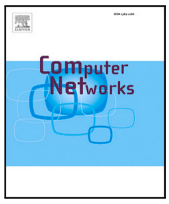
Computer Networks 259 (2025) 111088

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Reunion: Receiver-driven network load balancing mechanism in AI training clusters

Mingyao Wang [a], Keqiang He [c], Peirui Cao [a,*], Jiong Duan [a], Dongliang Lv [b], Zehao Yu [b], Yanqing Chen [a], Chengyuan Huang [a], Wanchun Dou [a], Guihai Chen [a], Chen Tian [a]

[a] Nanjing University, China
[b] ZTE, China
[c] Shanghai Jiao Tong university, China

ARTICLE INFO

Keywords:
Load balancing
Distributed
Network

ABSTRACT

RDMA over Converged Ethernet (RoCEv2) enables high-performance networking for large-scale model training but faces challenges due to traffic characteristics such as elephant flows, low entropy, and traffic bursts. Conventional load-balancing techniques like ECMP struggle with hash collisions, causing increased tail latency. Advanced solutions, such as source routing and enhanced ECMP, mitigate these issues but still have hash collisions when there are multiple sources. While packet spraying and flow slicing help alleviate load imbalances due to hash collisions, they can intensify packet reordering issues. Reunion, a novel mechanism for RoCEv2 environments, tackles three critical challenges of flowlet-based rerouting: (1) Rerouting decisions introduce new hash conflicts; (2) The out-of-order packets caused by rerouting has a significant impact on small flows; (3) Setting appropriate flowlet timeout values in high-bandwidth environments is difficult. By utilizing Count-Min-Sketch to filter out small flows and aggregating real-time congestion data, Reunion enables source switches to make dynamic rerouting decisions for elephant flows, minimizing congestion hotspots. Simulations conducted using NS-3 highlight Reunion's robustness and effectiveness in reducing tail latency. Under varying network loads, Reunion outperforms existing load-balancing schemes such as Conga, LetFlow, ECMP, and ConWeave, achieving tail latency reductions ranging from 10.9% to 62.1%.

1. Introduction

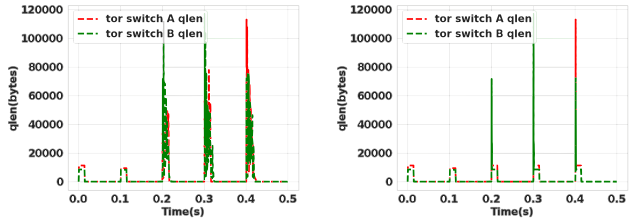
RDMA over Converged Ethernet (RoCEv2) delivers high-performance networking support for large-scale model training workloads. However, the traffic characteristics in these scenarios – such as elephant flows, low entropy, and bursts – pose significant challenges. For instance, in the context of Allreduce, the majority of the communication flows are elephant flows. The communication pattern ensures that within a stage, a node receives only one elephant flow, and the efficiency of each stage depends on the completion time of the slowest elephant flow. Therefore, it is necessary to ensure that the transmission paths of the elephant flows are entirely disjoint. Traditional load balancing methods like ECMP struggle to effectively manage hash collisions under these conditions, leading to increased tail latency [1–4] and decreased model training efficiency.

Enhanced-ECMP and source routing only alleviate partial congestion [2,5]. However, when multiple sources are involved, their routing decisions can conflict, undermining the effectiveness of these approaches. Techniques such as packet spraying and flow slicing [1] mitigate hash collisions and load imbalance, but they fail to address the performance degradation caused by packet reordering at the endpoints. AWS's Scalable Reliable Datagram (SRD) [6] achieves reliable transmission through intelligent path selection. However, its adoption in most data centers remains limited due to the extensive requirements for custom modules for SRD's operation.

Dividing elephant flows into finer granularity without reaching the per-packet level can enhance performance in scenarios with low traffic entropy, such as those involving large language models (LLMs) training. For instance, flowlet-based load balancing [7–9] in TCP networks enables effective load distribution while maintaining ordered delivery. Load-balancing schemes that perform rerouting at the flowlet level offer high flexibility, greater control over out-of-order delivery compared to per-packet schemes, and compatibility with topological heterogeneity. However, in RoCEv2-driven LLM training scenarios, flowlet-based rerouting schemes face three main challenges: (1) The rerouting decisions can introduce hash collisions because the new paths are not fully disjoint; (2) The out-of-order packet delivery caused by

* Corresponding author.
E-mail address: caopeirui@nju.edu.cn (P. Cao).

https://doi.org/10.1016/j.comnet.2025.111088
Received 19 December 2024; Received in revised form 21 January 2025; Accepted 25 January 2025
Available online 4 February 2025
1389-1286/© 2025 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.



(a) LetFlow (b) Reunion

Fig. 1. Switch queue length statistics.

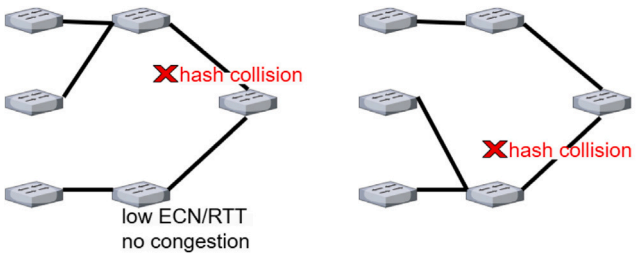


Fig. 2. New hash collision after rerouting.

rerouting has a significant impact on small flows; (3) Determining the flowlet timeout value (FTV) is difficult in high-bandwidth RDMA environments. While ConWeave [9] simplifies configuring the flowlet timeout (i.e., addressing the third challenge), it still struggles to mitigate the herd effect and achieve convergence. Additional details on these limitations are provided in Section 2.2.

Reunion addresses the three main challenges of flowlet-based load balancing by making more precise decisions for rerouting elephant flows. Designed for deployment on network switches, Reunion utilizes Count-Min-Sketch (CMS) to efficiently filter out small flows and gather real-time congestion data at each hop along the traffic path. The destination switch aggregates this congestion data to pinpoint potential bottlenecks and instructs the source switch to reroute specific elephant flows, avoiding congestion hotspots while minimizing the herd effect. By eliminating the need to fine-tune flowlet timeout values (FTVs), Reunion streamlines load balancing, ensuring robust and efficient path selection. A comprehensive explanation of the design is provided in Section 3.

In our NS-3 simulations, we evaluated the performance of Reunion and compared it with other state-of-the-art load-balancing schemes, including Conga [8], LetFlow [7], ECMP, and ConWeave [9]. Under high network load conditions, Reunion achieved significant reductions in tail latency: 43.8% compared to Conga, 44.8% compared to LetFlow, 44.9% compared to ECMP, and 10.9% compared to ConWeave. Under low network load conditions, the reductions were similarly impactful, with 62.1%, 46.3%, 45.4%, and 17.9% improvements, respectively. Additionally, experiments across various parameter configurations demonstrated the robustness of Reunion, consistently achieving at least a 61.9% reduction in tail latency compared to ECMP under 75% network load.

2. Background and motivation

2.1. Fine-grained network load balancing is necessary in AI training

In the context of large-scale model training [1–4,9–13], traffic patterns are dominated by elephant flows of comparable size originating from GPUs. This behavior contrasts with traditional data center communication patterns, such as those observed in Data Mining, MapReduce, and other similar workloads.

Table 1
The statistics of flows in a production training scenario.

Type	Flow count	Size
Tensor Parallelism (TP)	77 760	503 MB
Pipeline Parallelism (PP)	69 120	503 MB
Data Parallelism (DP)	3840	3.07 GB

Table 2
Comparison of existing load balancing schemes and Reunion.

