

CAVER: Enhancing RDMA Load Balancing by Hunting Less-Congested Paths

Haotian Deng
Tsinghua University
Beijing, China
denght23@mails.tsinghua.edu.cn

Jue Zhang
Beihang University
Beijing, China
zhangjue1@buaa.edu.cn

Menghao Zhang
Beihang University
Beijing, China
zhangmenghao0503@gmail.com

Yuan Yang
Tsinghua University
Beijing, China
yangyuan_thu@mail.tsinghua.edu.cn

Xiao Li
Tsinghua University
Beijing, China
li-x23@mails.tsinghua.edu.cn

Junhao Dong
Tsinghua University
Beijing, China
dongjh21@mails.tsinghua.edu.cn

Yi Liu
Tsinghua University
Beijing, China
liuy22@mails.tsinghua.edu.cn

Mingwei Xu
Tsinghua University
Beijing, China
xumw@tsinghua.edu.cn

Abstract

Load balancing (LB) is critical for reducing flow completion time (FCT) in RDMA networks where link congestion fluctuates rapidly. While many existing works have been devoted to improving LB performance in RDMA, they all select paths based on limited information, leaving a large number of available paths unexploited, which is prone to wasting the network bandwidth resources. In this paper, we aim to enhance existing works by optimizing the awareness of congestion status and propose CAVER, an RDMA-optimized LB framework. The key idea behind CAVER is to leverage ACK packets to establish a vector-protocol-like mechanism, enabling the efficient collection of real-time path congestion information and the selection of less-congested paths. We implement CAVER with NS3 and Tofino switches, and conduct simulations and real experiments. The results show that compared to state-of-the-art, CAVER improves average and 99th-percentile FCT by up to 46.0% and 32.2%, respectively.

CCS Concepts: • Networks → Network protocols; Data center networks.

Keywords: RDMA, load balancing, data center networks

ACM Reference Format:

Haotian Deng, Jue Zhang, Menghao Zhang, Yuan Yang, Xiao Li, Junhao Dong, Yi Liu, Mingwei Xu. 2026. CAVER: Enhancing RDMA Load Balancing by Hunting Less-Congested Paths. In *The 2026 ACM SIGOPS Annual Technical Conference (SIGOPS ATC 2026)*, November



This work is licensed under a Creative Commons Attribution 4.0 International License.

SIGOPS ATC 2026, Hong Kong, China

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2818-1/2026/11

<https://doi.org/10.1145/3828523.3857047>

15–18, 2026, Hong Kong, China. ACM, New York, NY, USA, 16 pages.
<https://doi.org/10.1145/3828523.3857047>

1 Introduction

Remote Direct Memory Access (RDMA) allows a network client to directly access the memory of a remote server without involving its CPU. By offloading I/O responsibilities from CPU to RDMA-enabled network interface cards (RNICs), RDMA bypasses the Linux network stacks and enables zero-copy networking, achieving ultra-high throughput and ultra-low latency. Many data centers (DCs) have adopted RDMA over Converged Ethernet v2 (RoCEv2) [11] to provide superior performance for emerging application paradigms such as cloud storage [5, 10] and distributed LLM training [9, 19, 42].

Modern DCs are designed with multiple equal-cost paths for each pair of servers to meet the ever-increasing data exchange requirements [2]. To fully utilize the network bandwidth, load balancing (LB) plays an important role by spreading flows among these paths. A widely used LB scheme, Equal Cost Multi-Path (ECMP), randomly stripes flows across available paths using flow hashing [25]. Despite its wide deployment, ECMP is known to have limited performance due to hash collisions and obliviousness to congestion [27, 46, 50].

Both industry and academia have made great progress recently in improving load balancing performance for RDMA [9, 12–14, 17, 26, 29, 46–48, 50]. Some solutions monitor the congestion status of the paths used by existing flows and may replace or remove certain paths [13, 14, 26, 29, 47]. However, these approaches can only obtain congestion information from in-use paths while having limited visibility into unused ones. Other solutions are either oblivious to congestion or rely only on local congestion signals [9, 12, 17, 48, 50]. These strategies rely on local information and are hard to cope with asymmetries such as link failures or non-uniform

background traffic, which are common in practical DCs, as previous work has shown [3, 49]. Consequently, existing solutions lack a global view of congestion states, which can leave many available paths unexploited and bandwidth resources underutilized, leading to limited LB performance.

Maintaining a global view of congestion states for RDMA is challenging, as there are a large number of parallel paths between two hosts in DC networks. Some approaches monitor the congestion state of each path by sending probes periodically [21, 41, 52]. However, these approaches either introduce substantial overhead [21, 52], or rely on the control plane to aggregate the probed result [41], which cannot provide sufficiently timely information for RDMA due to inherent processing limitations. Note that path congestion levels in an RDMA network can change rapidly, and outdated congestion information has little practical value. In this paper, we introduce an efficient, probe-free mechanism implemented entirely in the data plane, which identifies and maintains a set of less-congested paths instead of maintaining congestion states for all links.

We propose CAVER, a Congestion-Aware LB framework for RDMA with VECtoR protocol. CAVER leverages acknowledgement (ACK) packets to efficiently identify less-congested paths for each source-destination pair. The key insight is that a large number of ACK packets traverse most (if not all) links in a RoCEv2 network at any time [16]. CAVER exploits this property by piggybacking a path and its congestion information onto ACK packets. Furthermore, ACKs from different flows can share this information via programmable switches, enabling efficient path discovery. This approach resembles vector routing protocols such as RIP [31] and BGP [43]. Thus, CAVER breaks the limitation that the real-time view of path congestion level is constrained by recent flows, offering far more options for accurate routing decisions.

Designing CAVER is non-trivial, as congestion states for numerous paths are required, yet each ACK packet can piggyback only limited information due to bandwidth and processing constraints. To address this challenge, CAVER first identifies the least-congested path and subsequently leverages this information to select paths with acceptable congestion levels. Then, we propose a “Give-and-Take” strategy executed by each switch, ensuring that each ACK packet carries only the least-congested path along with one acceptable path, while at the same time enabling efficient information sharing and dissemination. The outcome of our path information collection is twofold. First, each switch obtains the least-congested path for each outport, enabling the determination of split ratios for packet spraying. Second, each ToR switch acquires a distinct set of acceptable paths for determining the routing or rerouting path of a flow.

CAVER can be incorporated into various LB schemes. For schemes that select the outport for each packet, CAVER determines the split ratio based on the congestion level of

the least-congested path for each outport, rather than relying on local information such as queue length. Thus, LB performance in handling network asymmetries can be significantly improved. For schemes that determine the routing or rerouting path for a flow or subflow, CAVER adopts a conflict-free path selection approach to prevent multiple flows from choosing the same path, especially when flows arrive concurrently.

We realize CAVER using the NS3 simulator [37], and also implement a prototype with Tofino switches [36]. The source code has been released on GitHub [7]. Evaluation results show that CAVER improves the average and 99th-percentile Flow Completion Time (FCT) by up to 46.0% and 32.2%, respectively, compared to state-of-the-art (SOTA). In particular, CAVER achieves a performance improvement of up to 55.6% over SOTA when handling flows with high concurrency. Furthermore, CAVER delivers a performance improvement of 19.2% compared to ECMP in partially deployed scenarios. Moreover, CAVER is resource-efficient, requiring less than one-third of the computational and memory resources of a low-end Tofino switch to handle a 10k-host network.

This work does not raise any ethical issues.

2 Background and Motivation

2.1 Existing LB Schemes for RDMA

RDMA load balancing is fundamentally constrained by RNIC hardware, motivating numerous studies on LB within these constraints. Some solutions [9, 17, 48] partition large flows into smaller messages by posting separate transport requests at the application layer, and distribute the messages across the QPs. Such QP-scaling-based methods often rely on a round-robin scheme to select a path for each subflow, because the flow splitting runs in the application layer and path congestion information for each QP is not maintained. ConWeave [47], Proteus [14], and FLB [13] propose carefully rerouting with switches’ support. FLB and Proteus limit the difference in RTT between paths for adjacent packets to be smaller than the sending time gap. ConWeave restricts on-the-fly packets of a single flow to at most two different paths and performs reordering at ToR switches to ensure in-order arrival. However, these approaches can only acquire congestion information of the paths used by existing flows and have limited information about the unused paths. On the other hand, MP-RDMA [29] and STrack [26] propose multipath transport protocols for RDMA by modifying RNICs hardware. They monitor congestion across the candidate paths of a flow and prune slower paths from subsequent transmissions, thereby reducing FCTs. These approaches use randomly designated paths and do not seek new paths that are less congested. Recently, some solutions propose to apply packet spraying to RDMA, making independent routing

decisions on a per-packet basis. For example, STELLAR [50] adopts oblivious packet spraying without congestion awareness, while DCP [27] employs adaptive routing that selects the least-loaded output port at each switch. However, as noted in prior load-balancing studies, both approaches lack visibility into downstream path conditions and therefore suffer under asymmetric topologies or traffic.

2.2 Collecting Path Information

Wide and accurate visibility into path congestion can help improve routing decisions by routing flows toward less-congested paths, thereby better utilizing available bandwidth. However, efficiently collecting such information is non-trivial. The main challenges are: 1) The large number of paths between each source and destination: For example, in a 3-layer CLOS topology with ($k=48$) [2], there are 576 equal-cost paths for each source-destination pair. Collecting and recording such a vast amount of path information for a single destination is burdensome, especially considering the approximately 20,000 potential destination hosts in such a topology. 2) The need for high collection frequency: Flows in data centers are highly dynamic, particularly for RDMA traffic, which is initially sent at line speed. Congestion on each path can fluctuate significantly within a single Round-Trip Time (RTT), necessitating frequent, real-time information collection. Otherwise, outdated information can become useless or even misleading, and routing based on inaccurate information can even be worse than random decisions [35, 49].

