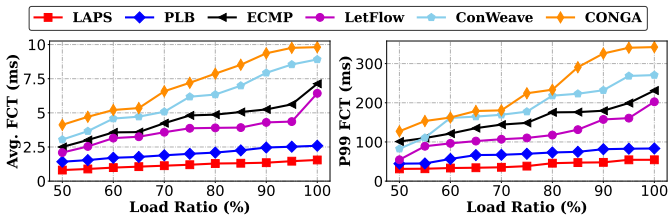


Figure 9: The FCTs on (Rail, DM, All2all).

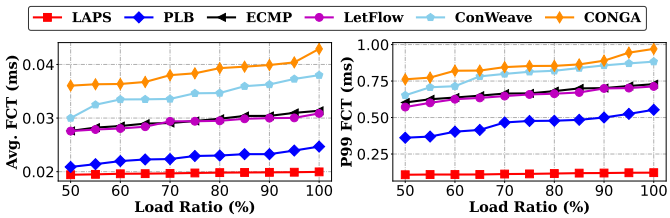


Figure 12: The FCTs on (Rail, STR, All2all).

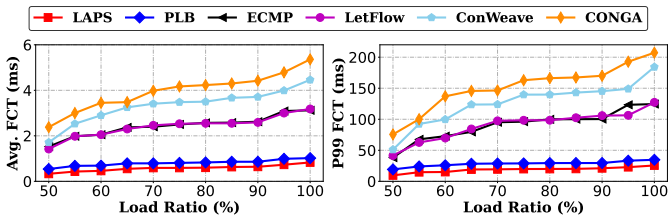


Figure 10: The FCTs on (Rail, RPC, All2all).

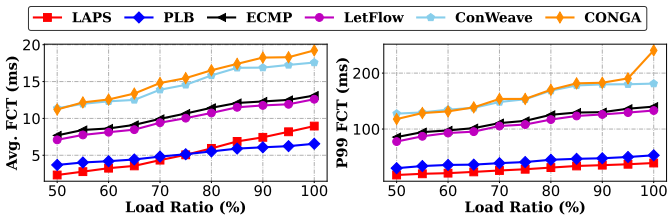


Figure 13: The FCTs on (Dragonfly, WS, All2all).

by 3.3×, 2.8×, 1.6×, 4.8×, and 6.1× compared to ECMP, LetFlow, PLB, ConWeave, and CONGA, respectively. Besides, LAPS exhibits the slowest growth rate in average FCT as the load ratio increases. As shown in Fig. 9, when the load ratio increases from 50% to 100%, LAPS only increases the average FCT by 0.64ms, whereas ECMP, LetFlow, PLB, ConWeave, and CONGA increase it by 4.6, 4.3, 1.1, 5.2, and 5.6ms, respectively.

P99 FCT. The second subfigures in Fig. 8∼17 illustrate the FCT of the 99th percentile flow. A smaller P99 FCT indicates fair treatment of all flows to meet the QoS of latency-sensitive applications such as real-time communication and interactive services. LAPS achieves the best and most stable performance across all scenarios. As shown in Fig. 12, at 80% load in (Rail, STR, All2all) , the P99 FCT of LAPS is only 0.12ms, while ECMP, LetFlow, PLB, ConWeave, and CONGA reach 0.67, 0.70, 0.48, 0.80, and 0.85ms, respectively. On the other hand, as the load ratio increases from 50% to 100% as shown in Fig. 8, LAPS only increases its P99 FCT by 14ms, whereas ECMP, LetFlow, PLB, ConWeave, and CONGA increase it by 46, 44, 18, 62, and 65ms, respectively.

Insight. ECMP randomly selects a path for each flow at the start of transmission. It does not account for the inequality of path costs. Worse still, it does not consider the significant flow size differences as shown in Fig. 7, which can easily lead to path congestion.