Scheme	Deployment	Granularity
FlowBender [15]	Host	Flow
MPTCP [16]	Host	Packet
Hermes [17]	Host	Packtet
Clove [18]	Host	Flowlet
Hedera [19]	Centralized	Flow
ECMP[20]	Switch	Flow
WCMP [21]	Switch	Flow
RPS [22]	Switch	Packet
DRILL [23]	Switch	Packet
Conga [8]	Switch	Flowlet
LetFlow [7]	Switch	Flowlet
ConWeave [9]	Switch	Flowlet
Reunion	Switch	Flowlet

Meta [4] summarizes this new communication pattern arising from LLM training as follows: **(1) Low Entropy; (2) Burstiness; (3) Elephant Flows.** Take Ring Allreduce as an example: each GPU's training computation output (i.e., gradients) must be synchronized with the outputs from other GPUs in the cluster to maintain a consistent global state. As illustrated in Fig. 1, the variations in switch queue length reflect the impact of synchronous microbursts generated by GPUs, leading to periodic queue buildups over time. In a production-scale large model training scenario at our partner company involving thousands of GPUs, the statistics for the generated flows are detailed in Table 1. The size of the flows in each phase is the same. RoCEv2 congestion control algorithms, e.g., DCQCN [14], are deployed to reduce congestion and packet loss for RDMA traffic. Additionally, the oversubscription ratio in the ToR-Aggregation layer is maintained at 1:1 to minimize hash collisions and reduce the likelihood of network congestion, as recommended in [2,3].

Due to the nature of collective communication, elephant flows often burst simultaneously, with the speed of the slowest flow dictating the efficiency of the training task. However, hash collisions can significantly prolong the completion time of affected flows. This issue has been observed in production environments by major companies such as Alibaba [2], Meta [4], ByteDance [3], and Google [1]. As such, minimizing the number of elephant flows that experience hash collisions is critical, ideally eliminating them.

The research on load balancing for traditional data center workloads is vibrant, as Table 2 summarizes. They perceive congestion and reroute in specific ways at different granularities and deployment locations [1,7,8,15,17–19,23–32]. In supporting large-scale model training workloads, ByteDance [3] and Alibaba [2] design their network topologies with pre-assigned fixed routes for all communications to optimize efficiency. Google [1] adopts a slicing approach, dividing flows into smaller segments, with each slice traversing a different path. Meta leverages e-ECMP [4] for enhanced load balancing and indicates plans to explore rerouting elephant flows at the flowlet granularity in future work, similar to the approach employed by LetFlow [7].

Flowlet granularity achieves a balance between packet granularity and flow granularity, offering numerous advantages. First, it offers greater flexibility compared to flow granularity and is easy to detect. If the time interval between the arrival of consecutive packets exceeds a predefined flow timeout value (FTV), the subsequent packet marks the start of a new flowlet. In scenarios dominated by elephant flows, the appearance of flowlets usually means congestion because the congestion

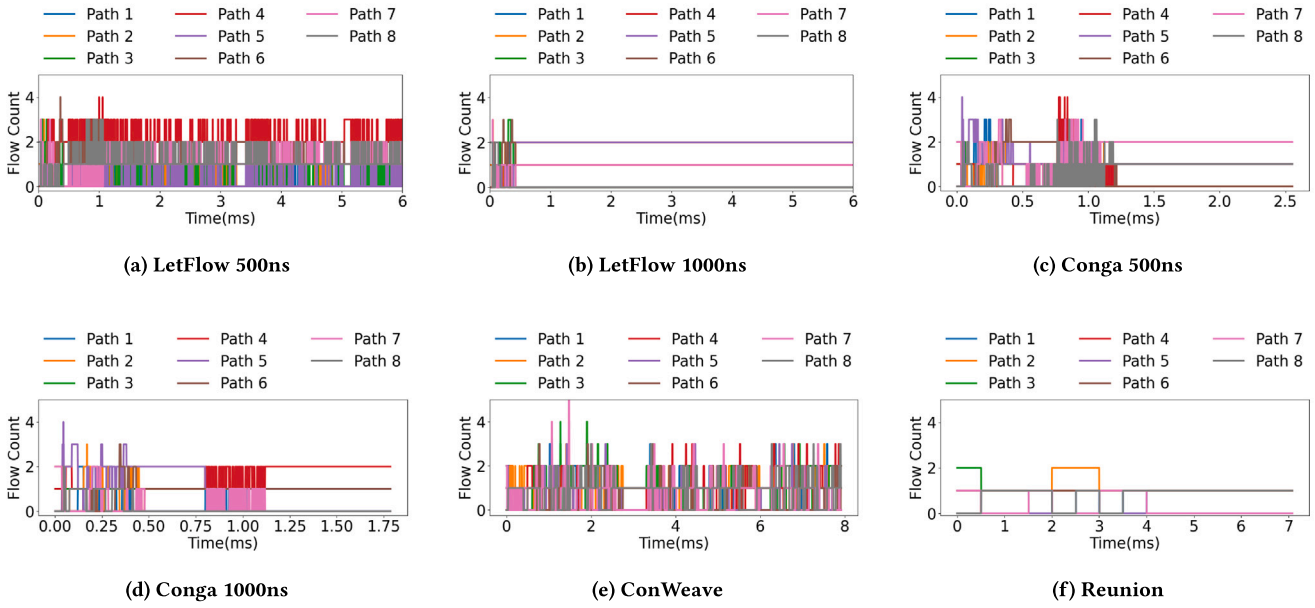


Fig. 3. Frequent rerouting by LetFlow, Conga, and ConWeave results in continuous hash collisions. The *y*-axis represents the number of flows per path, with the ideal state being one or fewer flows per path.

control algorithm will reduce the transmission speed after sensing congestion, thereby delaying the sending time of the next packet. Second, the performance of flowlets is more controllable than that of packet spraying because each flowlet is transmitted over a single path, making it easier to observe and resulting in fewer out-of-order packets. However, as we will discuss in the following subsection, flowlet-based load balancing has three major shortcomings: (1) Frequent rerouting triggers new hash collisions, leading to reduced throughput. (2) The out-of-order caused by rerouting has a significant impact on small flows; (3) Determining an appropriate FTV is challenging in high-bandwidth RDMA networks.

2.2. Rerouting decisions may introduce new hash collisions

If the rerouting decision introduces new hash collisions, the traffic load balancing performance remains suboptimal, as shown in Fig. 2. Because source switches are unaware of each other's rerouting decisions, they can cause potential hash collisions at specific hops even after rerouting. To examine the performance of state-of-the-art flowlet-based traffic load balancing schemes, we conducted a simulation using an 8 × 8 leaf-spine topology with an oversubscription of 1:1 (i.e., the leaf-spine network is non-blocking) and a network load of 75%. In this setup, for each elephant flow, there exists a path that does not intersect at all with the paths chosen by other elephant flows. If each path only carries one elephant flow, then there are no hash collisions, thereby achieving optimal performance. We randomly sampled eight paths and calculated the number of elephant flows on them.

We set Conga's FTV to be very small (500 ns and 1000 ns, respectively), which improves the identification of new flowlets in RDMA environments. For LetFlow, we used the same parameters as Conga; for ConWeave, a load-balancing algorithm designed for RDMA communications, we used the parameter values recommended in their original paper [9]. As shown in Fig. 3, while rerouting decisions are continuously made, they may still introduce new hash collisions, meaning that not every path eventually has fewer than or equal to one elephant flow. Hash collisions persist even after the system converges. ConWeave behaves differently, as it continuously performs rerouting in search of better paths.