Existing methods for collecting path congestion information struggle to meet both of these challenges simultaneously. Some approaches passively collect path congestion via in-flight packets. ConWeave [47] and HPCC [28] estimate congestion by measuring a flow’s RTT at either switches or hosts, while other works [3, 14, 20, 21] piggyback link utilization metrics onto data packets. Proteus [14] combines both metrics to improve accuracy. However, these methods share a common limitation: each packet can only observe congestion along the path it traverses. In RDMA, packets within a flow typically follow a single path due to the difficulty of efficient rerouting, so these packets all reflect congestion on a single path, regardless of how many packets are transmitted. This significantly limits the efficiency of path information collection. Another way is to actively send dedicated probes to collect path congestion. HULA [21] implements a distance vector protocol to identify the best path between a ToR switch and each destination, while Hermes [52] periodically probes random paths to sample congestion. However, when both high collection frequency and a large path scope are required, these methods introduce a significant number of probe packets and impose substantial bandwidth overhead. SGLB [41] introduces a link-state protocol to collect real-time congestion information with modest bandwidth overhead, but relies on switch control-plane computation,

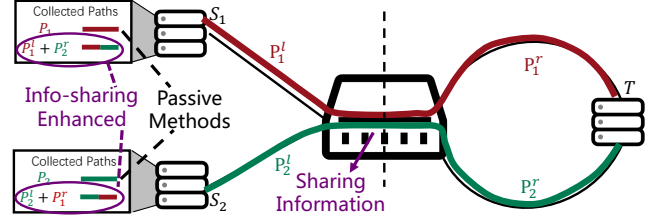


Figure 1. Example of path information sharing.

which struggles to track fast-changing congestion, and the control-to-data-plane bandwidth can limit scalability.

2.3 Information Sharing Strategy

Compared to active collection methods, passive path collection methods leverage existing in-flight data packets, thereby introducing negligible overhead. Given that a large volume of packets is always in flight and present on almost every link, these packets can naturally provide rich, real-time path information. However, current passive collection methods fail to fully exploit this potential. Since multiple packets from the same flow provide no additional information beyond that flow’s path, significant room remains to improve collection efficiency. We attribute this inefficiency to the lack of leveraging shareable information across flows. Specifically, packets from different flows can exchange congestion information, revealing paths that would otherwise remain undetected. By allowing a packet to carry congestion information for paths other than the one that the packet traverses, we can significantly enhance path visibility with negligible overhead.

Figure 1 illustrates the inefficiency of current passive path collection methods with a simple example. Consider two flows in the network: a flow from S_1 to T , where the red line represents the path of ACK packets, P_1 (i.e., $P_1^l + P_1^r$); and a flow from S_2 to T , where the green line represents the path of ACK packets, P_2 (i.e., $P_2^l + P_2^r$). Each ACK packet carries congestion information for the path it traverses. Consequently, S_1 and S_2 can only collect congestion information for their respective single paths, P_1 and P_2 . Fortunately, we observe that the congestion information collected by different flows is shareable, and one flow can leverage the information collected by another flow to infer additional path congestion information. Specifically, if the information of P_1^l & P_2^r is stored in the switch and carried by the ACK packet traversing P_1 , then the ACK packet can combine the congestion information of P_2^r with that of P_1^l , helping S_1 to infer the congestion status of a new path, $(P_1^l + P_2^r)$, even though no ACK packets have traversed this path directly. Leveraging this approach, S_1 and S_2 can collect congestion information for multiple paths. As the number of flows in the network increases, the potential for such combinations grows exponentially, enabling the discovery of more comprehensive path congestion information.

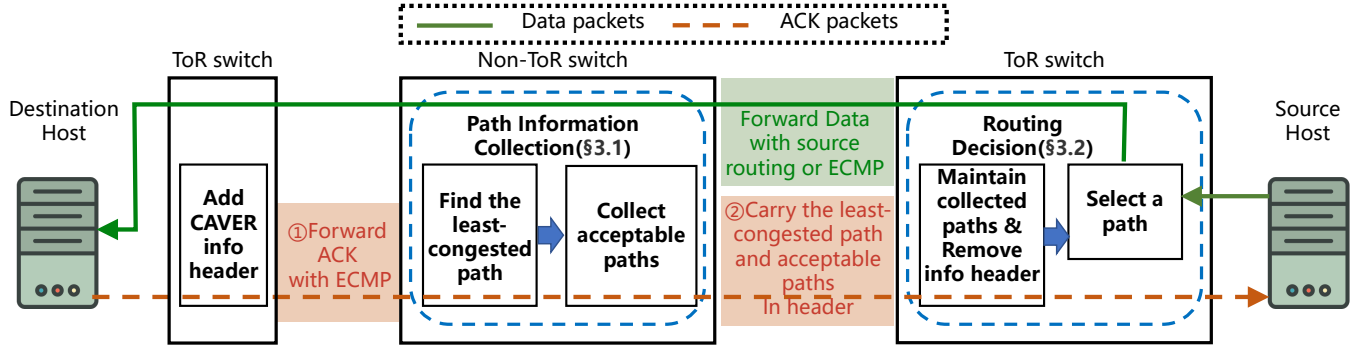


Figure 2. Overview of CAVER with flow path selection.

2.4 Collecting Acceptable Paths

Next, we decide which path information each switch should collect with the information sharing strategy. One straightforward approach is that each switch only collects the least-congested path for each destination, as done in HULA [21]. While the least-congested path may be optimal for a single flow or packet, this approach can perform poorly when handling concurrent flows arriving in a short period. Due to the inherent transmission delay of information, network congestion states may change within a short time but may not be promptly captured by switches. As a result, the currently recorded “least-congested” path on the switches may remain unchanged for a short period. Thus, multiple flows arriving in this period may all be routed to the same least-congested path, leading to conflicts.

Considering that concurrency is common in data center traffic, instead of routing based solely on the least-congested paths, we propose routing using “less-congested paths”, i.e., paths with congestion levels within an acceptable range of the least-congested path, which we refer to as acceptable paths. In this way, flows can avoid congested paths, and the chances of conflict are reduced. However, this requires ensuring two things when collecting path congestion information: 1) we should accurately collect the information of the least-congested path to avoid incorrectly identifying congested paths as acceptable ones, ensuring that flows are routed to relatively idle paths; and 2) we should collect as many truly acceptable paths as possible to provide enough routing options, helping to avoid conflicts.

3 CAVER Design

We propose CAVER, a new load-balancing framework to make smarter routing decisions for RDMA traffic. CAVER collects near real-time congestion information which can be incorporated into various LB schemes. For schemes that select the output port for each packet at each switch, CAVER determines the split ratio based on the remote congestion level of the least-congested path for each output. For schemes that determine the routing or rerouting path for a flow, CAVER

provides information about less-congested paths (i.e., acceptable paths) in the network and selects one smartly from these paths for each flow¹. Figure 2 depicts the overview of CAVER with flow path selection, which consists of two components: Path Information Collection and Routing Decision.

The Path Information Collection component, located in each non-ToR switch, combines congestion information from different pass-by flows to identify the least-congested path and acceptable paths to each destination. This component comprises four tables: 1) DRETable, storing the congestion extent (CE) by portID; 2) BestTable, mapping host IP address to the least-congested path; 3) RemoteCETable, mapping (host IP address, portID) pairs to CE of the least-congested path for this output; and 4) GoodTable, mapping host IP address to acceptable paths. This component utilizes the ACK packets to piggyback the information from the BestTable and the GoodTable to the next switch. Upon receiving an ACK packet from an ingress port, a switch combines the congestion information in the packet with its own port data from the DRETable to identify the least-congested and acceptable paths. This process continues along the path of an ACK packet until the ToR switch finally receives the path information. In summary, the Path Information Collection component aggregates information received from different switches to update and maintain the information stored in the BestTable, the RemoteCETable, and the GoodTable, and collects a set of candidate acceptable paths for ToR switches.

The Routing Decision component supports different LB schemes. For schemes that select the output port for each packet at each switch, this component resides on both ToR and non-ToR switches. It determines the split ratio based on the collected local and remote congestion information. This component on ToR switches comprises the DRETable and RemoteCETable discussed previously. For schemes that determine the routing or rerouting path for a flow, this component resides in ToR switches and aggregates information about the least-congested and acceptable paths from other switches

¹We present flow-level routing below; CAVER generalizes to flowlet-level and subflow-level routing with the same workflow (§ 3.2 and § 4.2).

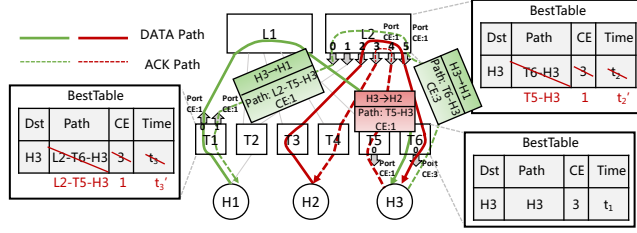


Figure 3. Example of finding the least congested path. The green and red blocks represent ACK packets.

to identify real-time routing options to each destination. It selects paths for passing flows from these acceptable paths, and uses two extra tables besides the DRETable and BestTable discussed previously: 1) PathTable, storing up to K acceptable paths for each host; and 2) FlowTable, storing the routing decisions for each flow (identified by flowId). When an ACK packet with path information is received, the component updates the BestTable with the least-congested path, compares the piggybacked acceptable paths, and adds them to the PathTable. For new flows, an acceptable path is selected from the PathTable, or ECMP is used if none are available. Once a routing decision is made, it is stored in the FlowTable, ensuring that all subsequent packets in the flow use the same path.