Compared to ECMP, both ConWeave and Conga are capable of switching flow paths based on congestion levels or RTT. However, their average and P99 FCT are worse than those of ECMP. A deeper analysis shows that ConWeave uses the destination ToR switch to buffer all packets on the newly selected path until all packets on the original path are passed. While this avoids OOO at the receiver's NIC, it puts pressure on the switch's buffer and delays the arrival of packets on the newly selected path. Additionally, ConWeave requires extra signaling packets to notify the source ToR when the path can be switched. This also increases the FCT and delays the time of switching path, especially for small flows.

Conga sends flowlets to the least congested path. However, flows in RDMA scenarios are sent at the NIC's maximum transmission rate from the start, making it difficult to split flows into flowlets. As shown in Fig. 18, the time interval δ between adjacent packets of the same flow is very small for the AllReduce and AllScatter operations of LLM training on the Dragonfly network, with the probability that δ exceeds the flowlet threshold of 50 microseconds smaller than 0.12%. Consequently, most flows stick to the same path. On the other hand, it considers neither path cost nor the scenarios that multiple switches are congested simultaneously.

Similarly, due to the difficulty of getting flowlets in RDMA as evidenced in Fig. 18, LetFlow transmits each flow almost entirely along the initially selected path until completion, resulting in a performance very close to ECMP. Meanwhile, Conga presents a severe "herd effect" that flows tend to rush to the same least congested path to quickly congest it.

The performance of PLB is closest to that of LAPS. In PLB, a flow uses the same path when it is not congested, avoiding OOO packets at the receiver. When path congestion is detected, the flow is switched to a random path when there are no in-flight packets. Moreover, if a flow detects congestion for

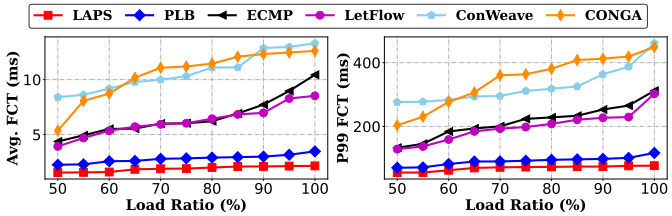


Figure 14: The FCTs on (Dragonfly, DM, All2all).

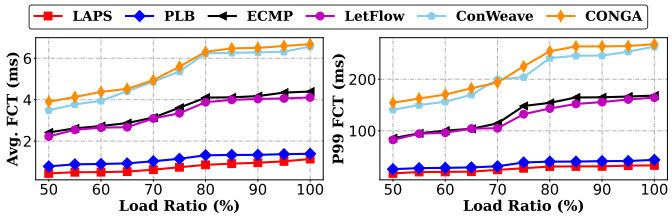


Figure 15: The FCTs on (Dragonfly, RPC, All2all).

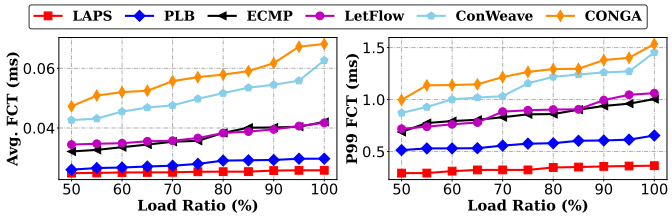


Figure 16: The FCTs on (Dragonfly, HDP, All2all).

12 consecutive times without finding a suitable switching opportunity, it will force the path switching. However, compared to LAPS, PLB gives no special treatment for OOO detection and lost retransmission, but relies on existing CC algorithms (DCTCP) and retransmission algorithms. Additionally, PLB is a flow-level LB scheme which randomly selects a new path for each flow, often requiring multiple attempts to find a suitable path.

LLM training: Table III and Table IV show the average and P99 FCTs for different LB schemes during the gradient aggregation in one round of LLAMA distributed training with and without random packet drops, which involves incast congestion. LAPS outperforms other algorithms in both average FCT and P99 FCT. For instance, given no packet drop, LAPS's P99 FCT is improved by 3.5×, 3.4×, 3.6×, 3.8×, and 1.4× compared to ECMP, LetFlow, PLB, ConWeave, and CONGA, respectively. Table III and Table IV not only demonstrate LAPS's ability to transmit packets along the minimal-cost path but also highlight its superior rate control capabilities under severe congestion.