2.3. Rerouting decisions impact smaller flows

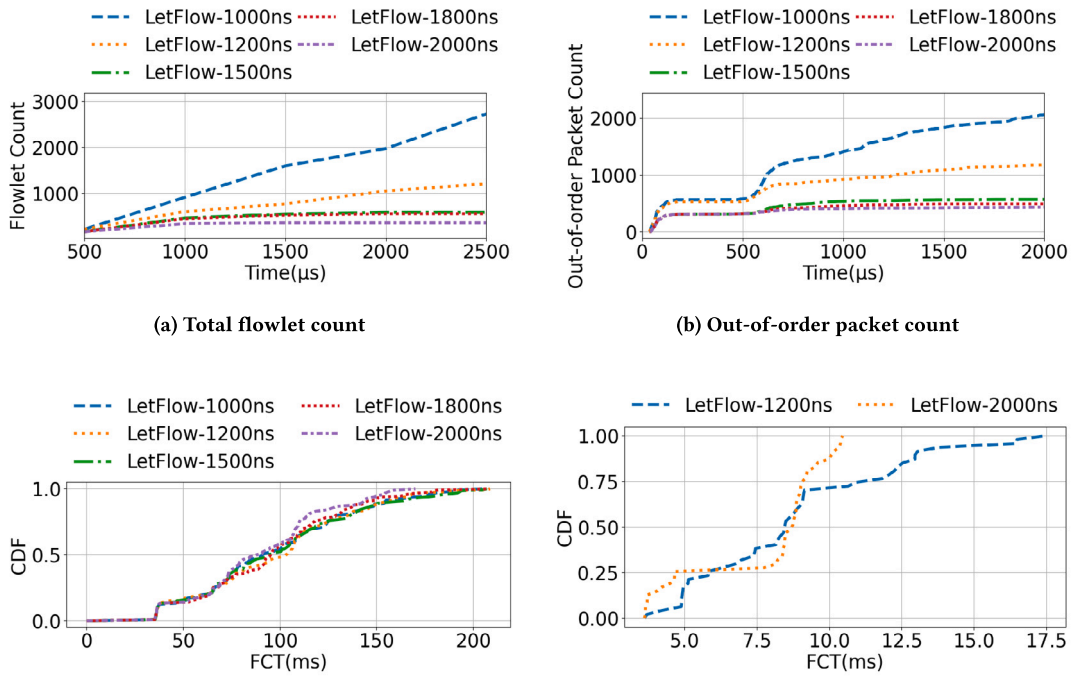
Rerouting decisions may provide more idle bandwidth but can also lead to more out-of-order packets. To improve the performance of short flows, load balancing schemes should balance these two factors, as emphasized by [25,33–35]. To analyze the relationship between packet reordering and rerouting frequency, we conducted simulations using an 8 × 8 leaf-spine topology, generating over 120 elephant flows based on the Allreduce communication pattern. In this scenario, since the network is lossless, out-of-order packets can only result from the rerouting decisions. From Fig. 4(a) and Fig. 4(b), we found that the amount of out-of-order generally increases as the number of flowlets increases. However, this does not necessarily seriously affect the elephant flow's FCT (shown in Fig. 4(c)) because a flow's throughput primarily depends on whether it still experiences hash collisions after the last rerouting decision. But for smaller flows (tens of megabytes), the impact of out-of-order packets is more pronounced, as shown in Fig. 4(d).

2.4. Flowlet timeout value is hard to choose

We observed that algorithms making routing decisions at the flowlet granularity have two phases following congestion events. In the first phase, rerouting decisions are frequent, leading to a substantial increase in the number of flowlets, as demonstrated in Fig. 4(a). In the second phase, there are almost no rerouting decisions. For example, there are no rerouting decisions after 0.5 ms in Fig. 3(b), and few new flowlets after 1 ms if the Flowlet Timeout Value (FTV) is larger than 1200 ns in Fig. 4(a).

We first analyze the rerouting decisions in the first phase. Consider a sender transmitting at 100 Gbps, with packet sizes of 1000 bytes: the smallest feasible FTV spans tens of nanoseconds, while the FTV for responding to congestion events ranges in the hundreds of microseconds. This broad range makes precise FTV tuning challenging. We do a simulation to test the actual performance. In order to identify flowlets more effectively, we adjust the congestion control algorithm to be more conservative (i.e., DCQCN's minimum rate, additive increase, and hyper-additive increase are adjusted very low).

In the first phase, rerouting events exhibit substantial variability, with minor parameter adjustments leading to significant changes in



(a) Total flowlet count

(b) Out-of-order packet count

(c) The FCT distribution of elephant flows

(d) The FCT distribution of smaller flows

Fig. 4. LetFlow's timeout value is hard to choose.

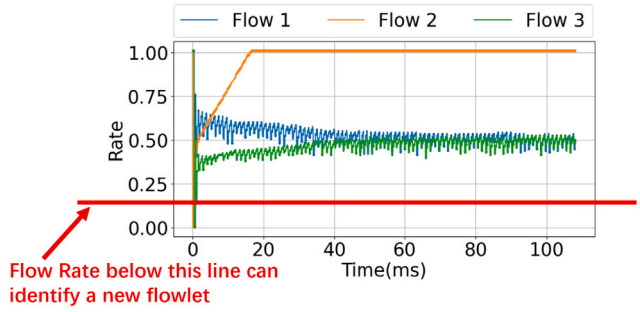


Fig. 5. Illustration of why no new flowlets are created after the hash collision. When the FTV is set to a relatively high value (e.g., 1.2 μs), new flowlets are not identified after a hash collision due to the inability of the mechanism to detect sufficiently long inter-packet gaps.

their frequency. As Fig. 4(a) shows, LetFlow with an FTV of 1000 ns generated over 2500 flowlets within a short 2.5 ms window. This behavior closely resembles packet spraying, undermining the advantages of flowlet-based rerouting, such as mitigating packet reordering while achieving fine-grained traffic load balancing. In contrast, LetFlow with a 2000 ns FTV identified almost no flowlets (a flat curve indicates no new flowlets are generated over time), effectively reverting to a flow-granularity load balancing mechanism. This behavior highlights the sensitivity of flowlet-based mechanisms to FTV configurations, particularly in high-bandwidth RDMA environments.

In the second phase, where no rerouting is triggered, a mechanism that performs rerouting with flowlet granularity might miss opportunities for rerouting. To prevent an excessive number of new flowlets, as shown in Fig. 4(a), we set the FTV to 1.2 μs. The flow rate diagram in Fig. 5 illustrates a scenario where flows experience conflicts during rerouting. When Flow 1 and Flow 2 encounter a hash collision at path 1, congestion prompts both flows to decrease their rates and generate new flowlets. The flowlet from Flow 1 randomly chooses to stay in its path. The flowlet from Flow 2 reroutes to path 2 and conflicts with Flow 3. Then, Flow 3 decreases the rate, and its flowlet randomly reroutes to path 1. Finally, Flow 1 and Flow 3 increase their speeds to share the bottleneck link bandwidth equally, and no new flowlets are generated, thus losing the opportunity to reroute to another better path.

The interrelationships of rerouting challenges are illustrated in Fig. 6. From our analysis of flowlet-based algorithms, we derive three key insights to address these challenges: (1) Coordinating Rerouting Decisions Across Multiple Sources. Independent rerouting decisions in scenarios with multiple sources often lead to hash conflicts at shared network links or switches. An effective rerouting strategy should account for these interactions, aiming for a globally optimized path selection to reduce congestion hotspots and improve load balancing. (2) Prioritizing Elephant Flows for Rerouting. Elephant flows, which are significant contributors to congestion, should be the primary target for rerouting strategies. This focus ensures that the performance of smaller flows, which are more sensitive to packet reordering, remains unaffected. (3) Simplifying Parameter Configuration. Effective rerouting strategies must rely on parameters that are easy to configure. For instance, the flowlet flow timeout value (FTV), a critical parameter in flowlet-based rerouting, is challenging to tune, particularly in high-bandwidth RDMA networks. Simplifying these parameters ensures consistent performance across varying network conditions and facilitates deployment.

3. Design and algorithms

Given the challenges discussed in Section 2, specifically the limitations of load balancing solutions in RDMA environments, we aim to achieve the following four design goals: (1) Accurately detect bottleneck and highly utilized links to ensure correct rerouting decisions; (2) Restrict the number of rerouted elephant flows on a single bottleneck link to prevent the herd effect; (3) Identify elephant flows and mouse flows and adopt different strategies for them to prevent out-of-order

Algorithm 1 Destination Switch Notifying Phase

Input:
linkTable ← the table that records the number of elephant flows on each link
CongestionTable ← This table is composed of key-value pairs. Key: Links where hash collisions occur. Value: flow ID.
t ← the tolerance for hash collisions of elephant flows

```
linkTable, highUtilLinks ← GetLinkStats(FlowletTable, PathTable, linkTable, t)
notificationPackets ← []
for congestionLink in congestionTable do
    init notify packet p
    p.flowId ← congestionTable[congestionLink].flowId
    p.congestionLink ← congestionLink
    p.highlyUtilizedLink ← highUtilLinks
    notificationPackets.add(p)
end for
Send notificationPackets
function GetLinkStats(FlowletTable, PathTable, linkTable, t)
    for entry in FlowletTable do
        if entry.elephantflow then
            pathId ← entry.pathid
            linkPath ← PathTable[pathId]
            for link in linkPath do
                linkTable[link] ← linkTable[link] + 1
            end for
        end if
    end for
    for link in linkTable do
        if linkTable[link] > t then
            highUtilLinks.append(link)
        end if
    end for
    return linkTable, highUtilLinks
end function
```

Algorithm 2 Source Switch Rerouting Phase

Input:
linkTable ← the table that records the number of elephant flows on each link
t ← the tolerance for hash collisions of elephant flows

```
linkTable, _ ← GetLinkStats(FlowletTable, PathTable, linkTable, t)
for packet in notificationPackets do
    pathids ← FindAvailablePaths(FlowletTable[packet.flowId].dst)
    for pathid in pathids do
        linkPath ← PathTable[[pathid]]
        if all link in linkPath has linkTable[link] < t then
            Increase the values in the linkTable for all links in pathLink by 1
            FlowletTable[flowId].pathid gets pathid
        end if
    end for
end for
```

delivery for mouse flows; (4) Ensure that parameters are robust.