3.1 Path Information Collection

Unlike existing approaches that only advertise the least-congested path [3, 21], CAVER collects congestion information for multiple paths to improve routing decisions. The key challenge lies in deciding which paths are “acceptable” and should be carried. To this end, we first propose to estimate the CE of a switch port, and leverage a vector protocol-like method to identify the current least-congested path and its CE. Then, we use the current best CE to filter out the paths that are seriously congested. Finally, we propose a *Give-and-Take* strategy to spread path information efficiently.

Estimating CE for each switch port. Similar to CONGA [3], CAVER leverages the 32-bit Discounting Rate Estimator (DRE) at each switch port to estimate its CE value. The DRE is increased by the size of each transmitted data packet and periodically decayed by a multiplicative factor α : $DRE \leftarrow DRE \times (1 - \alpha)$. Let $W = T_{DRE}/\alpha$, where T_{DRE} is the decay interval. The DRE value can then be approximated as $R \times W$, where R is the average traffic rate over window W . We compute $CE = \frac{DRE}{C \cdot W} \times 255$, where C is the link data rate. Thus, CE represents the egress-link utilization on an 8-bit linear scale from 0 to 255.

Maintaining and updating least-congested paths. With each port’s CE value, the next key problem is to obtain the least-congested paths for each destination. To achieve efficient information sharing across different flows, as shown in Figure 3, CAVER designs a BestTable in each switch to store

the current least-congested path information. Each entry in the table maps a destination to three fields: 1) *Path* (32-bit), storing the least-congested path, 2) *CE* (8-bit), storing the CE value of this path, 3) *Time* (32-bit), storing the timestamp of the last update of this entry. Note that the path stores the chain of outgoing ports to a destination, with each port represented by 8 bits, allowing for up to 255 ports. In a 3-layer topology, such as fat-tree, a maximum of 4 hops is sufficient (with no need to store the last hop). For topologies with more hops, the path length can be expanded accordingly.

In CAVER, each ACK packet piggybacks the current least-congested path and corresponding CE value (in fields *BestPath* and *BestCE*). ACK packets from different flows to the same destination cooperate to compute the least-congested path to the destination via a vector-protocol-like mechanism. When an ACK packet passes a switch, it updates the BestTable based on the piggybacked information to maintain the current least-congested path information. We use the example in Figure 3 to illustrate the process. We first consider the green flow. When an ACK packet arrives at destination ToR switch T6, T6 records in the BestTable that the least-congested path to destination H3 is through port 0 with CE 3 and timestamp t_1 . Then, the ACK packet is forwarded through the network, with each switch (L2, T1, etc.) adding itself to the path and updating the BestTable if the path remains the least congested. If the CE value changes, the switch computes the maximum CE value along the path and updates the BestTable accordingly. Now let us consider the red flow, which arrives later than the green flow. Similarly, an ACK packet is sent from T5 to L2, carrying the path T5-H3 and CE 1. When this packet arrives at L2, L2 finds that the path for this in-port is less congested, so path T5-H3 replaces path T6-H3 as the least-congested path to reach H3, with the smaller CE value 1. This updated path information is propagated back to switches T1 and T4 by ACK packets from the red and green flows, respectively (not illustrated in Figure 3). Eventually, the BestTable of T1 to destination H3 is updated to path L2-T5-H3 with CE 1.

Note that we adopt ECMP for routing ACK packets, which can distribute ACK packets across almost all links. In this way, CAVER collects path congestion information without being influenced by temporary congestion, allowing CAVER to timely identify paths that recover from congestion.

Maintaining outputs’ remote CE. To track remote congestion, CAVER maintains a RemoteCETable mapping each (destination, outputID) pair to two fields: *CE* (8-bit) and *Time* (32-bit). The CE field stores the congestion extent of the least-congested path to represent the aggregated congestion status of all paths from this output to the destination. Entries are updated with the piggybacked BestCE value when an ACK packet traverses the switch.

Selecting acceptable paths. With the least-congested path stored in the BestTable, we now decide whether path p is acceptable to the same destination. Let ce_p denote the CE value

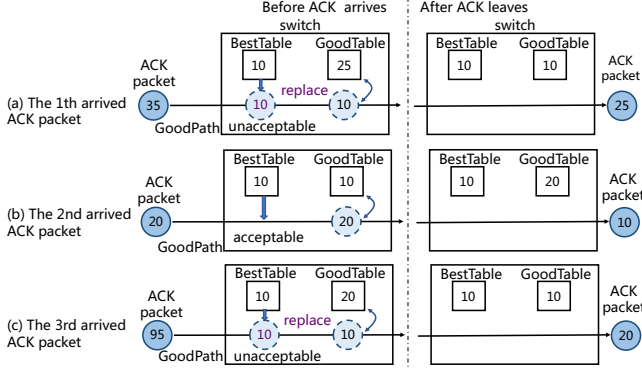


Figure 4. An example of our Give-and-Take strategy. The parameter β is set to 0.9, allowing paths with CE values below 30 to be considered acceptable.

of path p , ce_{best} represent the current least CE value stored in BestTable, and β be a constant that satisfies $0 \leq \beta \leq 1$. Since the CE value (ranges from 0 to 255) represents the bandwidth utilization of a path, we calculate p 's residual bandwidth ratio as $255 - ce_p$ and determine whether a path is acceptable by comparing its residual bandwidth ratio with that of the least-congested path. A path is considered acceptable if $255 - ce_p$ is greater than $\beta \times (255 - ce_{best})$.

Give-and-Take strategy. Straightforwardly, we can store all acceptable paths in each switch, and each ACK packet piggybacks all the information for a certain destination. Unfortunately, such a method incurs a couple of practical issues. For an ACK packet piggybacking multiple acceptable paths, we have to access the table (usually implemented with register array in switches) that stores acceptable paths multiple times. As one entry in a table can only be accessed once in a switching pipeline [36], one approach to enable multiple visits for an entry is to recirculate the ACK packet, but this will result in throughput degradation. Another possible solution is to split the table into multiple sub-tables, and place different sub-tables into different stages. However, this method requires a large number of stages to construct and manage the sub-tables, which may exceed the maximum number of stages in a switching pipeline (i.e., 12 for Tofino [36]).

To address the issue, we observe that it is not necessary to store all acceptable paths at any time. Instead, we can store just one acceptable path in a switch and update it frequently to cover more acceptable paths across a period. Meanwhile, each ACK packet carries just one acceptable path that is different from the paths carried by other ACK packets even within the same flow. Because there are a large number of ACK packets, we can provide ToR switches with sufficient paths for routing decisions. To this end, we propose a Give-and-Take strategy between an ACK packet and a switch, enabling efficient information sharing among ACK packets of all flows. Each switch maintains a GoodTable alongside the BestTable, which has the same structure. In addition to

the least-congested path, an ACK packet also carries the information of one good path (in the field of *GoodPath* and *GoodCE*). If the *GoodPath* in the ACK packet is acceptable, CAVER performs a trading-like exchange: it replaces the Path and CE in the GoodTable with the *GoodPath* and *GoodCE* from the ACK packet, and the ACK packet will carry the old *GoodPath* and *GoodCE* from the switch. In contrast, if either the Path in the GoodTable or the *GoodPath* in the ACK packet is unacceptable, we first replace the unacceptable path with the current least-congested path and then perform the Give-and-Take.

We use the example in Figure 4 to further illustrate the Give-and-Take strategy. Three ACK packets from the same host arrive at the local switch in sequence, and the situations before and after each ACK packet arrival are shown accordingly, in which the numbers indicate the CE values. The CE value in the BestTable remains unchanged, so paths with CE values below 30 are acceptable (β is set to 0.9). For the first and third ACK packets, the *GoodCE* value (e.g., 35, 95) exceeds 30, so the carried *GoodPath* is replaced with the Path in the BestTable first. Then the ACK packet “gives” the carried *GoodPath* (e.g., 10) to the local GoodTable and “takes” away the path stored in the local GoodTable (e.g., 25, 20). For the second ACK packet, since its *GoodPath* is acceptable, the Give-and-Take process occurs as usual.

When propagating acceptable paths, ACKs from the same flow should carry diverse paths, giving the receiving ToR more routing options. With this trading-like exchange, where an ACK always swaps its carried path with the path stored in the GoodTable, different ACK packets naturally propagate different acceptable paths, while each acceptable path is advertised to at least one ToR switch. This naturally reduces the probability that different ToR switches select the same path and conflict with each other.

Path collection algorithm. Putting this together, we develop the algorithm for path information collection, which runs in intermediate switches except ToR switches. The algorithm can be efficiently implemented in current programmable switches. The inputs include the received ACK packet and the ingress port. Given the ACK packet carries both the least-congested path and an acceptable path, our algorithm first updates the least-congested path, which just involves accessing the local BestTable. Then, the algorithm performs the Give-and-Take strategy, which just involves visiting the local GoodTable. As such, the ACK packet is dealt with sequentially in two stages in a programmable switch and forwarded to the next hop.

3.2 Routing Decision

For schemes that select the outport for each packet at each switch, routing decisions are made based on the split ratio among outports. For schemes that select the routing or rerouting path for a flow, routing decisions are made by ToR

switches that connect to source hosts. We describe how to decide the split ratio and select a proper path for each flow. **Workflow for the split ratio.** For adaptive routing schemes that select the outport for each packet based on local information [38, 50], CAVER provides better choices by using the local CE stored in DRETable and remote CE stored in RemoteCETable maintained in each switch. The visibility over remote links makes this enhancement especially useful for coping with asymmetries such as link failures or non-uniform background traffic, which are common in practical DCs, as prior work has shown [3, 49]. Since remote CE in RemoteCETable may be stale, directly selecting the least-congested outport can be inaccurate. Instead, we adopt a power-of-two approach: for each packet, CAVER randomly samples two available outports toward the destination and selects the less congested one. The congestion of each outport is estimated as the larger value between the local CE and remote CE, where expired remote CE defaults to 0 to encourage exploration. To avoid two table look-ups per packet, we partition both DRETable and RemoteCETable into two sub-tables, each covering half of the ports.