Traffic Distribution. Fig. 19 shows the traffic distribution across paths with different lengths for the same setting as Fig. 9, where L_i denotes the i-th path class sorted in ascending path length. For each path class, we aggregate the traffic carried by all paths in that class and normalize its traffic share by the share under uniform packet spraying. Therefore, a normalized value larger than 1 indicates that the path class carries more traffic than uniform allocation. The results show that LAPS performs latency-aware packet spraying across all

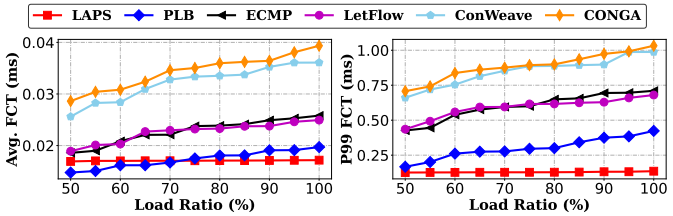


Figure 17: The FCTs on (Dragonfly, STR, All2all).

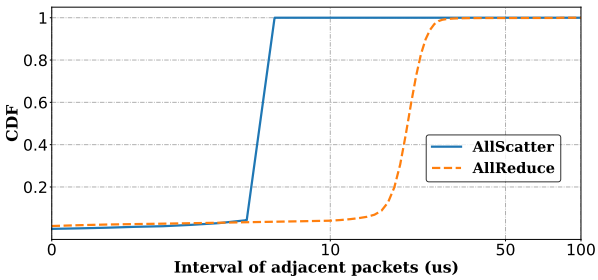


Figure 18: The distribution of intervals of adjacent packets in LLM training on Dragonfly.

load ratios from 50% to 100%. First, LAPS assigns larger traffic shares to shorter paths, which is consistent with its objective of reducing FCT. Second, LAPS does not concentrate all traffic on the shortest paths. Instead, it still allocates part of the traffic to longer paths when their real-time latency is competitive. This behavior indicates that LAPS can utilize multiple candidate paths at runtime and avoid the herd effect.

Lost Retransmission: We test the algorithms in the presence of packet loss by setting a 0.0001% packet corruption probability per link. As shown in Table III and Table IV, LAPS achieves the best average FCT and P99 FCT. More importantly, it demonstrates that LAPS's average FCT and P99 FCT are almost unaffected by random packet loss. In contrast, the average FCT and P99 FCT of other algorithms increase under the same condition. For instance, the average FCTs for ECMP, LetFlow, Conga, ConWeave, and PLB increase by 16, 25, 7, 47, and 8 ms, respectively. This is because LAPS can distinguish truly lost packets from OOO packets so fast selective retransmission is possible.

Buffer Size. When LAPS is used, Fig. 20 shows the statistics of the buffer size due to OOO packets at the receiver for the LLM training on Rail networks. As shown in Fig. 20, although LAPS employs packet spraying, it achieves a low overall packet reordering ratio. For the AllScatter operations when no congestion is experienced, only 0.2% packets are found to be reordered and the buffer size is only 0.003KB on average. Even for the AllReduce operations that trigger severe congestion, LAPS requires buffering fewer than 16 1KB packets on average, with only less than 4% probability that the buffer size exceeds 30KB. This indicates that LAPS has low pressure on the receiver's OOO packet buffer and reordering process. Considering the scenarios such as LLM training for which the number of flows is relatively small, the current commercial RDMA NICs are sufficient to store and reorder the OOO packets.