To this end, we propose Reunion, an effective load-balancing mechanism designed to disperse elephant flows efficiently. It addresses hash collisions among elephant flows, which can significantly hinder large-scale model training jobs. Reunion operates through three key roles: source switch, intermediate switch, and destination switch, with each role executing distinct logic to enable dynamic and coordinated rerouting. In terms of parameters, Reunion only needs two parameters: (a) Statistical interval (s): the time interval for collecting flow statistics, and (b) Hash collision tolerance (t): the allowable threshold for hash collisions during rerouting. Note Reunion eliminates the need for precise FTV tuning, relying on flowlet FTV only as a fallback mechanism. More details are discussed in Section 4.2.

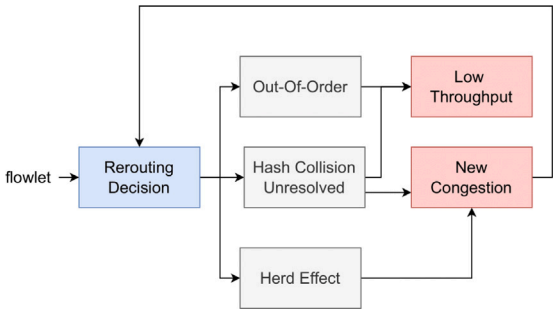


Fig. 6. Interrelationships of rerouting problems.

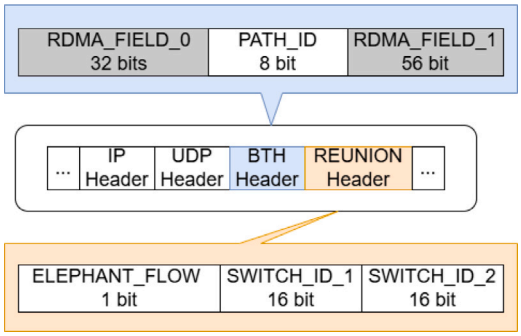


Fig. 7. Reunion packet header.

3.1. Header fields and parameters

In Reunion, each packet contains a PathID field and a Reunion header to enable path encoding and congestion management. The PathID field, situated in the RDMA BTH header, encodes the path a packet traverses through the network. For example, in a 2-tier leaf-spine network, the 8-bit PathID field can represent up to 256 distinct paths[1] [9]. The Reunion header includes the following three fields: (1) an elephant flow bit to indicate if a flow is an elephant flow or not; (2) two SwitchIDs that are used to indicate the first encountered bottleneck link in the path due to elephant flow hash collision. When the source switch processes packets, it employs the Count-Min-Sketch (CMS) algorithm [36] to determine if a flow qualifies as an elephant flow. If identified, the switch appropriately labels the packets using the elephant flow bit in the Reunion header.

The statistical time interval s is configured on both the source and destination switches and is used to probe and evaluate the network conditions within a specific time window.

The hash collision tolerance variable t indicates the maximum number of elephant flows that can be sent with hash collisions on a single link within the network in time interval s. If elephant flows should be transmitted at line rate, we recommend setting t to 1. A value other than one typically occurs in cases of large flow slicing, where an elephant flow is divided into two or more subflows and sent simultaneously.

[1] For 3-tier Clos networks, we can extend the Reunion header to carry a larger PathID field, allowing for the encoding of additional paths.

3.2. Elephant flow detection

Elephant flows are identified using the well-known Count-Min Sketch (CMS) [36] algorithm. CMS was initially proposed as a method for obtaining hotspots in Content Delivery Networks (CDNs), effectively identifying the top-k hot data items. During the training of large-scale machine learning models, various parallel strategies and synchronization mechanisms may still generate smaller flows of similar size. To address this, we employ CMS to filter out small flows.

The CMS algorithm enables switches to effectively detect large flows by leveraging a Bloom filter. When a packet arrives, the source switch applies multiple hash functions to its 5-tuple, calculating the corresponding table entries. The switch then increments the value in each of these entries. The switch retrieves the smallest value across all entries to determine whether a packet belongs to a large flow. If this value is among the largest observed, CMS classifies the packet as part of an elephant flow. However, CMS has a limitation: its accuracy decreases when the number of flows becomes excessively large, potentially leading to inaccurate classifications.

If there are m flows and each switch has k entries, with a total of a hash functions, the estimated frequency of a flow is given by $f_q = min(C_{j,h_j,(q)}), j \in [t]$. Assuming $a = log(1/\delta), k = 2/\epsilon$, when a and k are chosen with these values, the probability that the number of false positives exceeds $m * \epsilon$ is less than δ. For example, in a scenario with 20,000 flows, where each switch has 2000 flowlet table entries and uses 7 hash functions, the probability of more than 20 false positives per CMS calculation is less than 0.01. These false positives arise from conflicts between the entries of small flows and large flows in the CMS table. An increase in false positives reduces the accuracy of CMS identification results. The number of elephant flows transmitted by the source switch typically does not exceed 200 (capped by the number of GPUs connected to the switch), so calculated by the aforementioned formula, the false positives per instance will not exceed 1. We set the number of hot data items to be t (the slicing threshold in our scheme) times the number of uplinks. Under these conditions, all elephant flows are correctly identified during transmission.

Only the source switch executes the CMS algorithm. If the CMS result indicates that this is a large flow, then the elephant flow bit in the packet header is set to 1.

3.3. Hash collision detection phase

In this phase, the source switch and intermediate-hop switches monitor whether elephant flows transmitted on a link to their next hop are experiencing severe hash collisions. If such collisions are detected, the switches record the IDs of themselves and their next-hop switches (indicating a bottleneck link) into the packet header. As illustrated in Fig. 7, the Reunion header contains the switch's ID and its next-hop switch's ID, which specify a bottleneck link together. Note that this phase is not executed on the destination switch, as only the source and intermediate-hop switches are capable of identifying severe hash collisions that occur across flows destined for different destination switches. For received elephant flow data packets, events take place in the following order. (1) Before sending the packet to the next-hop switch, the switch detects: (a) The number of elephant flows on the same outgoing port exceeds the hash collision tolerance threshold t. (b) The switch at the current hop will write its own ID and the next-hop switch's ID into the packet header's bottleneck link field. However, if the link field in the packet header already has a value, the switch will not update the packet header because hash collisions occurring on earlier links in the path should be addressed first. (2) The destination switch receives the data packets. The destination switch updates the Bottleneck Link Table based on the packet: (a) The source switch is determined based on the Path ID. (b) The packet's bottleneck link and flow ID are written into the table. This process allows the destination switch to identify which links are bottlenecks due to hash collisions between elephant flows and to update the Bottleneck Link Table accordingly.