Workflow for the flow path selection. A ToR switch maintains four tables, namely DRETable, BestTable, PathTable and FlowTable. DRETable and BestTable have the same functionality and structure as discussed above. PathTable stores up to K real-time acceptable paths for each destination host. We select paths from the PathTable for packets of new arrival flows. Details of this table's structure will be introduced later. We store the route path for each flow in the FlowTable, guaranteeing that all packets of a flow are forwarded along a single path. FlowTable uses the hash value of a packet's 5-tuple (16-bit) as a key, which we refer to as flowId. Each entry has two values: one stores whether to use source routing or ECMP for this flow (1-bit), and the other stores the corresponding path (32-bit) for source routing (0 for ECMP).

ToR switches process ACK packets similarly but more simply than other switches. The source-side ToR attaches the ingress port's CE value to the ACK and forwards it via ECMP. The destination-side ToR extracts BestPath and GoodPath from the ACK, updates BestTable if necessary, and evaluates whether GoodPath is acceptable, similar to non-ToR switches. If GoodPath is acceptable, it is added to PathTable.

Figure 5 shows how a ToR switch handles data packets. For each packet, CAVER computes its flowId and uses a bitarray to determine whether it belongs to a new flow; bits expire automatically after a period of inactivity to mitigate hash collisions. For new flows, CAVER consults the PathTable: if an acceptable path is available, it encodes the path in the header for source routing; otherwise, it encodes one bit in the header indicating routing with ECMP. The routing decision is cached in the FlowTable so that subsequent packets of the same flow follow the same path.

Now, we look at determining the forwarding path of a new flow with the PathTable. Although all paths stored in

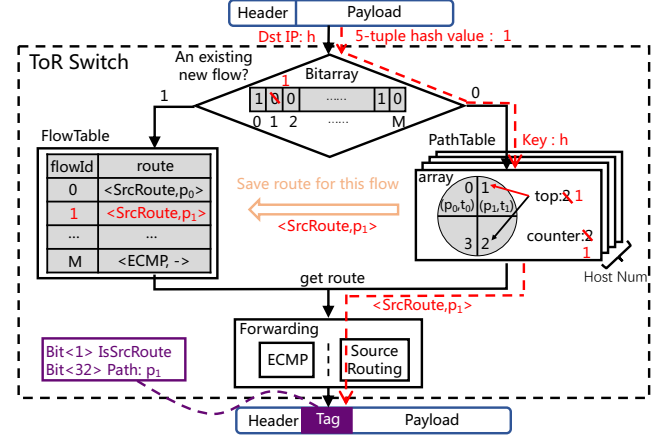


Figure 5. ToR switch data packet process pipeline. The array maximum length K is set to 4. Red line shows an example of processing a packet belongs to a new flow terminated at host h with flowId equals to 1.

PathTable have acceptable congestion content and thus can serve as potential path choices, we should follow two basic principles to achieve a better routing decision. First, two different flows that arrive within a short time period should not use the same forwarding path so as to minimize the potential contention. Second, given that the CE value for each path in the PathTable may change over time, it is essential to prioritize the most recently updated path for a newly incoming flow instead of old paths that may become outdated.

Following these two principles, we introduce our path selection strategy: For each incoming flow, CAVER select the most recently received and unselected path from the stored acceptable paths, and guarantee this path is not outdated; If no suitable path is available, ECMP is used as a fallback. To implement this, as shown in Figure 5, each PathTable entry maintains a K -length circular buffer storing acceptable paths and their reception times. For each new flow, CAVER selects the newest unused path from the buffer, ensuring each path is used at most once; it falls back to ECMP when the buffer is empty, or the selected path has expired beyond T_{exp} .

Putting the above together, we develop the overall algorithm for making routing decisions, which runs in ToR switches. The algorithm processes two types of input: an acceptable path p carried by an ACK packet a or a data packet d . For an ACK packet, the algorithm stores path p into the PathTable. For a data packet, the algorithm first checks if d belongs to an existing flow using the FlowTable. If not, the algorithm checks the PathTable for valid paths. If no paths are available or all paths are expired, the algorithm falls back to ECMP. Otherwise, the algorithm selects the newest path from the PathTable. In all cases, routing decisions are stored in the FlowTable.

Enhancing other LB schemes. Besides enhancing the packet LB schemes and flow LB schemes, CAVER can also be incorporated into other LB schemes to enhance their routing decisions, e.g., flowlet or subflow LB schemes. Existing flowlet LB schemes such as QP-scaling schemes [9, 17, 48] and subflow LB schemes such as multi-path RDMA schemes [26, 29] split a flow into multiple flowlets or subflows, each with a distinct port number. These schemes typically assign random paths to flowlets/subflows. When CAVER is used instead, each flowlet/subflow is treated as a single flow and assigned a proper path using our PathTable and FlowTable, without requiring any modification to end hosts. For rerouting schemes [13, 47], the PathTable of CAVER can be used to select a rerouting path, providing better visibility on path congestion information.

3.3 Implementation

We implement a prototype of CAVER on Tofino programmable switches. Recall that BestTable, GoodTable, and DRETable are maintained in non-ToR switches, while ToR switches manage BestTable, PathTable, FlowTable, and DRETable. All tables are implemented using register arrays (SRAM) in Tofino, ensuring state maintenance within the data plane without the need for packet recirculation, mirroring, or multicast. The source code is publicly available at Github [7].

CAVER introduces reasonable SRAM overhead. BestTable and GoodTable are keyed by destination, requiring 0.45MB per table for 50k hosts (9B/entry). RemoteCETable is keyed by (destination, outputID) pairs, requiring 8MB for 50k hosts with 32 outputs (5B/entry). PathTable stores up to k_{max} acceptable paths per destination, and thus consumes 1.6MB of SRAM for 50k hosts when each entry uses 8B and k_{max} is 4. DRETable adds negligible overhead, needing only 4 bytes per port.

The FlowTable memory footprint depends on how flow collisions are handled. A packet must not use a cached path intended for another destination host. One approach is to design the hash function such that flows to different destination hosts do not map to the same FlowTable entries. This is suitable for deployments with fewer hosts, where sufficient independent entries can be retained for each destination. Alternatively, all hosts can share the entire FlowTable, with each entry additionally storing a 24-bit destination host ID, adding 3 bytes per entry. When matching a FlowTable entry, CAVER compares the stored ID with the packet's destination ID; if they differ, the packet ignores the cached route and falls back to ECMP. This approach is more suitable for deployments with more hosts. With a 19-bit hash, the FlowTable contains 512K entries and requires 4.1 MB and 5.6 MB of SRAM under the two approaches, respectively.

In summary, flow/flowlet/subflow LB schemes require the BestTable, GoodTable, PathTable, FlowTable, and DRETable, consuming 0.9 MB in non-ToR switches and 6.6 MB in ToR

switches. Packet LB schemes require the BestTable, RemoteCETable, and DRETable, consuming 8.5 MB of SRAM per switch. Both fit comfortably within the 50–100 MB SRAM budget of commodity ASICs [33].

For communication overhead, CAVER's data packet header comprises a single path field². For an n -layer network with a maximum $2n - 2$ hops between ToR switches, the path field contains $16n - 16$ bits when allocating 8 bits for each hop's output. In typical 3-layer data centers, this results in a 4-byte path field, creating 0.27% overhead under 1.5KB MTU. CAVER's ACK packet header includes two path fields and the corresponding CE values (8 bits), totaling 10 bytes (16% overhead compared to 62-byte RoCE ACK packets [16]).

We also implement CAVER with the NS3 simulator based on ConWeave [47], to which we add ~1300 lines of C code.

3.4 Parameter Choices

CAVER has four main parameters: (i) β , the threshold for acceptable paths; (ii) K , the maximum number of paths kept per destination in the PathTable; (iii) T_{exp} , the freshness threshold for path information in the BestTable, GoodTable, RemoteCETable, and PathTable; and (iv) $W = T_{DRE}/\alpha$, the DRE time constant. The parameter β controls the set of acceptable paths for flow-level LB schemes. A value too close to 1 restricts routing flexibility by admitting only near-optimal paths, while a small value risks admitting congested paths. A moderate value balances these effects. K controls the number of path choices stored per destination in PathTable, with memory as the primary constraint. Although each path choice is selected by at most one flow, a small K is enough in practice, as new paths are added upon each ACK packet and ECMP handles the case when no path choice is available. T_{exp} bounds the staleness of path information. A small T_{exp} discards recently learned paths and forces CAVER to fall back to ECMP or local-congestion-based adaptive routing, whereas a large T_{exp} retains paths whose congestion state may have changed. Intuitively, T_{exp} should be on the order of RTTs (base RTT is 10 us in our simulation) to match the congestion-feedback loop. W controls the link utilization estimator's averaging window. A small W makes the DRE respond quicker, but also makes it more susceptible to noise from transient traffic bursts. Intuitively, W should also be set on the order of RTTs to filter the sub-RTT traffic bursts. Overall, we have found CAVER's performance to be fairly robust with: $\beta = 0.5$ to 0.8 , $K = 2$ to 4 , $T_{exp} = 30$ to 80 us, $W = 30$ to 80 us. The default parameter values for our implementation are: $\beta = 0.8$, $K = 4$, $T_{exp} = 50$ us, $\alpha = 0.2$, and $T_{DRE} = 10$ us; these defaults maintain stable performance across all simulation and testbed evaluations. We further validate the sensitivity in § 4.2.

²The 1-bit IsSrcRoute flag can be stored in the header reserved field [16].

4 Evaluation

We evaluate CAVER using large-scale NS3 simulations and hardware testbed experiments to answer the following questions: 1) How effective is CAVER under different traffic patterns and topologies (§4.1)? 2) How well does each part of CAVER perform (§4.2)? 3) Can CAVER be efficiently implemented on current switching hardware (§4.3)?

