LEFT: LightwEight and FasT packet Reordering for RDMA

Peihao Huang [†]
Central South University &
Hunan University

Xin Zhang
Hunan University

Zhigang Chen [‡]
Central South University

Can Liu
Hunan University

Guo Chen [‡]
Hunan University

ABSTRACT

RDMA, as a cutting-edge networking technology, has gained extensive adoption in large-scale data centers due to its exceptional characteristics, such as low and stable latency, high throughput and low CPU utilization. However, due to the limited on-chip memory of the RDMA Network Interface Cards (RNIC), commercial RDMA usually only supports single-path transmission and cannot fully utilize the rich parallel paths within the DCN, resulting in insufficient bandwidth utilization. Multipath transmission can improve bandwidth utilization, but the out-of-order (OoO) packets it brings negatively impacts the performance of RNICs. Recent works have attempted to address this issue by using bitmaps to record OoO packets to better support multipath transmission. However, these approaches either consume excessive memory for maintaining bitmaps, leading to poor connection scalability, or introduce high latency in bitmap sharing. Consequently, implementing efficient packet reordering in RDMA remains a challenge.

To this end, we propose LEFT, a fast and lightweight RDMA packet reorder. It is fast due to its ability to process packet reordering at speeds exceeding 200Gbps on the RNIC, and it is scalable by introducing enhanced dual-state shared bitmap schemes, thereby consuming minimal memory even under high concurrency. Specifically, LEFT adopts a fast and slow path of packet reordering to reduce latency when dealing with multipath RTT. Simulation evaluation shows that LEFT maintains a throughput of 94%+ even when the multi-path RTT difference is 32 times, which is 180% higher than the throughput achieved by using an ordinary shared bitmap pool. By adding an average of only 7B of extra on-chip states for each connection, LEFT achieves up to 1.54x higher throughput while reducing 99% tail FCT by 29% compared to single path transmission.

CCS CONCEPTS

• **Networks** → **Transport protocols**;

KEYWORDS

data centers, network architecture, network hardware, transport layer

ACM Reference Format:
Peihao Huang [†], Xin Zhang, Zhigang Chen [‡], Can Liu, and Guo Chen [‡] . 2024. LEFT: LightwEight and FasT packet Reordering for RDMA. In *The 8th Asia-Pacific Workshop on Networking (APNet 2024), August 3–4, 2024, Sydney, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3663408.3663418

1 INTRODUCTION

Remote Direct Memory Access (RDMA) enables end hosts to exchange data directly within remote memory while relieving the CPU by offloading the network stack to the network interface card hardware. The characteristics of low latency, high throughput, and bypass CPU through RNIC hardware leading to RoCEv2 [2] as the widespread presence of RDMA in data centers [6, 13, 20, 22, 27].

Data center networks (DCNs) have abundant parallel paths internally, which are balanced to provide high throughput [19]. However, due to limited on-chip memory, RDMA only supports single-path transmission, and numerous studies have shown it cannot fully utilize the network bandwidth [3, 4, 14, 26].

Many works have been proposed to address the shortcomings of single-path transmission [7, 12, 14], but result in massive amounts of OoO packets, due to the switch queue length and latency [10, 12, 24]. However, the current RDMA has low tolerance for OoO packets. Once received, they will trigger retransmission and be processed by the go-back-N (GBN) mechanism [8, 9], which has low overhead but poor performance [21, 23, 27].

The fast and scalable recording and processing of OoO packets on RNICs are crucial. Bitmaps have been introduced as a mechanism for reordering OoO packets [7, 21, 23, 27], but finding the right balance between performance and scalability remains a challenge. Specifically, IRN [23] requires adding 5 bandwidth delay product (BDP) size bitmaps (*e.g.*, ~500B) to each connection [7, 23]. It significantly increases the memory footprint of each connection on the RNIC by about 2x, thus reducing the scalability of the RNIC. SRNIC [27] addresses the issue by moving the bitmap to the host and recording it through software. Nevertheless, factors such as PCIe latency (~2us) and software instability slow down the hardware pipeline and reduce the throughput performance of the RNIC. MELO [21] utilizes a fixed-size shared bitmap to meet the needs of all connections and increases or decreases the bitmap size allocated by the connection in a linked list structure. While it achieves scalability, it fails to keep speed due to the lack of consideration for

[†]: This work was done when Peihao Huang worked as a visiting student at Hunan University.
[‡]: Corresponding author.

APNet 2024, August 3–4, 2024, Sydney, Australia

ACM ISBN 979-8-4007-1758-1/24/08.
https://doi.org/10.1145/3663408.3663418

the arrival patterns of OoO packets caused by multi-path scenarios. Bitmap latency is directly proportional to the degree of connection disorder, making it difficult to stabilize at smaller values.