Table III: FCTs on (Rail, LLM, AllReduce)

Drop ratio	0						0.0001%					
Scheme	ECMP	LetFlow	Conga	ConWeave	PLB	LAPS	ECMP	LetFlow	Conga	ConWeave	PLB	LAPS
Avg. FCT (ms)	564	553	540	563	211	162	580	578	547	610	219	162
P99 FCT(ms)	617	589	628	659	235	175	645	633	658	757	240	175

Table IV: FCTs on (Dragonfly, LLM, AllReduce)

Drop ratio	0						0.0001%					
Scheme	ECMP	LetFlow	Conga	ConWeave	PLB	LAPS	ECMP	LetFlow	Conga	ConWeave	PLB	LAPS
Avg. FCT (ms)	1337	1328	1373	1350	661	537	1340	1334	1373	1363	661	538
P99 FCT (ms)	1423	1420	1428	1418	690	574	1427	1427	1423	1424	690	575

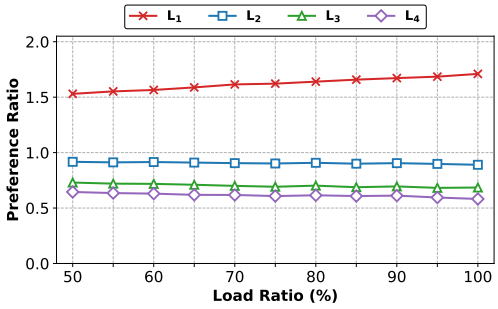


Figure 19: Traffic distribution across unequal-cost paths.

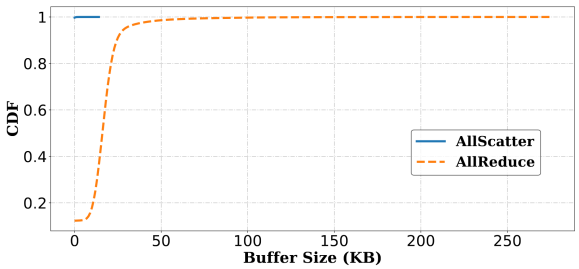


Figure 20: The buffer size in LLM training on Rail.

C. Hardware Prototype Analysis

We implement a prototype of LAPS using 1,129 lines of code in Chisel 5.0 (which maps to 5,992 lines of code in SystemVerilog) on a Xilinx Alveo U280 Accelerator Card. The PST table in the prototype has a size of 1,024 to support up to 1,024 servers, with each `PST` entry containing 8 valid `pids`. Correspondingly, the `PIT` table in the prototype has a size of 1,024×8=8,192, with each `PIT` entry allowing 4 anchors. The rate limiting module can support up to 1,024 flows. When evaluating the resource utilization, we excluded the network interface card functions themselves (Xilinx 100G CMAC IP, queues, schedulers, etc.) and only included the LAPS parsing/deparsing modules and the corresponding processing pipeline. We separately evaluated the resources consumed by the `PST`, `PIT`, rate limiting module, and Softmax, as shown in Table V.

Table V: Hardware consumption on FPGA

	LUT	**FF**	**CARRY8**	**BRAM**
`PST` Table	74	177	0	8
`PIT` Table	2950	2976	0	112
Rate Limiting Tables	3012	3866	119	8
Softmax Function	26949	7938	1752	0
Total	33808 (2.51%)	16711 (0.64%)	1909 (1.17%)	135.5 (6.72%)

The synthesis results indicate that the resource consumption of LAPS for the FPGA on U280 is acceptable, and it is feasible to deploy LAPS on an FPGA-based SmartNIC. As a future work, we will implement LAPS on multi-core-processor-based SmartNICs which would provide better scalability and extensibility.

V. Related Work

Previous works on multi-path LB are mainly for topologies with equal-cost paths. The approaches can be classified based on the primary deployment location as the first dimension and on the granularity as the second dimension.

Switch-based: ECMP and WCMP [52] distribute flows to different links. Flare [3], LetFlow [50], LocalFlow [53], and BurstBalancer [51] spray segments of flows (e.g., flowlet) across links. RPS and Drill [54] spray packets to different links. CONGA [55], Hula [56], and Proteus [39], [57] distributes flowlets or flows according to the RTT or link utilization.

Controller-based: Hedera [58], Mahout [59], Freeway [60], DRLet [61], and Fastpass [62] determine the paths for flows or packets based on exposed network and application information. Relying on a central controller, such solutions cannot scale to large networks and react to micro-bursts [63].

Host-based: MPTCP [64], Flowbender [65], Presto [6], and Clove [66] modify the transport layer to split a single flow into subflows for transmission on different paths. PLB [37] changes the flow label of any congested flow in hope that the in-network ECMP may change its path.