Topologies. We evaluate CAVER in a 3-layer fat-tree topology ($k=8$) with an over-subscription ratio of 2:1 [2]. There are 256 hosts, with 16 paths between each pair of them. This topology matches the largest topologies used in recent LB studies [13, 14, 47]. We also evaluate CAVER in topologies with different scales and topologies with bonding [42].

Workloads. We use three widely-used realistic workloads, i.e., AliCloud Storage [28], Meta Hadoop [45], Solar RPC [34]. These workloads lack concurrency. Thus, we also conduct simulations with modified workloads to evaluate CAVER under concurrent flows. In particular, we introduce different levels of concurrency to the AliCloud Storage workload and include LLM training workloads sampled from production servers.

Schemes compared. We compare our CAVER with ECMP, CONGA [3], HULA [21], and ConWeave [47]. For CONGA, we use a flowlet time gap of 100 us even though it has been widely shown that RDMA flows do not have flowlets [17, 47]. For HULA, we set the probe interval to 70 us following the recommendation [21]. For ConWeave, we use the default parameter values [47]. For CAVER, the parameters are set as described in § 3.4.

Configurations and metrics. We use DCQCN [54] for congestion control and priority-based flow control (PFC) [15] for flow control, both of which are commonly adopted in lossless networks, with parameters following the recommendations from HPCC[28]. For data transmission, we use the RoCEv2 Reliable Connection (RC) write mode. Each flow generates an ACK for the last data packet, while other data packets generate ACKs periodically. The interval between these data packets, which determines when ACKs are generated, is referred to as the ACK ratio. In our simulation, we configure the ratio as 1, i.e., every data packet generates an ACK. We also evaluate CAVER under different ACK packet ratios in § 4.2. To evaluate the overall performance, we use the FCT slowdown, the actual FCT of a flow normalized by the base FCT when the network has no other traffic.

4.1 Overall Effectiveness

FCT Reduction. Figure 6 shows the average and tail FCT slowdown under AliCloud Storage workload with varying traffic load levels. We can see that CAVER outperforms other LB schemes, especially in high-load scenarios. At a load level of 80%, CAVER reduces the average FCT slowdown by 12%—with a reduction of 9% for small flows (<20KB) and a reduction of 18.75% for large flows (>100KB)—and reduces tail

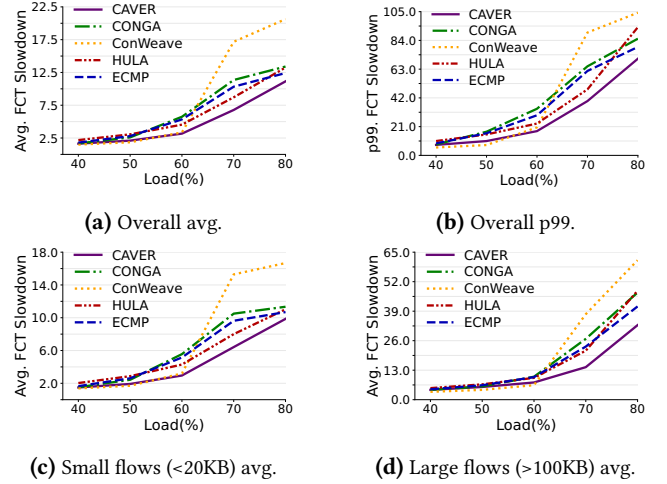


Figure 6. FCT slowdown for AliCloud Storage workloads in the 3-layer fat-tree topology with 256 hosts.

latency by 20%, compared to the best of other LB approaches. CAVER achieves this by collecting real-time and comprehensive path congestion information, which enables CAVER to avoid severely congested paths when making routing decisions. Furthermore, in high-load scenarios, more ACK packets can be used by CAVER to find less-congested paths. In contrast, ConWeave performs poorly under high load, because the reordering mechanism consumes additional buffer space and increases buffer pressure, making PFC more likely to be triggered and propagated. As a result, at the load level of 80%, CAVER reduces average FCT slowdown by 47% and tail FCT slowdown by 32% compared to ConWeave. For lower load scenarios, CAVER achieves similar performance as ConWeave and outperforms other LB methods.

Impact of topologies. Figure 7 shows average FCT slowdown across topologies. Figure 7 (a) shows the results in a 2-layer CLOS topology with 8 equal-cost paths between each pair of hosts (over-subscription ratio of 2:1), and Figure 7 (b) shows the results in a 3-layer fat-tree topology with 4 equal-cost paths (over-subscription ratio of 1:1). CAVER still outperforms existing LB methods even with fewer paths. In the 2-layer CLOS topology, CAVER shows a similar performance to ConWeave and outperforms ECMP by 44.4% at the load of 70%. In the 3-layer fat-tree topology, CAVER maintains strong performance under low load but experiences a slight performance decline at 80% load. The limited number of paths in the topology makes it difficult to find acceptable paths beyond the least-congested one. As a result, the least-congested path is often selected, leading to conflicts.

Some DC networks use bonding [42] to increase the up-link bandwidth of hosts, where each host connects to two or more ToR switches. We evaluate CAVER's performance on a 3-layer fat-tree topology with 128 hosts, and hosts employ

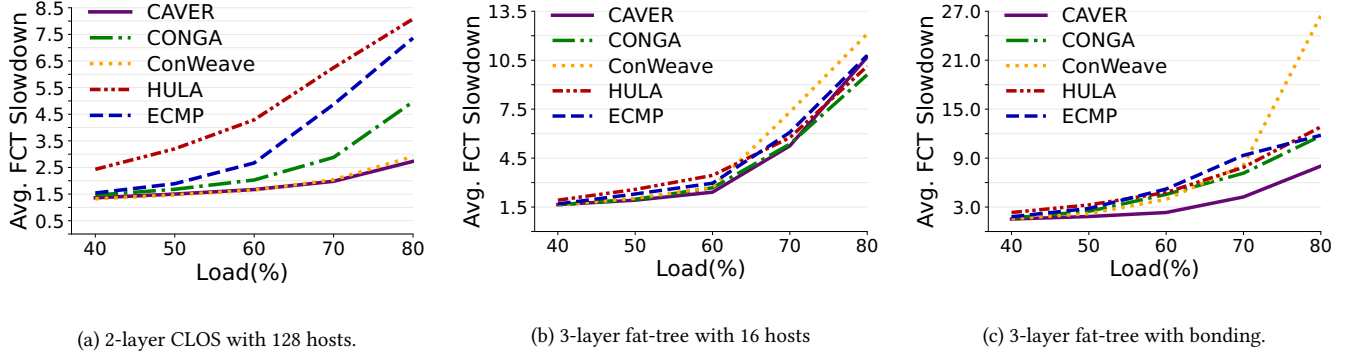


Figure 7. Average FCT slowdown under AliCloud storage workload in different topologies.

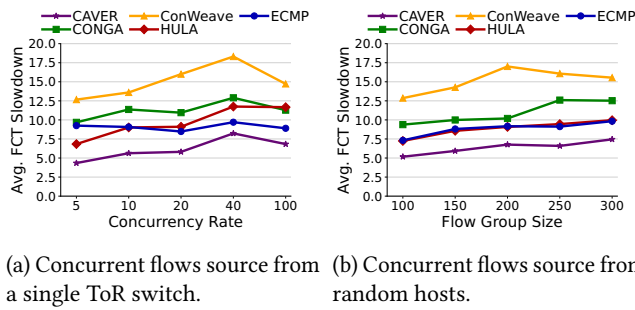


Figure 8. Concurrent flows in 3-layer fat-tree topology.

dual uplinks to ToR switches for enhanced bandwidth provisioning. This bonding configuration creates 32 equal-cost paths between cross-pod source-destination pairs – double the non-bonded scenario. Figure 7 (c) shows the results. CAVER outperforms existing LB methods under all traffic load levels. In particular, CAVER outperforms ConWeave by 50% under the load level of 60% and outperforms ECMP by 20% under the load level of 80%. This improvement comes from the ability to identify less-congested paths when the number of available paths is large.

Concurrent workloads. Concurrency is common in data-center traffic [4, 28, 45], where multiple flows target a set of hosts within a short time. With 20% load AliCloud Storage traffic as background, we inject concurrent traffic by periodically generating a group of flows targeting hosts under a randomly selected ToR. We consider two types of concurrent flows: one sourced from hosts under a single ToR switch, and the other sourced from hosts randomly distributed across the network. We control concurrency via two parameters: the concurrency rate, defined as the ratio of average arrival intervals between background and concurrent flows (higher values indicate tighter arrivals); and flow group size, defined as the number of flows per concurrent group (higher values indicate more concurrent flows). Concurrent flow sizes are drawn from the distribution of AliCloud Storage workloads.

Table 1. FCT slowdown for LLM training workload. Small flows are less than 1MB. Large flows are larger than 1MB.

LB	AVG. FCT slowdown	P99 FCT slowdown	Small flows FCT slowdown	Large flows FCT slowdown
CAVER	5.25	62.91	3.10	30.44
CONGA	5.44	64.79	3.20	31.65
ConWeave	6.43	62.02	4.68	26.44
HULA	5.74	75.82	2.96	37.55
ECMP	5.54	60.95	3.49	29.70

Figure 8 shows CAVER’s performance under concurrent workloads, varying concurrency rates and flow group sizes. In Figure 8 (a), we fix the flow group size at 200 and vary the concurrency rate. In Figure 8 (b), we fix the concurrency rate at 20 and vary the flow group size. CAVER shows significant improvement under all scenarios, reducing average FCT slowdown by up to 42.2% compared to the best-performed LB methods. HULA and CONGA perform poorly because they select the least-congested paths for routing, leading to severe path competition among concurrent flows. ConWeave increases congestion on the destination ToR switch of concurrent flows by consuming buffer for reordering, which further degrades performance. CAVER ensures stable performance under high concurrency by providing sufficient acceptable paths and using a conflict-free routing strategy.