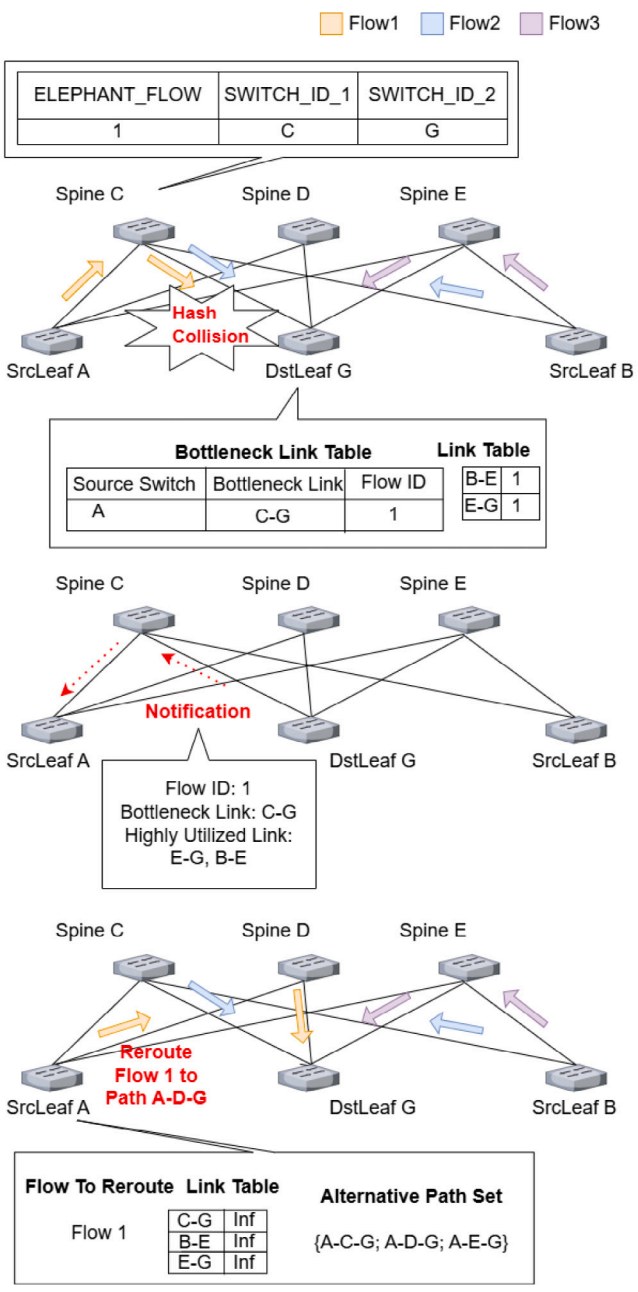


Fig. 8. The rerouting process of Reunion. The example of Reunion in a single flow case can be generalized to multiple flows.

3.4. High link utilization detection phase

In this phase, the destination switch determines which links should be avoided for rerouting.

For received elephant flow data packets, the destination switch performs the following actions: (1) Data packets reach the destination switch. (2) The destination switch updates its Link Table based on the incoming packets. Each row in the table has the Path ID in the first column, and the other columns represent the number of different elephant flows currently being transmitted on each link that the Path ID traverses. (3) During each statistical interval s, if the number of elephant flows on a link meets or exceeds the slicing threshold t, the destination switch considers that link highly utilized.

3.5. Notification phase

In this phase, the destination switch informs the source switch about current network conditions, specifically: (1) The ID of the flow selected for rerouting. (2) The specific link on which this flow is experiencing a hash collision. (3) The set of high-utilization links that should not be used for rerouting by the source switch.

At the end of each statistical interval s, the destination switch performs the following actions: (1) Construct a notification packet that includes: (a) The flow ID, which informs the source switch of the flow that should be rerouted. (b) The bottleneck link informs the source switch where the hash collision occurred for this flow. (c) A list of high utilization links informing the source switch which links should be avoided during rerouting. (2) Send the notification packet to the source switch, which is determined by calculating the Path ID. (3) Clear the Link Table to prepare for the next statistical interval.

Take Fig. 8 as an example: after the destination leaf switch G receives the congestion signal through the Reunion header from the congested hop (Spine C), it records the network state as outlined in Algorithm 1. During the statistical interval, the destination switch first calculates the number of elephant flows on each link. Then, it iterates Bottleneck Link Table to identify the links experiencing hash collisions. A random elephant flow is selected from one of these links, and the source switch of this flow is determined. In this example, destination switch G chooses Flow 1 and notifies source switch A that links (spine switch $E \rightarrow$ leaf switch G and leaf switch $B \rightarrow$ spine switch E) are not selectable. Each destination switch only selects one source switch to notify about a congested hop, thus avoiding the herd effect. Note that the selection of source switches at the bottleneck hop is random. Each packet from a congested link updates the Bottleneck Link Table at the destination switch. Before the notification phase begins, the last packet to arrive could be randomly from any source switch. This randomness also ensures that even if the selected source switch cannot change its path, other switches have the chance to alleviate hash collisions after the next notification phase.

3.6. Rerouting phase

In this phase, the source switch reroutes the elephant flows that have encountered hash collisions based on the information sent by the destination switch.

First, all elephant flows sent from the source switch must be recorded in the Flow Path Table. The source switch maintains a Link Table similar to the destination switch. Specifically, within the statistical interval s, the source switch performs the following actions: (1) Increases the count of elephant flows on the links traversed by the Path IDs of the elephant flows it sends out, based on the Path Table. (2) Upon receiving the notification packet, for the bottleneck links and high utilization links mentioned in the notification, the source switch marks the number of elephant flows on those links as infinite, indicating that these links are not selectable for subsequent rerouting. (3) Records the flow IDs that need to be rerouted as indicated by the notification packet.

When the statistical interval ends, for each flow that needs to be rerouted, the source switch finds the available Path IDs based on the destination switch ID it is supposed to send to. Then, (1) The source switch searches for an available Path ID. An available Path means it does not traverse any link with a flow count exceeding the slicing threshold. This excludes bottleneck and high-utilization links. (2) Once an available Path is found, the Link Table is updated to increase the elephant flow count on the traversed links. This prevents new hash collisions for other flows that need to be rerouted. The Flowlet Table is also updated to reflect the new Path ID chosen for the flow. (3) Finally, the Link Table is cleared to prepare for the next statistical interval.

Take Fig. 8 as an example: during the rerouting phase, leaf switch A updates the link status based on the information of the transmitted elephant flows and marks the links indicated by the destination switch G as not selectable. Finally, it chooses a path without hash collisions for rerouting.

By adopting this approach, Reunion can achieve optimality in a 2-tier Clos network topology,[2] ensuring that every rerouting decision is correct. This means that, in a 2-tier leaf-spine network topology, Reunion dynamically reroutes traffic in such a way that the congestion caused by hash collisions is minimized, leading to improved network performance. In a 3-tier Clos network topology, Reunion works by reducing the number of elephant flows experiencing hash collisions with each rerouting decision. This ensures that even in more complex network topologies, the mechanism continues to mitigate congestion effectively.

3.7. The impact of out-of-order packets

To assess the degree of out-of-order packets, consider the following scenario. The flow starts with the rate of r_1. Initially, it can be transmitted at maximum speed under ideal network conditions without requiring path changes. However, after the hash collision, the flow receives a congestion signal, triggering rerouting decisions. If the new path has more available bandwidth, the rerouted packets may arrive earlier than the original path, leading to out-of-order packet delivery.

The last packet that caused the rerouting decision made has the sequence number of S_i and was sent from the source server at t_0. The flow gets rerouted, whether the decision is made by a host or switch at time $t_0 + \delta t$, and the source decreases the speed, which causes the sending rate drop from r_1 to $k * r_1$. The first packet on the new path has a sequence number of S_j. Therefore, The maximum number of bytes out of order is S_j - S_i. The original path from server L_0 to server L_1 has a delay of t_1, and the reverse path has a delay of t_1'; the new path from server L_0 to server L_1 has a delay of t_2, and the reverse path has the delay of t_2'. The maximum number of the bytes delivered out-of-order can be calculated as Eq. (1):

$$\begin{cases} S_j - S_i = & r_1 \Delta t + r_2(t_2 + t_2') \\ t_1 = & \Delta t + t_2 \\ r_2 = & k * r_1 \end{cases} \tag{1}$$

When congestion is detected, the source server reduces its rate based on its congestion control algorithm (CCA), which helps mitigate congestion quickly. Our simulation, with a 12 MB switch buffer, 100 Gbps links, and an elephant flow size of 2 GB, demonstrates that the impact of rerouting on packet order is minimal (< 0.1%). This is due to the fact that Reunion only reroutes an elephant flow at most once in each time interval s, and the average number of rerouting events across different load scenarios and parameter settings is low. Consequently, the proportion of out-of-order packets caused by rerouting is negligible. Furthermore, the rate of the elephant flow converges rapidly, as illustrated in Fig. 5. This ensures that the performance degradation due to out-of-order delivery remains minimal.