To mitigate the packet reordering issue for packet spraying [7], SRED selectively drops packets that would lead to unequal queue lengths [67], and QDAPS and QDAPS* [68] ensure a packet has longer queuing delay than the previous packet of the same flow. These local solutions do not work for unequal-cost paths. With added complexity, CAPS [69], HTPC [70], and Corrective [71] introduce a coding layer between TCP and IP, and spray the coded packets to address the reorder problem.

Adaptive routing can be applied to utilize the unequal-cost paths. UGAL-L [13] always sprays packets to the port with the smallest queue length. UGAL-G [13] uses the queue length of other switches and hop count to estimate the path latency. PAR [72] only uses the minimal path but allows intermediate switches to reroute to avoid congestion. Q-adaptive [73] adopts reinforcement learning technique to predict global path

conditions based on local information.

DCTCP [20] and DCQCN [21] are widely deployed CC algorithms in DCN. Many others are used in wide-area networks [74], [75]. CC can be achieved with or without the assist of network switches, based on ECN, RTT, or other signals, and using window or credit-based mechanisms. In recent years, many improvements to DCTCP and DCQCN are proposed [76], [47]. However, all these works consider CC independent of LB, making them ill-suited for the unequal-cost multi-path networks. STrack [77] is a joint LB/CC algorithm for packet spraying on equal-cost multi-path networks using ECN and RTT.

VI. Conclusion

LAPS is a joint load-balancing and congestion-control algorithm to support unequal-cost multi-path packet spraying on arbitrary network topologies and for any traffic patterns, catering to HPC/AI workloads in multi-tenant data center networks. Although simple, LAPS occupies a unique niche in the wide spectrum of load-balancing algorithms: it is a host-based stateful packet-to-path load balancing scheme relying on real-time global information. Moreover, it reveals that the congestion-control and loss-recovery algorithms need changes to fit in the new scenario, and LAPS provides a practical solution with synergistic benefits.

References

[1] Y. Wan, H. Song, Y. Jia, Y. Yang, T. Huang, and Z. Chen. LAPS: Joint Load Balancing and Congestion Control on Unequal-cost Multi-path Data Center Networks. In *Proceedings of the 2nd Workshop on Networks for AI Computing*, pages 11–18, 2025.

[2] C. Hopps. RFC2992: Analysis of an Equal-Cost Multi-Path Algorithm. Technical report, RFC Editor, 2000.

[3] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review*, 37(2):51–62, 3 2007.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[5] M. Alizadeh and T. Edsall. On the Data Path Performance of Leaf-Spine Datacenter Fabrics. In *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pages 71–74, San Jose, CA, USA, 8 2013. IEEE.

[6] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella. Presto: Edge-based Load Balancing for Fast Datacenter Networks. *ACM SIGCOMM Computer Communication Review*, 45(4):465–478, 9 2015.

[7] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the Impact of Packet Spraying in Data Center Networks. In *2013 Proceedings IEEE INFOCOM*, pages 2130–2138, Turin, Italy, 4 2013. IEEE.

[8] W. Li, X. Liu, Y. Li, Y. Jin, H. Tian, et al. Understanding Communication Characteristics of Distributed Training. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, APNet '24, pages 1–8, New York, NY, USA, 2024. Association for Computing Machinery.

[9] S. Rajasekaran, M. Ghobadi, and A. Akella. CASSINI: Network-Aware Job Scheduling in Machine Learning Clusters. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1403–1420, Santa Clara, CA, 4 2024. USENIX Association.

[10] W. Wang, M. Ghobadi, K. Shakeri, Y. Zhang, and N. Hasani. Rail-only: A Low-Cost High-Performance Network for Training LLMs with Trillion Parameters. 2024.

[11] UEC. UEC Progresses Towards v1.0 Set of Specifications, 3 2024.

[12] N. P. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, et al. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings, 2023.

[13] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. *ACM SIGARCH Computer Architecture News*, 36(3):77–88, 6 2008.