We have observed that shared bitmaps are a promising approach but are not suitable for connections with large bitmaps, while ordinary bitmaps offer low latency but result in shallow memory usage for connections with small bitmaps. Therefore, we raise the following question: Is it possible to combine the advantages of shared bitmap and ordinary bitmap methods to achieve both fast and scalable OoO packet reordering?

In this paper, we propose LEFT, a LightwEight and FasT packet Reordering for RDMA. LEFT maintains comparable "Fast" performance to ordinary bitmaps while providing memory performance as "Lightweight" as shared bitmaps.

First, LEFT enhances the shared bitmap by designing a dual-state shared bitmap with a cache pool, where large bitmaps are cached to reduce the access latency of the bitmap pool. Second, LEFT employs packet gathering techniques to reduce inter-connection packet reordering and lower cache miss probabilities, further improving the speed of shared bitmaps. Lastly, LEFT uses a scheduler to make packets with minimal reordering bypass gathering, reducing the frequency of conflicts in the gather queue and enhancing the gathering efficiency. This approach also ensures that the tail latency of small flows remains unaffected.

We have implemented LEFT in the NS3 simulations with HPCC. Our experiments show that LEFT achieves 1.4 times the application throughput compared to single-path transmission while reducing the 99% tail flow completion time (FCT) by 3.11 times compared to go-back-N. Furthermore, LEFT only adds 7B per-connection state compared to GBN.

2 BACKGROUND AND MOTIVATION

2.1 Needs for OoO packet reorder

Per-flow insufficient utilization of bandwidth: Current RDMA uses a single network path to transmit connection messages, which limits the utilization of the overall network bandwidth. Equal-cost class multipath (ECMP) routing is the primary method used to balance RDMA traffic in current data center network architectures [26, 44, 45]. Essentially, ECMP hashes different connections to different paths. However, as highlighted in previous studies [15, 26], ECMP suffers from traffic imbalance due to hash collisions, resulting in lower overall network utilization.

We use the same topology in §4.1, set one group to send traffic to another group, and measured the throughput of the connections. Observation from Fig.1 shows that single-path transmission fails to fully utilize the path bandwidth, resulting in only 50% throughput. Therefore, spreading traffic across multiple paths at a finer granularity than flows is crucial to achieving high network utilization.

OoO packet degrade performance: Several works have proposed the use of fine-grained transmission to effectively leverage the abundant network paths in DCN [12, 14, 16, 25, 26]. Unfortunately, most of these prior studies did not consider the impact of OoO packets on RDMA performance, or only apply to fat-tree topologies [25]. Meanwhile, RDMA is highly sensitive to reordering. When the RNIC receives an OoO packet, it indicates packet loss and immediately triggers loss recovery, resulting in a decrease in the sending RNIC's transmission rate.

As shown in Fig.1, under per-packet granularity multi-path transmission, the proportion of OoO packets received at the receiver increases significantly when the load exceeds 0.3. The throughput rate of the GBN dropped to less than 10%, and the selective retransmission (SR) also reduced the throughput rate.

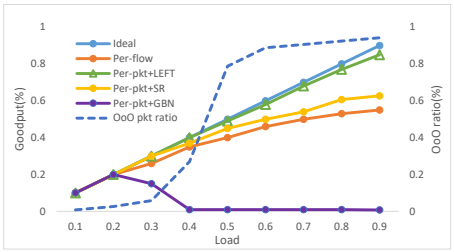

Figure 1: The impact of OoO packets on throughput.

2.2 Reorder needs fast and scalable

Bitmap brings new matters: Recent research attempts to record the OoO packets by bitmaps [7, 21, 23, 27]. Each connection requires 5 BDP-sized bitmaps (one bitmap requires 500 slots to accommodate the BDP for a network with a bandwidth of 200 Gbps and an RTT difference of 2 to 20 us across different paths [1]). So, for 5K connections, these bitmaps consume 1.5 MB of memory. In scenarios with larger RTTs, such as cross-DCN communications and WAN RDMA, the overhead of bitmaps becomes more significant. But the on-chip memory in the NIC is small and expensive, usually around 2 MB [15, 27].

The shared bitmap pool can reduce the bitmap overhead to a few kilobytes [21]. Each connection dynamically allocates a bitmap and links them together using a linked list data structure. However, additional delays will be introduced when OoO packets need to access the middle part of the linked list. As shown in §4.3, this latency becomes a performance bottleneck for the RNIC when the pipeline has to wait for the bitmap to return the latest expected next receive packet (RCV_NXT).

Bitmap needs scalable: In RNIC, the number of active connections can reach thousands [11], while the available memory resources are scarce. Therefore, the RNIC requires a scalable OoO reorder to accurately handle OoO packet records for thousands of concurrent connections while satisfying hardware memory resource constraints.