4. Evaluation

We use NS-3 simulations to evaluate Reunion's performance and compare it against flowlet-based rerouting mechanisms, including Conga, LetFlow, and ConWeave. The simulations utilize an 8 × 8 leaf-spine topology with a 1:1 oversubscription ratio, where each link had a 100 Gbps capacity and 1 μs latency. Parameters for each scheme are carefully tuned to their optimal settings. Since DCQCN's default parameters are not suitable for a scale of over a hundred nodes, we adjust $(K_{min}, K_{max}, P_{max})$ to (100KB, 400 KB, 0.2). The switch buffer size is set to 12 MB, and PFC is enabled with a dynamic threshold as suggested

[2] It has been shown in HPN [2] that a 2-tier Clos network is already sufficient to ensure communication across ten thousand GPU cards.

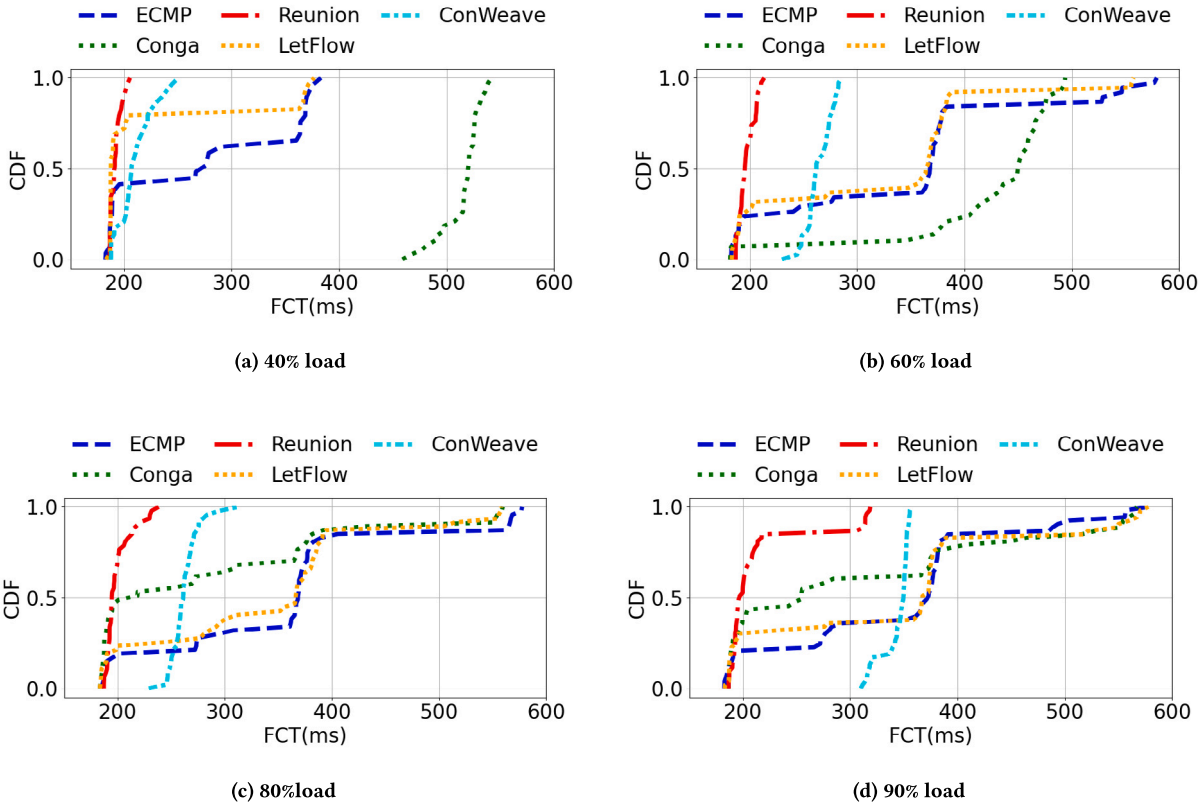


Fig. 9. FCT under different network traffic loads.

in [9]. Unless otherwise specified, we set Reunion's parameter t to 1 and parameter s to 1000 μs. We implement a Ring Allreduce communication pattern based on the traffic characteristics outlined in Table 1, where the size of the simultaneously communicating elephant flows is set to 2 GB. For the evaluation, we select hosts located across different racks to initiate communication. The network load is varied to analyze the behavior and performance under diverse traffic conditions. We focus on improving overall communication efficiency, especially tail latency. Therefore, we use the FCT (Flow Completion Time) distribution of elephant flows to characterize the performance and use tail FCT as a key metric for comparison.

4.1. Reduction on tail FCT

Fig. 9 demonstrates the effectiveness of Reunion in reducing tail FCT under varying network loads. In Fig. 10, we evaluate the performance of ECMP, Reunion, ConWeave, Conga, and LetFlow by measuring their maximum FCTs. We observe that Reunion completes the slowest flows at least 10.9%–25.2% faster than other schemes for diverse network loads between 40% to 90%. This performance is attributed to Reunion's ability to quickly converge to disjoint paths, minimizing congestion and ensuring more efficient load balancing.

At a network load of 90%, Reunion cannot guarantee near-optimal FCTs for all flows, with approximately 20% of flows experiencing FCTs that are twice the optimal value. This limitation arises because, under such high load conditions, no source switch can identify idle paths for rerouting, as illustrated in Fig. 11.[3] This scenario typically occurs when the load nears saturation, where the average load of 90% allows for instantaneous peaks reaching 100%. In all other cases, Reunion

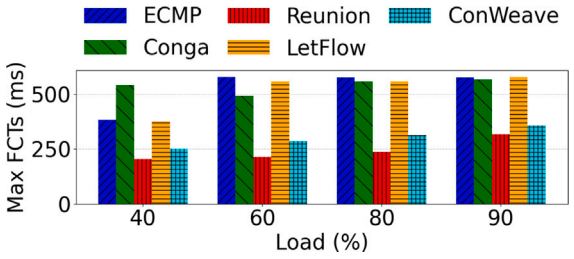


Fig. 10. FCTs comparison.

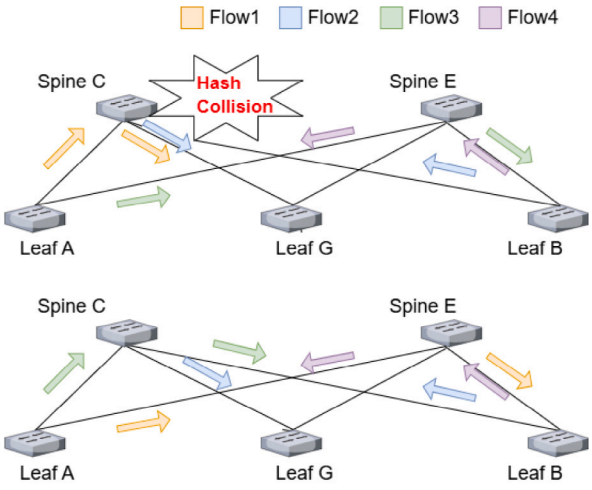


Fig. 11. When there are few alternative paths, Reunion may not be able to find a perfect path for rerouting.

[3] Unless the paths taken by Flow 1 and Flow 3 are swapped, there will be no available paths to reroute and avoid the hotspot at Spine C.

effectively identifies alternative paths to mitigate congestion.

We find that under high-load conditions, ConWeave exhibits a smaller FCT range compared to other schemes, with tail latency only 15.2% higher than its minimum FCT. This is because ConWeave continuously reroutes all flows experiencing congestion, and flows without hash collisions may experience hash collisions after other flows are rerouted, leading to similar FCTs. In contrast, Reunion, Conga, and LetFlow exhibit more step-like FCT distributions. For Reunion, this pattern is due to its targeted rerouting mechanism: destination switches only notify source switches about flows with hash collisions, ensuring that uncongested flows maintain their paths. Once Conga and LetFlow complete their rerouting convergence, they similarly avoid further path changes. Even under these conditions, Reunion still performs slightly better than ConWeave (by 10.9%). This advantage is attributed to Reunion's strategy of marking paths as idle once transmissions for collision-free flows are completed. These idle paths are subsequently utilized for rerouting, accelerating the completion of remaining flows, and improving overall efficiency.