[14] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking Data Centers Randomly. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 225–238, San Jose, CA, 4 2012. USENIX Association.

[15] H. Liao, B. Liu, X. Chen, Z. Guo, C. Cheng, et al. Ub-mesh: a hierarchically localized nd-fullmesh datacenter network architecture. *IEEE Micro*, 2025.

[16] Nvidia. NVLink and NVLink Switch, 2024.

[17] UALink Consortium. Ultra Accelerator Link, 2024.

[18] IBTA. Infiniband trade association, 2024.

[19] S. Yeluri. Should UEC and UAL Merge?, 2024.

[20] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, et al. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010.

[21] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, et al. Congestion control for large-scale rdma deployments. In *Proceedings of the 2015 ACM SIGCOMM Conference*, 2015.

[22] Y. Lu, G. Chen, B. Li, K. Tan, Y. Xiong, et al. Multi-Path Transport for RDMA in Datacenters. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 357–371, Renton, WA, 4 2018. USENIX Association.

[23] G. Chen, Y. Lu, B. Li, K. Tan, Y. Xiong, et al. MP-RDMA: Enabling RDMA With Multi-Path Transport in Datacenters. *IEEE/ACM Transactions on Networking*, 27(6):2308–2323, 12 2019.

[24] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, et al. Revisiting Network Support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 313–326, New York, NY, USA, 8 2018. ACM.

[25] J. Yen. Finding the k shortest loopless paths in a network. *Management Science 17(11)*, 1971.

[26] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, et al. In-band Network Telemetry: A Survey. *Computer Networks*, 186:107763, 2021.

[27] Broadcom. In-band Telemetry, 2017.

[28] X. Yang, L. Zeng, Z. Liu, Y. Chen, G. Lv, et al. FastRMT: A high-speed data plane programmable system for micro-architecture innovation. *Chinese Journal of Computers*, 47(2):473–490, 2 2024.

[29] A. Forencich, A. C. Snoeren, G. Porter, and G. Papen. Corundum: An open-source 100-gbps nic. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 38–46, 2020.

[30] D. Bansal, G. DeGrace, R. Tewari, M. Zygmunt, J. Grantham, et al. Disaggregating stateful network functions. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1469–1487, 2023.

[31] N. Sundar, B. Burres, Y. Li, D. Minturn, B. Johnson, and N. Jain. 9.4 an in-depth look at the intel ipu e2000. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 162–164. IEEE, 2023.

[32] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[33] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, et al. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 15–28, 2016.

[34] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[35] Q. Sun, Z. Di, Z. Lv, F. Song, Q. Xiang, et al. A high speed softmax vlsi architecture based on basic-split. In *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 1–3, 2018.

[36] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[37] M. A. Qureshi, Y. Cheng, Q. Yin, Q. Fu, G. Kumar, et al. PLB: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, pages 207–218, New York, NY, USA, 2022. Association for Computing Machinery.

[38] C. H. Song, X. Z. Khooi, R. Joshi, I. Choi, J. Li, and M. C. Chan. Conweave: Network load balancing with in-network reordering support for rdma. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 816–831. ACM, 9 2023.

[39] J. Zhang, W. Bai, and K. Chen. Enabling ECN for datacenter networks with RTT variations. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*,

CoNEXT '19, pages 233–245, New York, NY, USA, 2019. Association for Computing Machinery.

[40] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury. Resilient Datacenter Load Balancing in the Wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 253–266, New York, NY, USA, 2017. Association for Computing Machinery.

[41] Google. Tcp-plb open source, 2024.

[42] Xilinx. Alveo U280 Data Center Accelerator Card. https://www.avnet.com/opasdata/d120001/medias/docus/196/XLX-A-U280-A32G-DEV-G-Datasheet.pdf, 2024.

[43] Y. Wan. Laps open source. https://github.com/wany16/Laps-ns3, 2025.

[44] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, et al. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[45] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, et al. VL2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[46] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 221–235, New York, NY, USA, 2018. Association for Computing Machinery.

[47] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 44–58. ACM, 8 2019.