LLM training workload. We simulate LLM training workloads mixed with cloud services on a three-layer fat-tree topology ($k = 8$) with 128 hosts and a 1:1 oversubscription ratio. LLM training periodically generates large flows that contend with other traffic. We model a complete pipeline parallelism process, including forward and backward propagation, and data parallelism (ring-all-reduce). Cloud service workloads contribute an average link load of 30%.

Table 1 presents the results, where flows larger than 1MB primarily originate from LLM training. CAVER achieves the lowest average and small-flow FCT slowdown, while ConWeave achieves the lowest large-flow FCT slowdown at the expense of small flows. ConWeave reroutes ongoing flows

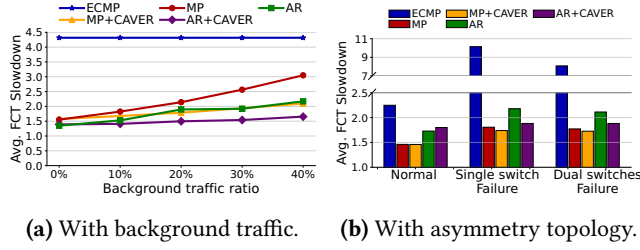


Figure 9. Existing LB methods enhanced with CAVER.

to avoid congestion, improving large-flow performance but potentially interfering with small flows. In contrast, CAVER does not reroute existing flows, which may cause congestion among large flows. However, ACKs from these flows continuously advertise path conditions, steering subsequent small flows away from congested paths and reducing their FCT slowdown.

4.2 Dive into CAVER

Enhancing existing LB. Background traffic and link failure introduce asymmetries into DC networks, which can severely affect LB performance. We evaluate whether CAVER can be incorporated into existing LB schemes to improve routing decisions in these scenarios. We combine CAVER with two representative methods, MP-RDMA (MP) [29] and adaptive routing (AR) [38]. To simulate potentially uneven background traffic, we introduce different ratios of background RDMA traffic routed with ECMP. To simulate link failure, we randomly reduce two links' bandwidth by half, either belonging to a single Core switch or dual Core switches.

Figure 9 shows the results. In Figure 9 (a), MP and AR perform well without background traffic. However, when background traffic is introduced, their performance degrades by up to 96.2% and 61.0%, respectively. When incorporated with CAVER, the performance increases by up to 45.4% and 31.2%, respectively. In Figure 9 (b), all methods' performance degrades due to the overall reduction in network bandwidth. MP performs relatively well, as it can remove failure paths by monitoring the RTT of flows. AR's performance is up to 22.3% worse than MP's, as switches are unaware of remote failures when making routing decisions. When incorporated with CAVER, AR's performance can increase by up to 15.8%.

Path collection efficiency. Recall that path information collection and routing decision are main components of CAVER. We conduct simulations to show the effectiveness of the key mechanisms of these components. First, we evaluate CAVER without information sharing, where each ACK packet only carries the CE value of the ACK path. Second, we evaluate CAVER without acceptable paths, where the ToR switch always selects the least-congested path.

Figure 10 shows the results. CAVER outperforms the other two variants at different levels of traffic load, indicating that

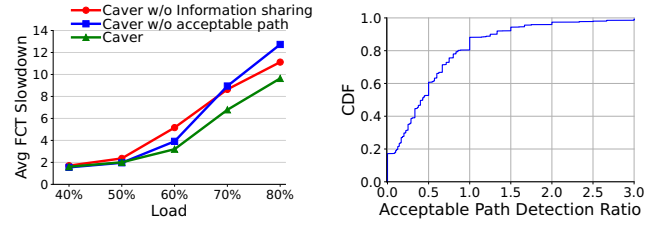


Figure 10. breakdown performance of CAVER. Figure 11. Acceptable path collected by CAVER v.s. actual acceptable paths.

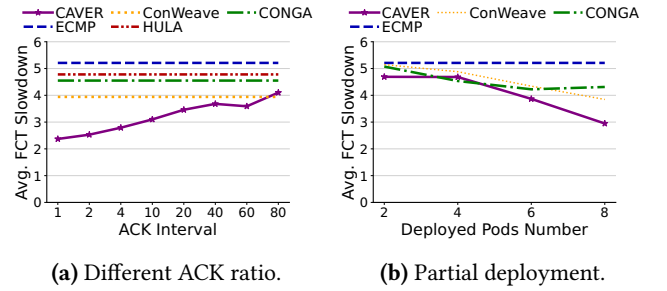


Figure 12. CAVER with different settings.

both mechanisms contribute to improving performance. For example, under moderate load (a load level of 60%), the improvement reaches 38.3% and 18.8% compared to CAVER without information sharing and without acceptable paths, respectively. Notably, as the load increases, CAVER without acceptable paths suffers a rapid increase in FCT. This is because when the number of flows increases, simply choosing the least-congested path leads to more conflicts. This highlights the importance of selecting acceptable paths.

We evaluate how many acceptable paths can be found by CAVER, compared to the total number of acceptable paths. We measure the number of distinct acceptable paths received by a ToR switch in each 50us window and the total number of acceptable paths within the window. Then we compute the acceptable path detection ratio. Figure 11 shows the CDF of the ratio under load level of 60% with AliCloud storage workload. We can see that the acceptable path detection ratio reaches 1 with an empirical probability of 80%, which means CAVER can find all acceptable paths in most cases. Interestingly, the ratio may be greater than 1, which means CAVER may find more acceptable paths than actual existing ones. This is due to the unavoidable delay in information transmission. In such cases, some acceptable paths found have already become unacceptable. However, the probability of such cases is small, so the performance is still good.

The impact of ACK ratio. RoCEv2 doesn't explicitly define the ACK ratio within the protocol [16]. The most widely deployed Mellanox RNICs have a default ACK ratio³ of 52 and provide a hardware interface (MLX5DV [39]) for adjustment. Figure 12 (a) shows CAVER's performance with different ACK ratios in the 3-layer fat-tree topology with bonding under moderate load (60%). In general, with more ACK packets generated, the amount of path congestion information piggybacked by ACK packets increases, improving CAVER's ability to collect real-time congestion information, leading to better performance.

The practical constraint on reducing the ACK ratio is the packet-processing capacity of the NIC and switch. Our small-message-dominated workload, with an average message size of 512 KB, already imposes a high packet-processing load. Even when the BPS of CX7 (400 Gbps) and Tofino1 (6.5 Tbps) is saturated and every data packet triggers an ACK, the resulting packet rates are only 140/260 Mpps and 3.0/4.8 Bpps, respectively, both within device capacity. This leaves sufficient headroom to reduce the ACK ratio for more frequent congestion feedback. For workloads with higher packet rates, the ACK ratio can be increased accordingly to limit the packet-processing overhead. Our results show that CAVER still outperforms the baseline by 25.0% with one ACK per ten data packets, and achieves at least 8.6% FCT improvement over baseline approaches even with the default ACK ratio.

Partial deployment. Deploying network-dependent solutions in data centers is challenging due to incompatibility with legacy ASIC-based switches. However, CAVER supports incremental deployment, mitigating this issue. In topologies where core switches connect multiple pods, CAVER can be deployed in only a subset of pods, while switches in other pods and core switches can stay unchanged. Traffic originating from CAVER-enabled pods is managed by CAVER, while other traffic simply adopts ECMP, ensuring seamless coexistence and interoperability.

Under partial deployment, core switches simply forward ACKs without CAVER-specific logic. The preceding CAVER-capable switch infers the core-switch port from the selected ACK output port, recording the complete path but omitting that port's congestion state, potentially reducing path selection accuracy. For data forwarding, the preceding switch pushes an MPLS label specifying the output port; the core switch forwards accordingly and pops the label, relying on widely available MPLS functionality.

Figure 12 (b) shows the results on a fat-tree topology with 8 pods. CAVER consistently outperforms ECMP and generally achieves better performance than other baselines as deployment scales. This is because CAVER adapts its routing decisions for managed flows based on load introduced by unmanaged flows, ensuring efficient load balancing even in mixed environments. We do not evaluate HULA, which is

³Measured with a Mellanox ConnectX-6 RNIC under default settings.

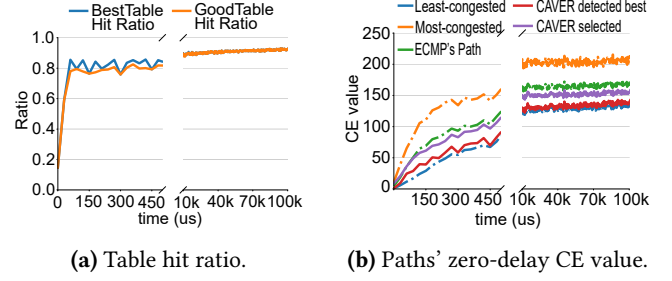


Figure 13. CAVER cold-start with $t=0$ traffic injection. Zero-delay CE refers to a path's CE value calculated with the instantaneous links' CE value, reflecting precise real-time congestion status.

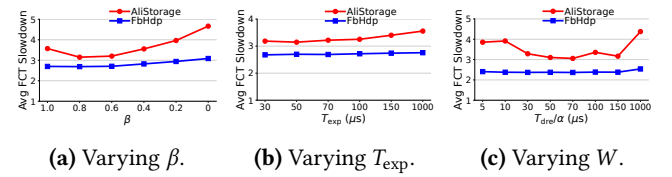


Figure 14. Parameter sensitivity under different workloads.

hard to partially deploy since core switches must support both processing and multicasting HULA's probe packets.