4.2. Robustness

To test Reunion's robustness, we evaluate the impact of varying the statistical interval parameter (s) on the frequency of rerouting decisions and their impact on Flow Completion Time (FCT). As shown in Fig. 12, the number of rerouting events positively correlates with s. Smaller s values do not always result in faster convergence because rerouting depends on whether the destination switch can designate a source switch capable of finding a suitable path. This improves only when the designated source successfully reroutes. Nearly all rerouting decisions are completed within 30 ms of the elephant flow's transmission when s is set to 2500 µs, and within 6 ms when s is 250 µs. Across different parameter settings, Reunion achieves a tail FCT improvement of 61.9% to 64.7% compared to ECMP.

Furthermore, unlike LetFlow, which only relies on FTV for rerouting, the FTV of Reunion is only used as a fallback mechanism so that it can be set to a relatively large value (such as the 500 µs recommended in LetFlow [7]).

4.3. When allowing out-of-order packets

The analysis above shows that an increased number of flowlets can lead to more out-of-order packet delivery. A natural question arises: if support for out-of-order packets is enhanced, could this improve the performance of Conga and LetFlow? To explore this, we conducted simulations where we disabled PFC (Priority Flow Control) and employed IRN's SACK (Selective Acknowledgment) mechanism, which is designed to handle out-of-order packets, as described by Mittal et al. in their study on revisiting congestion control mechanisms for RDMA networks [37]. Our results show that while the average tail latency for all mechanisms increased with this modification, the increase was relatively modest–less than 10%. Despite this adjustment, Reunion continues to outperform other mechanisms, as illustrated in Fig. 13. This suggests that even with enhanced support for out-of-order packets, Reunion's rerouting approach remains more effective at minimizing tail latency.

4.4. Compatibility with flow slicing

To prevent hash collisions caused by elephant flows, an effective strategy is to reduce the probability of collisions by slicing the flows. Assume that k flows from different servers in one rack compete for b core links. Each path can have p flows transmitting at link rate r, which means the core link rate is $p * r$ when the oversubscription ratio is 1 in the leaf-spine network. The ideal case is that each path is chosen by at most $max(1, k/b)$ flow slices. However, if one path is selected by c slices, where c exceeds p, each flow slice gets a fair share of $r * p/c$. The mean

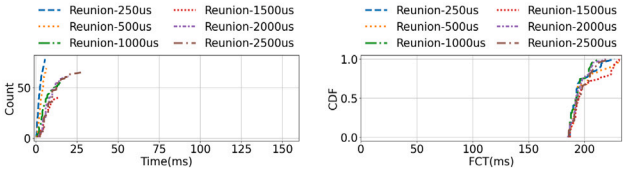


Fig. 12. Reunion robustness. If all flows are sent at full speed, the FCT is 160 ms. All rerouting decisions occur within the first 30 ms for Reunion with different parameters, and the final FCT distribution does not vary significantly.

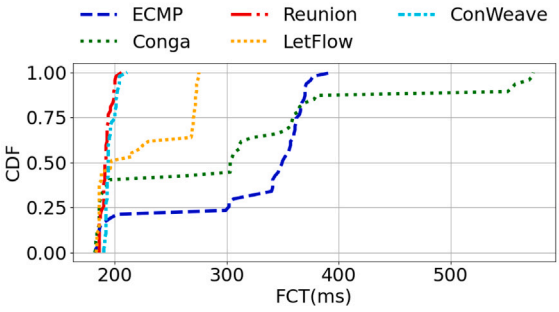


Fig. 13. FCT with IRN SACK support.

speed of a flow can thus be calculated using the formula in Eq. (2):

$$E(rate) = \frac{\sum_{i=0}^{p-1} \binom{k-1}{i} * r + \sum_{i=p}^{k-1} \binom{k-1}{i} * \frac{p}{i} * r}{b^{k-1}} \quad (2)$$

We conduct experiments by slicing elephant flows to assess the impact of flow slicing on the FCT of elephant flows. In this study, the load balancing algorithm treats each sliced flow as a separate entity. Therefore, the hash collision tolerance parameter t in Reunion is set to the corresponding number of slices. Flow slicing allows the subflows of a sliced flow to be transmitted simultaneously, so the highest FCT among the subflows determines the FCT of the elephant flow to which they belong. We observe that the number of rerouting events in Reunion nearly doubled, but the tail latency of the flows was similar to the unsliced case. As shown in Fig. 14, the results for slicing flows into two, three, and four segments are displayed in the left, middle, and right sections, respectively. For flow slicing into three or four segments, ConWeave achieves tail latency around 3% lower than Reunion. This improvement occurs because the large, bursty elephant flows are divided into smaller sub-flows, making the traffic distribution more uniform. However, this also puts additional pressure on ConWeave's reordering mechanism. The performance of LetFlow and Conga improves as more slices are created, which leads to reduced tail FCT. For example, slicing the flow into 2 to 4 segments can improve Conga's performance by at least 39.2%. For ECMP, the performance can be enhanced by 25.5%. For LetFlow, the improvement is 10.1%. However, increasing the number of slices does not yield additional benefits for ECMP and LetFlow. This is because splitting flows into more slices results in more sub-flows being transmitted simultaneously. While this improves load distribution, it also exacerbates queue buildup due to micro-bursts, offsetting the benefits of increased granularity. Consequently, dividing flows into too many slices does not necessarily enhance performance further.

5. Conclusion

This paper introduces Reunion, a receiver-driven flowlet-level traffic load-balancing scheme designed to address the unique challenges of AI training clusters. Reunion leverages real-time congestion information

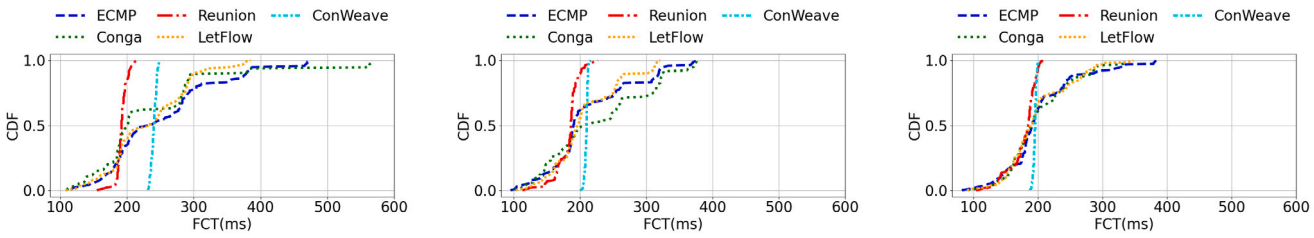


Fig. 14. Reunion with flow slicing. When the number of slices is 2, 3, or 4, the performance of Reunion shows little variation and significantly outperforms LetFlow, Conga, and ECMP.

collected in the switch dataplane to enable destination switches to notify and coordinate source switches to reroute elephant flows based on network dynamics. This approach effectively resolves elephant flow hash collisions while preserving the performance of small flows and minimizing packet reordering. Reunion's lightweight design makes it deployable on existing switch hardware with minimal modifications. Our simulation results demonstrate that Reunion effectively reduces tail FCTs by 10.9% to 62.1% compared to LetFlow, ECMP, Conga, and ConWeave under high network load in realistic workload patterns.

CRediT authorship contribution statement

Mingyao Wang: Writing – review & editing, Writing – original draft, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Keqiang He:** Writing – review & editing, Supervision. **Peirui Cao:** Writing – review & editing. **Jiong Duan:** Software, Formal analysis, Data curation. **Dongliang Lv:** Supervision, Resources, Funding acquisition. **Zehao Yu:** Supervision, Resources, Funding acquisition. **Yanqing Chen:** Writing – review & editing, Formal analysis. **Chengyuan Huang:** Writing – review & editing. **Wanchun Dou:** Writing – review & editing. **Guihai Chen:** Writing – review & editing. **Chen Tian:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This research is supported by the National Natural Science Foundation of China under Grant Numbers 62325205 and 62172204, the Nanjing University-China Mobile Communications Group Co.,Ltd. Joint Institute, the Fundamental Research Funds for the Central Universities, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

Data availability

The authors are unable or have chosen not to specify which data has been used.