[48] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, et al. Llama: Open and efficient foundation language models. *ArXiv*, 2023.

[49] I. T. Association et al. Infinibandtm architecture specification volume 1 release 1.3 (general specifications), 2015.

[50] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 407–420, Boston, MA, 3 2017. USENIX Association.

[51] Z. Liu, Y. Zhao, Z. Fan, T. Yang, X. Li, et al. BurstBalancer: Do Less, Better Balance for Large-Scale Data Center Traffic. *IEEE Transactions on Parallel and Distributed Systems*, 35(6):932–949, 2024.

[52] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, et al. WCMP: weighted cost multipathing for improved fairness in data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, pages 1–4, New York, NY, USA, 2014. Association for Computing Machinery.

[53] S. Sen, D. Shue, S. Ihm, and M. J. Freedman. Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 151–162, New York, NY, USA, 2013. Association for Computing Machinery.

[54] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian. DRILL: Micro Load Balancing for Low-latency Data Center Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 225–238, New York, NY, USA, 2017. ACM.

[55] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, et al. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 503–514, New York, NY, USA, 8 2014. ACM.

[56] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. HULA: Scalable Load Balancing Using Programmable Data Planes. In *Proceedings of the Symposium on SDN Research*, SOSR '16, page 1, New York, NY, USA, 2016. Association for Computing Machinery.

[57] J. Hu, C. Zeng, Z. Wang, J. Zhang, K. Guo, et al. Load Balancing With Multi-Level Signals for Lossless Datacenter Networks. *IEEE/ACM Transactions on Networking*, 32(3):2736–2748, 2024.

[58] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 19, USA, 2010. USENIX Association.

[59] A. R. Curtis, W. Kim, and P. Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings IEEE INFOCOM*, pages 1629–1637, Shanghai, China, 2011. IEEE.

[60] W. Wang, Y. Sun, K. Zheng, M. A. Kaafar, D. Li, and Z. Li. Freeway: Adaptively Isolating the Elephant and Mice Flows on Different Transmission Paths. In *2014 IEEE 22nd International Conference on Network Protocols*, pages 362–367, Raleigh, NC, USA, 2014. IEEE.

[61] X. Diao, H. Gu, W. Wei, G. Jiang, and B. Li. Deep Reinforcement Learning Based Dynamic Flowlet Switching for DCN. *IEEE Transactions on Cloud Computing*, 12(2):580–593, 2024.

[62] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: a centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 307–318, New York, NY, USA, 2014. Association for Computing Machinery.

[63] Y. Wang, D. Li, Y. Lu, J. Wu, H. Shao, and Y. Wang. Elixir: A High-performance and Low-cost Approach to Managing Hardware/Software Hybrid Flow Tables Considering Flow Burstiness. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 535–550, Renton, WA, 4 2022. USENIX Association.

[64] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, pages 1–14, Boston, MA, 3 2011. USENIX Association.

[65] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene. FlowBender: Flow-level Adaptive Routing for Improved Latency and Throughput in Datacenter Networks. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 149–160, New York, NY, USA, 12 2014. ACM.

[66] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, et al. Clove: Congestion-Aware Load Balancing at the Virtual Edge. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, pages 323–335, New York, NY, USA, 2017. Association for Computing Machinery.

[67] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[68] J. Huang, W. Lyu, W. Li, J. Wang, and T. He. Mitigating Packet Reordering for Random Packet Spraying in Data Center Networks. *IEEE/ACM Transactions on Networking*, 29(3):1183–1196, 2021.

[69] J. Hu, J. Huang, W. Lv, Y. Zhou, J. Wang, and T. He. CAPS: Coding-Based Adaptive Packet Spraying to Reduce Flow Completion Time in Data Center. *IEEE/ACM Transactions on Networking*, 27(6):2338–2353, 2019.

[70] J. Huang, S. Wang, S. Li, S. Zou, J. Hu, and J. Wang. HTPC: heterogeneous traffic-aware partition coding for random packet spraying in data center networks. *Journal of Cloud Computing*, 10(1):31, 2021.