Cold-start process of CAVER. We dive into the cold-start process of CAVER. Flows start entering the network at time zero, and all tables begin with empty. On each flow's arrival, we record whether there are valid entries in the PathTable and the BestTable at the source ToR switch. We also compute all paths' CE value based on the instantaneous links' CE value, i.e., paths' zero-delay CE value. Specifically, we record the zero-delay CE value of CAVER's detected least-congested path, CAVER's selected acceptable path for this flow, and the path corresponding to the ECMP result of this flow. We also search for the least-congested and the most-congested path from all the paths. In Figure 13, the period of 0-450 us includes the warm-up process of CAVER. After 2-3 RTT (20 us), CAVER finds a less-congested path than ECMP besides the detected least-congested path for 80% of flows, indicating a short warm-up process. The period after 10ms reflects CAVER's stable performance. CAVER can detect an acceptable path for 87% of flows. The best path recorded by CAVER has nearly identical CE to the actual least-congested path. The zero-delay CE gap between CAVER selected paths and the actual least-congested path is bounded within 20%, which aligns with our predefined CE threshold for judging the acceptable path. This validates the accuracy and timeliness of CAVER's collected path congestion information.

Parameter sensitivity of CAVER. We study how different parameter settings affect CAVER's performance; Figure 14 shows the results. β determines whether a path is considered acceptable. At $\beta = 1$, CAVER routes almost exclusively over

the least-congested path, which may cause concurrent contention and conflicts; at $\beta = 0$, all paths are deemed acceptable, degenerating to ECMP. For AliStorage, performance degrades when β is either too large or too small, whereas for FbHdp it stays stable and improves slightly as β grows. For both workloads, $\beta \in [0.6, 0.8]$ achieves good performance.

T_{exp} acts as an expiration mechanism that prevents reliance on stale path information. Performance degrades only when T_{exp} is very large, because CAVER updates path information frequently and thus operates on fresh data most of the time. Expiration rarely triggers, and even in the few cases where information is slightly stale, it remains useful within a short window.

$W = T_{\text{DRE}}/\alpha$ controls the effective averaging window of the link utilization estimator. An excessively small W makes path selection overly sensitive to transient traffic bursts, causing otherwise suitable paths to be classified as congested and avoided. An excessively large W delays the estimator’s response to changes in link utilization. For both workloads, $W \in [30, 70] \mu\text{s}$, corresponding to three to seven times the base RTT ($10 \mu\text{s}$), achieves good performance.

4.3 Hardware Testbed Prototype

Setup for hardware testbed. Our testbed topology consists of two leaf switches and two spine switches. CAVER is deployed on all switches. We utilize the multi-pipeline feature provided by Tofino1 [36] to virtualize one Tofino1 into two leaf switches and another Tofino1 into two spine switches. To achieve an oversubscription ratio of 2:1, we configured the links between switches to operate at 10 Gbps and the links between leaf switches and hosts to operate at 25 Gbps. We do not choose higher link bandwidth because our servers are all equipped with PCIe 3.0 x8 slots, which provide a maximum bandwidth of 60 Gbps per NIC. As a result, each NIC with two ports can support up to 25 Gbps per port. The evaluation of CAVER with higher link bandwidth has been shown above in simulation (§4.1). We enable the PFC for flow control and DCQCN for congestion control, with the same parameters for DCQCN as in simulations. We compare CAVER with ECMP, and do not compare with CONGA and HULA due to the lack of available P4 implementations. ConWeave requires specific features of Tofino2, which cannot be implemented with Tofino1.

Results. Figure 15 shows CAVER’s performance with different RDMA applications. We set the load level at 60% for all workloads. We observe that CAVER reduces the average FCT by 11%-49% compared to ECMP across different workloads. Additionally, CAVER decreases tail FCT by up to 86.5% compared to ECMP. These results demonstrate that CAVER effectively collects path information and selects acceptable paths for each flow to improve RDMA LB performance.

Overhead. CAVER’s resource usage is evaluated on a low-end Tofino1 switch with limited hardware resources. Table 2

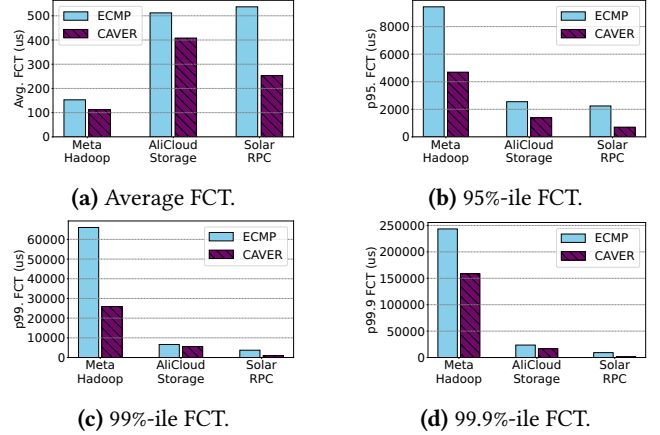


Figure 15. FCT comparison for different workload with 60% in Lossless RDMA measured in the testbed.

Table 2. CAVER’s resource consumption on switches in a 10k-host topology.

LB	Switch	Computational		Memory	
		ALUs	Gateways	SRAM	TCAM
Flow	ToR	12.5%	2.5%	16.5%	5.0%
	Non-ToR	4.2%	3.1%	2.1%	5.0%
Packet	ToR	6.3%	3.0%	17.4%	5.0%
	Non-ToR	6.3%	3.0%	17.4%	5.0%

shows hardware resource utilization with table sizes configured for up to 10k hosts. Overhead comes from CAVER’s functions as well as baseline switch functionality, including routing tables and ARP tables. In total, CAVER consumes less than 20% of computational resources and SRAM on both ToR and non-ToR switches for both LB schemes. In theory, the SRAM footprint should be about 4.5% per ToR and 1.1% per non-ToR switch for flow-level LB, and about 5.7% per switch for packet-level LB at this scale. The measured usage in Table 2 is larger, mainly because register arrays allocate a minimum of 128 bits per entry, whereas a BestTable entry needs only 72 bits, a PathTable entry 64 bits, a RemoteCETable entry 40 bits, and a FlowTable entry 16 bits; this theory-to-practice ratio (128/width) is independent of scale. Based on this trend, extrapolating to the 50k-host scale, we estimate the actual maximum footprint at about 8.5 MB for flow-level LB and about 25 MB for packet-level LB; the latter exceeds our Tofino1 (15MB) but is still reasonable for Tofino2 (50–100 MB). This overhead can be partially mitigated by packing multiple entries into a single register: for packet-level LB, three 40-bit RemoteCETable entries fit in one 128-bit register (3× reduction); for flow-level LB, two 64-bit PathTable entries fit in one register (2× reduction).

For overhead on link bandwidth, CAVER adds headers to data packets and ACK packets, introducing no extra packets.

5 Discussion

Memory consumption. If CAVER’s state exceeds the available switch SRAM at scale, two simple changes can reduce its memory footprint. First, the BestTable, GoodTable, and PathTable can use the destination ToR instead of the destination host as the key. Since hosts under the same ToR share the same network paths, their ACK information can also be shared, improving path discovery. Second, these tables can maintain entries only for hosts with higher performance requirements, while traffic to other destinations falls back to ECMP or packet spraying. Both changes require only table configuration during initialization. In addition, the SRAM capacity constraint can be further relaxed by leveraging external memory [22].

Synchronized cold traffic. When traffic arrives simultaneously after a long idle period, as in LLM training, CAVER initially falls back to ECMP (flow-level) or packet spraying (packet-level). Without fresh path information, transient congestion remains possible. As updated information arrives, CAVER steers subsequent traffic away from congestion caused by earlier traffic, preventing further buildup. The initial congestion can be further mitigated through application-level coordination that slightly staggers the start times of flows, allowing later flows to benefit from fresher path information and select less-loaded paths with fewer collisions.

Path exploration in larger networks. In extremely large networks, the number of equal-cost paths grows, and CAVER may not collect congestion for all paths if ACK volume per destination remains unchanged. This issue is not unique to CAVER but also affects other LB schemes [3, 14, 21, 52]. One possible solution is to increase the ACK packet ratio, even adding some probing packets, which can help detect more paths effectively.

Handling more failure scenarios. Beyond the bandwidth degradation evaluated, CAVER remains robust under other failures. Upon complete link failure, ACKs stop advertising affected paths, and stale PathTable entries quickly expire. For packets already on these paths, the adjacent switch falls back to ECMP, selecting a valid output port and updating the header for subsequent ECMP forwarding. End-host recovery for data-packet and ACK losses remains unchanged. A lost ACK only removes one path-information update without affecting data delivery.

Deployability of CAVER. CAVER requires all switches in the network to support its functionality to achieve full LB performance improvement. However, these switches do not need to be as highly programmable as Tofino; they only need to support state updates and maintenance in the data plane. This capability is not particularly difficult to implement in future switch chips. We hope that CAVER can provide insights for the design of next-generation RDMA switches.

6 Related Work

Besides the most relevant LB works discussed in the main text, CAVER is also inspired by the following topics.

Congestion detection. Several approaches enhance transport performance by leveraging network feedback to infer path congestion. HPCC [28], PINT [6], PowerTCP [1] and Poseidon [51] collect per-switch congestion information through in-network telemetry piggybacked by ACK packets. Swift [24], Falcon [46], OSCAR [53], and CCC [18] infer path congestion from RTT or delay measurements obtained from ACK feedback. However, both approaches can only observe congestion information along the paths traversed by existing traffic, limiting their visibility into alternative paths. HULA [21] uses dedicated probes to cover all paths, while Hermes [52] actively probes additional candidate paths, both at the cost of extra probing traffic. In contrast, CAVER provides an efficient and low-overhead solution for accurately detecting congestion levels across multiple paths.

Non-cooperative routing. Effective LB routing requires avoiding conflicts in a distributed manner. Non-cooperative routing has been widely studied, with prior work showing that selfish strategies can degrade global performance [23, 32, 35, 44]. Game-theoretic approaches mitigate this issue by introducing pricing models [8, 30] or restricting individual policies [40]. Inspired by these insights, CAVER adopts a lightweight strategy where each source ToR makes slight compromises, ensuring efficient bandwidth usage and reducing congestion from path competition.