References

[1] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, Noah Fiedel, PaLM: Scaling language modeling with pathways, 2022, http://dx.doi.org/10.48550/ARXIV.2204.02311.

[2] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, Dennis Cai, Alibaba HPN: A data center network for large language model training, in: Proceedings of the ACM SIGCOMM 2024 Conference, in: ACM SIGCOMM '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 691–706, http://dx.doi.org/10.1145/3651890.3672265.

[3] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, Xin Liu, MegaScale: Scaling large language model training to more than 10,000 GPUs, 2024, arXiv:2402.15627.

[4] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, Hongyi Zeng, RDMA over ethernet for distributed training at meta scale, in: Proceedings of the ACM SIGCOMM 2024 Conference, in: ACM SIGCOMM '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 57–70, http://dx.doi.org/10.1145/3651890.3672233.

[5] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al., Rdma over ethernet for distributed training at meta scale, in: Proceedings of the ACM SIGCOMM 2024 Conference, 2024, pp. 57–70.

[6] Leah Shalev, Hani Ayoub, Nafea Bshara, Erez Sabbag, A cloud-optimized transport protocol for elastic and scalable HPC, IEEE Micro 40 (6) (2020) 67–73.

[7] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, Tom Edsall, Let it flow: Resilient asymmetric load balancing with flowlet switching, in: NSDI, vol. 17, 2017, pp. 407–420.

[8] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al., CONGA: Distributed congestion-aware load balancing for datacenters, in: Proceedings of the 2014 ACM Conference on SIGCOMM, 2014, pp. 503–514.

[9] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, Mun Choon Chan, Network load balancing with in-network reordering support for RDMA, in: Proceedings of the ACM SIGCOMM 2023 Conference, in: ACM SIGCOMM '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 816–831, http://dx.doi.org/10.1145/3603269.3604849.

[10] Huasha Zhao, John Canny, Kylix: A sparse allreduce for commodity clusters, in: 2014 43rd International Conference on Parallel Processing, IEEE, 2014, pp. 273–282.

[11] Amith R. Mamidala, Jiuxing Liu, Dhabaleswar K. Panda, Efficient barrier and allreduce on infiniband clusters using multicast and adaptive algorithms, in: 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No. 04EX935), IEEE, 2004, pp. 135–144.

[12] Alexander Sergeev, Mike Del Balso, Horovod: fast and easy distributed deep learning in TensorFlow, 2018, arXiv:1802.05799.

[13] Rajeev Thakur, Rolf Rabenseifner, William Gropp, Optimization of collective communication operations in MPICH, Int. J. High Perform. Comput. Appl. 19 (1) (2005) 49–66, http://dx.doi.org/10.1177/1094342005051521.

[14] Yibo Zhu, Monia Ghobadi, Vishal Misra, Jitendra Padhye, ECN or delay: Lessons learnt from analysis of DCQCN and TIMELY, in: Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies, CoNEXT '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 313–327, http://dx.doi.org/10.1145/2999572.2999593.

[15] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, Fabien 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, 2014, pp. 149–160.

[16] Sébastien Barré, Christoph Paasch, Olivier Bonaventure, MultiPath TCP: from theory to practice, in: Proceedings of the 10th International IFIP TC 6 Conference on Networking - Volume Part I, NETWORKING '11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 444–457.

[17] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, Mosharaf Chowdhury, Resilient datacenter load balancing in the wild, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017, pp. 253–266.

[18] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, Jennifer Rexford, Clove: Congestion-aware load balancing at the virtual edge, in: Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies, 2017, pp. 323–335.

[19] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al., Hedera: dynamic flow scheduling for data center networks, in: Nsdi, vol. 10, (no. 8) San Jose, USA, 2010, pp. 89–92.

[20] C. Hopps, Analysis of an Equal-Cost Multi-Path Algorithm, RFC, RFC Editor, 2000.

[21] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, Amin Vahdat, WCMP: weighted cost multipathing for improved fairness in data centers, in: Proceedings of the Ninth European Conference on Computer Systems, EuroSys '14, Association for Computing Machinery, New York, NY, USA, 2014.

[22] Advait Dixit, Pawan Prakash, Y Charlie Hu, Ramana Rao Kompella, On the impact of packet spraying in data center networks, in: 2013 Proceedings IEEE INFOCOM, IEEE, 2013, pp. 2130–2138.

[23] Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, Amin 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, 2017, pp. 225–238.

[24] Jiaxin Cao, Rui Xia, Pengkun Yang, Chuanxiong Guo, Guohan Lu, Lihua Yuan, Yixin Zheng, Haitao Wu, Yongqiang Xiong, Dave Maltz, Per-packet load-balanced, low-latency routing for clos-based data center networks, in: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT '13, Association for Computing Machinery, New York, NY, USA, 2013, pp. 49–60, http://dx.doi.org/10.1145/2535372.2535375.

[25] Jiaqing Dong, Lijuan Tan, Chen Tian, Yuhang Zhou, Yi Wang, Wanchun Dou, Guihai Chen, MEET: rack-level pooling based load balancing in datacenter networks, IEEE Trans. Parallel Distrib. Syst. 33 (12) (2022) 3628–3639.

[26] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, Aditya Akella, Presto: Edge-based load balancing for fast datacenter networks, in: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 465–478, http://dx.doi.org/10.1145/2785956.2787507.

[27] Yi Wang, Ya-nan Jiang, Qiufang Ma, Chen Tian, Bo Bai, Gong Zhang, RDMA load balancing via data partition, in: 2019 28th International Conference on Computer Communication and Networks, ICCCN, 2019, pp. 1–8, http://dx.doi.org/10.1109/ICCCN.2019.8847077.

[28] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, Jennifer Rexford, Hula: Scalable load balancing using programmable data planes, in: Proceedings of the Symposium on SDN Research, 2016, pp. 1–12.

[29] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, Abdul Kabbani, PLB: Congestion signals are simple and effective for network load balancing, in: Proceedings of the ACM SIGCOMM 2022 Conference, SIGCOMM '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 207–218, http://dx.doi.org/10.1145/3544216.3544226.

[30] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, Randy Katz, DeTail: Reducing the flow completion time tail in datacenter networks, in: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 2012, pp. 139–150.

[31] Dan Li, Du Lin, Changlin Jiang, Lingqiang Wang, SOPA: Source routing based packet-level multi-path routing in data center networks, ZTE Commun. 16 (2) (2018) 42–54.

[32] Weibo Cai, Shulin Yang, Gang Sun, Qiming Zhang, Hongfang Yu, Adaptive load balancing for parameter servers in distributed machine learning over heterogeneous networks, ZTE Commun. 21 (1) (2023) 72–80.

[33] Sen Liu, Yongbo Gao, Zixuan Chen, Jiarui Ye, Haiyang Xu, Furong Liang, Wei Yan, Zerui Tian, Quanwei Sun, Zehua Guo, Yang Xu, Halflife: An adaptive flowlet-based load balancer with fading timeout in data center networks, in: Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 66–81, http://dx.doi.org/10.1145/3627703.3650062.

[34] Guo Chen, Yuanwei Lu, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Thomas Moscibroda, Mp-rdma: enabling rdma with multi-path transport in datacenters, IEEE/ACM Trans. Netw. 27 (6) (2019) 2308–2323.

[35] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, Thomas Moscibroda, Multi-path transport for {RDMA} in datacenters, in: 15th {USENIX} Symposium on Networked Systems Design and Implementation, {NSDI} 18, 2018, pp. 357–371.

[36] Graham Cormode, S. Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, J. Algorithms 55 (1) (2005) 58–75, http://dx.doi.org/10.1016/j.jalgor.2003.12.001.

[37] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, Scott Shenker, Revisiting network support for RDMA, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 313–326.