[71] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, et al. Reducing Web Latency: the Virtue of Gentle Aggression. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 159–170, New York, NY, USA, 8 2013. ACM.

[72] N. Jiang, J. Kim, and W. J. Dally. Indirect adaptive routing on large scale interconnection networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 220–231, New York, NY, USA, 2009. Association for Computing Machinery.

[73] Y. Kang, X. Wang, and Z. Lan. Q-adaptive: A Multi-Agent Reinforcement Learning Based Routing on Dragonfly Network. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '21, pages 189–200, New York, NY, USA, 2021. Association for Computing Machinery.

[74] V. Arun and H. Balakrishnan. Copa: Practical Delay-Based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 329–342, Renton, WA, April 2018. USENIX Association.

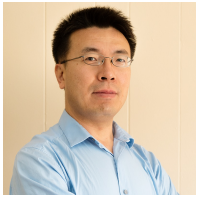
[75] G. Zeng, W. Bai, G. Chen, K. Chen, D. Han, et al. Congestion control for cross-datacenter networks. *IEEE/ACM Transactions on Networking*, 30(5):2074–2089, 2022.

[76] G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, et al. Swift: Delay is simple and effective for congestion control in the datacenter. SIGCOMM '20, page 514–528, New York, NY, USA, 2020. Association for Computing Machinery.

[77] Y. Le, R. Pan, P. Newman, J. Blendin, A. Kabbani, et al. Strack: A reliable multipath transport for ai/ml clusters, 2024.

Ying Wan received the B.S. degree in communication engineering from Northwestern Polytechnical University, Xi'an, China, in 2016, and the Ph.D. degree in computer science and engineering from Tsinghua University, Beijing, China, in 2022. He is now an associate professor with the School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China. His research interests include high-performance switches/routers, software-defined networks, computing power network, and data center networks.

Jinyu Xiao received the B.S. degree in information and computing science from Hunan University of Science and Technology, Xiangtan, China. He is currently pursuing the M.E. degree in the School of Cyber Science and Engineering, Southeast University, Nanjing, China, under the guidance of Dr. Ying Wan. His research focuses on traffic load balancing in data center networks.



Haoyu Song (Senior Member, IEEE) received the BE degree in electronics engineering from Tsinghua University, in 1997, and the MS and DSc degrees in computer engineering from Washington University in St. Louis in 2003 and 2006, respectively. He is a senior principal network architect with Futurewei Technologies, USA. His research interests include software defined network, network virtualization and cloud computing, high performance networked systems, algorithms for network packet processing and intrusion detection.

Zhikang Chen received Bachelor of Engineering degree at Department of Computer Science and Technology, Tsinghua University, Beijing. He is currently pursuing the Ph.D. degree in Computer Science with Tsinghua University, Beijing, China, under the guidance of Dr. Bin Liu. His research focuses on high-performance data plane architectures, packet classification, and packet scheduling.

Yu Jia received the BE degree in Sichuan University in 2012. He is a senior principal network architect with China Mobile (Suzhou) Software Technology Co., Ltd, China. His research interests include Future Network Architectures, Software-defined Networks, and Quantum Computing.

Yunhui Yang received the BE degree from Nanchang University in 2019 and the ME degree from Central South University in 2022. He is an R&D engineer with ChinaMobile (Suzhou) Software Technology Co., Ltd., China. His research interests include Software-defined Networks, and Quantum Computing.

Bin Liu (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer science and engineering from Northwestern Polytechnical University, Xi'an, China, in 1988 and 1993, respectively. He is now a Full Professor with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His current research areas include high-performance switches/routers, network processors, and greening the Internet. He has received numerous awards from China, including the Distinguished Young Scholar of China and won the inaugural Applied Network Research Prize sponsored by ISOC and IRTF in 2011.

Tao Huang (Senior Member, IEEE) received the Ph.D. degree in communication and information systems from Beijing University of Posts and Telecommunications in 2007. He is currently the Director of the Future Networks Research Center, Purple Mountain Laboratories, Nanjing, China. His current research interests include network architecture, computing power network, and network for AI.