7 Conclusion

In this paper, we identify the inefficiency of existing LB schemes in improving RDMA performance, as they primarily rely on rerouting flows after detecting congestion. Instead, we take a different approach – optimizing the initial path selection for each flow to enhance LB performance. We present CAVER, a framework that efficiently gathers near real-time path congestion information and leverages it to route flows along less-congested paths. Through software simulations and hardware testbed evaluations, we show that CAVER outperforms existing LB methods, particularly under heavy traffic loads. Moreover, CAVER is resource-efficient to be implemented in current switching hardware.

Acknowledgments

We thank our shepherd KyoungSoo Park and the anonymous reviewers for their valuable feedback. This work was funded in part by the National Natural Science Foundation of China (No. 62402025), the Open Research Fund of State Key Laboratory of Internet Architecture (HLW2025ZD01), the Shandong Provincial Natural Science Foundation of China (ZR2024LZH011), and the Fundamental Research Funds for the Central Universities. Menghao Zhang, Yuan Yang, and Mingwei Xu are the corresponding authors.

References

- [1] Vamsi Addanki, Oliver Michel, and Stefan Schmid. 2022. PowerTCP: Pushing the performance limits of datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 51–70.
- [2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review* 38, 4 (2008), 63–74.
- [3] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the ACM SIGCOMM 2014 Conference*. 503–514.
- [4] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*. 63–74.
- [5] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, et al. 2023. Empowering Azure Storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 49–67.
- [6] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic in-band network telemetry. In *Proceedings of the ACM SIGCOMM 2020 Conference*. 662–680.
- [7] CAVER. 2025. CAVER repository. <https://github.com/CAVER-LB>. (2025).
- [8] Ron Cocchi, Scott Shenker, Deborah Estrin, and Lixia Zhang. 1993. Pricing in computer networks: motivation, formulation, and example. *IEEE/ACM Transactions on networking* 1, 6 (1993), 614–627.
- [9] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 57–70.
- [10] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. 2021. When Cloud Storage Meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 519–533.
- [11] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 202–215.
- [12] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the ACM SIGCOMM 2017 Conference*. 29–42.
- [13] Jinbin Hu, Wenxue Li, Xiangzhou Liu, Junfeng Wang, Bowen Liu, Ping Yin, Jianxin Wang, Jiawei Huang, and Kai Chen. 2025. FLB: Fine-grained Load Balancing for Lossless Datacenter Networks. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. 365–380.
- [14] Jinbin Hu, Chaoliang Zeng, Zilong Wang, Junxue Zhang, Kun Guo, Hong Xu, Jiawei Huang, and Kai Chen. 2023. Enabling Load Balancing for Lossless Datacenters. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. 1–11.
- [15] IEEE. [n. d.]. IEEE 802.1Qbb: Priority-based Flow Control. <http://www.ieee802.org/1/pages/802.1bb.html>. ([n. d.]). IEEE 802.1 Working Group.
- [16] InfiniBand Trade Association. 2020. *InfiniBand Architecture Specification Release 1.4 Annex A17: RoCEv2*. Technical Report. InfiniBand Trade Association.
- [17] X. Jia, Z. Yao, C. Peng, et al. 2024. Turbo: Efficient Communication Framework for Large-scale Data Processing Cluster. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 540–553.
- [18] Wanchun Jiang, Haoyang Li, Kai Wang, Yujie Hu, Xiao Han, Danfeng Shan, Fengyuan Ren, Jiawei Huang, and Jianxin Wang. 2026. CCC: Re-architecting Delay-based Congestion Control in Datacenter Networks. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. 1553–1570.
- [19] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. 2024. MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 745–760.
- [20] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. 2017. Clove: Congestion-Aware Load Balancing at the Virtual Edge. In *Proceedings of the 13th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 323–335.
- [21] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. HULA: Scalable Load Balancing Using Programmable Data Planes. In *Proceedings of the Symposium on SDN Research (SOSR)*. 1–12.
- [22] Daehyeok Kim, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, and Srinivasan Seshan. 2018. Generic External Memory for Switch Data Planes. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. 1–7.
- [23] Elias Koutsoupias and Christos Papadimitriou. 2009. Worst-Case Equilibria. *Computer science review* 3, 2 (2009), 65–69.
- [24] Gautam Kumar, Nandita Dukkupati, Keon Jang, Hassan MG Wasel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the ACM SIGCOMM 2020 Conference*. 514–528.
- [25] Petr Lapukhov, Ariff Premji, and Jon Mitchell. 2016. *Use of BGP for Routing in Large-Scale Data Centers*. Technical Report RFC7938. RFC Editor. <https://www.rfc-editor.org/info/rfc7938>
- [26] Yanfang Le, Rong Pan, Peter Newman, Jeremias Blendin, Abdul Kabani, Vipin Jain, Raghava Sivaramu, and Francis Matus. 2024. STrack: A Reliable Multipath Transport for AI/ML Clusters. *arXiv preprint cs.NI/2407.15266* (2024).
- [27] Wenxue Li, Xiangzhou Liu, Yunxuan Zhang, Zihao Wang, Wei Gu, Tao Qian, Gaoxiong Zeng, Shoushou Ren, Xinyang Huang, Zhenghang Ren, et al. 2025. Revisiting RDMA Reliability for Lossy Fabrics. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 85–98.
- [28] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: high precision congestion control. In *Proceedings of the ACM SIGCOMM 2019 Conference*. 44–58.
- [29] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. 2018. Multi-Path Transport for RDMA in Datacenters. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 357–371.
- [30] Jeffrey K MacKie-Mason and Hal R Varian. 1995. Pricing congestible network resources. *IEEE journal on Selected Areas in Communications* 13, 7 (1995), 1141–1149.
- [31] Gary Malkin. 1998. *RIP version 2*. Technical Report. Technical Report.
- [32] Marios Mavronicolas and Paul Spirakis. 2001. The price of selfish routing. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. 510–519.
- [33] R. Miao, H. Zeng, C. Kim, and et al. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the ACM SIGCOMM 2017 Conference*. 15–28.

- [34] R. Miao, L. Zhu, S. Ma, et al. 2022. From luna to solar: the evolutions of the compute-to-storage networks in Alibaba cloud. In *Proceedings of the ACM SIGCOMM 2022 Conference*. ACM, 753–766.
- [35] Michael Mitzenmacher. 2001. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (2001), 1094–1104.
- [36] Barefoot Networks. 2023. *Tofino Native Architecture*. Barefoot Networks. https://github.com/barefootnetworks/Open-Tofino/blob/master/PUBLIC_Tofino-Native-Arch.pdf Accessed: 2024-12-15.
- [37] NS-3 Consortium. 2023. Network Simulator 3 (NS-3). (2023). <https://www.nsnam.org/> Accessed: 2025-01-17.
- [38] NVIDIA. 2025. Adaptive routing. (2025). <https://docs.nvidia.com/networking/display/ufmsdnappmv4184/adaptive+routing> Accessed: Sep. 10, 2025.
- [39] NVIDIA Networking. 2023. *RDMA Core Documentation*. <https://docs.nvidia.com/networking/display/rdmacore50> Accessed: December 14, 2024.
- [40] Ariel Orda, Raphael Rom, and Nahum Shimkin. 1993. Competitive routing in multiuser communication networks. *IEEE/ACM Transactions on networking* 1, 5 (1993), 510–521.
- [41] Cheng Qi, Wei Wu, Yu Wang, et al. 2025. SGLB: Scalable and Robust Global Load Balancing in Commodity AI Clusters. In *Proceedings of the ACM SIGCOMM 2025 Conference*. ACM, 626–644.
- [42] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. 2024. Alibaba HPN: A Data Center Network for Large Language Model Training. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 691–706.
- [43] Yakov Rekhter, Tony Li, and Susan Hares. 2006. *A Border Gateway Protocol 4 (BGP-4)*. Technical Report. Technical Report.
- [44] Tim Roughgarden and Éva Tardos. 2002. How bad is selfish routing? *Journal of the ACM (JACM)* 49, 2 (2002), 236–259.
- [45] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the Social Network’s (Datacenter) Network. In *Proceedings of the ACM SIGCOMM 2015 Conference*. 123–137.
- [46] Ankit Singhvi, Nandita Dukkkipati, Praveen Chandra, et al. 2025. Falcon: A Reliable, Low-Latency Hardware Transport. In *Proceedings of the ACM SIGCOMM 2025 Conference*. ACM, 248–263.
- [47] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. 2023. Network Load Balancing with In-network Reordering Support for RDMA. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 816–831.
- [48] Feng Tian, Yang Zhang, Wei Ye, Cheng Jin, Ziyang Wu, and Zhi-Li Zhang. 2021. Accelerating Distributed Deep Learning using Multi-Path RDMA in Data Center Networks. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. 88–100.
- [49] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 407–420.
- [50] Jiang Liu Wang, Chao Wang, Wei Lin, Ennan Zhai, Jiesheng Wu, and Qiang Liu. 2025. Alibaba Stellar: A New Generation RDMA Network for Cloud AI. (2025).
- [51] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, TS Eugene Ng, Neal Cardwell, and Nandita Dukkkipati. 2023. Poseidon: Efficient, robust, and practical datacenter CC via deployable INT. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 255–274.
- [52] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. 2017. Resilient Datacenter Load Balancing in the Wild. In *Proceedings of the ACM SIGCOMM 2017 Conference*. 253–266.
- [53] Zhaochen Zhang, Feiyang Xue, Rui Ning, Keqiang He, Gianni Antichi, Jiaqi Gao, Zhimeng Yin, Kexin Liu, Rui Li, Zhengqi Cui, et al. 2026. OSCAR: O(1)-Step Convergence and Readily-deployable Congestion Control. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. 543–570.
- [54] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohammad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 523–536.