Bitmap needs fast: Efficiency is crucial when it comes to the sorting speed of the reordering. Slow sorting can result in a backlog of incoming packets waiting to be processed by the RNIC, exacerbating network congestion [5]. The speed of the reordering process needs to be faster than the arrival rate of packets. Ideally, all operations should converge within 40 ns (when receiving a 1 KB packet at a 200 Gbps bandwidth).

2.3 Challenges and intuition

LEFT on RNIC needs to address the following key challenges to be both fast and scalable.

How to ensure that bitmaps meet low memory requirements while maintaining fast processing speed? Sharing bitmap pool is a good way to achieve scalability, but it needs to be faster. LEFT improves upon the shared bitmap pool by introducing a dual-state bitmap pool with caching and pooling capabilities. This enhancement allows the connection bitmaps to switch between two states, which is both fast and lightweight.

How to meet the low-latency requirements of almost unlimited connection bitmaps with a limited number of caches? LEFT sacrifices latency to gather packets with heavy bitmap usage, reducing OoO packets inter different connections. This helps reduce the probability of bitmap cache misses and improves overall efficiency.

How to efficiently gather packets of almost unlimited connections using a limited gathering queue? LEFT observed that a small number of large flows occupy the majority of the bandwidth, making them more susceptible to significant reordering. On the other hand, small flows, even if they experience reordering, can generally meet their speed requirements without utilizing bitmap caching.

LEFT introduces a scheduler that implements a fast-slow path design in light of this. The gather is bypassed for low-latency bitmaps and non-OoO packets, thereby avoiding any additional completion time. However, for high-latency packets, the gather is directed to the module. This approach enables the alleviation of gathering queue conflicts and packet aggregation even with a small number of gather queues.

3 LEFT DESIGN

3.1 Overview

As shown in Fig.2, LEFT is implemented in the transport layer of the RNIC. The blue arrows in the diagram represent the data path, where OoO packets are first DMAed to the corresponding connection's reordering buffer and then submitted to the application after the out of order is restored. In the control plane, routing is determined based on the degree of packet reordering. The solid and dashed red lines represent the fast and slow paths, respectively. Packets on the slow path need to pass through the gather module to reduce the degree of disorder between flows, which will add at most a few us of gather delay.

There are four modules that cooperate to achieve low-latency OoO packet reordering.

- Reorder logic: Handles the logical reordering aspect of the packet rearrangement process, where the OoO packets no longer trigger retransmissions.
- Dual-state bitmap pool: Maintains a dual-state bitmap that is shared by all connections and dynamically assigns bitmaps to record the reordering status for each connection.
- Gather queue: Aggregating packets from different connections, reducing inter-connection disorder, improving bitmap cache hit rate, and reduced bitmap latency.
- Scheduler: The scheduler selects the path for incoming packets based on the degree of packet reordering, ensuring that the latency of large bitmaps' connections packets converges within 40ns. At the same time, the aggregation latency does not affect the connection packets of small bitmaps or those without bitmaps.

In the following subsection, we zoom into each design component and elaborate on how they can achieve high performance with a small on-chip memory footprint together.

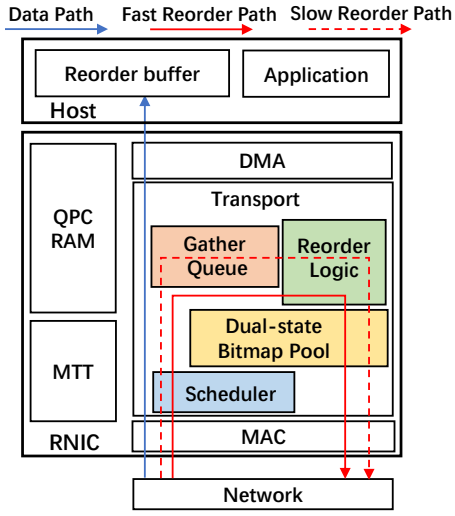

Figure 2: LEFT Overview

3.2 Dual-state bitmap pool

As shown in Fig.3, the connection bitmap in LEFT has two states: bitmap in pool and bitmap in cache.

Share block bitmap: The bitmap pool is divided into equally sized blocks, and these blocks are dynamically allocated to connections based on the degree of disorder. Each bitmap block is connected by the Nxt_ptr pointer, which indicates the address of the next bitmap block. The available block bitmap indicates which bitmaps have been allocated and which are available for allocation. It is initialized with all 0, indicating that all bitmap blocks are available.

Bitmap cache: An ordinary ring bitmap composed of registers, consisting of several banks, used to compensate for the high latency of random access to linked bitmap blocks.

Bitmap manager: Upon receiving OoO packets, the reorder module sends the start address of the connection bitmap along with the offset of the OoO packet to the bitmap manager. The bitmap manager first checks if the bitmap is present in the cache. If it is found in the cache, the bitmap manager updates the bitmap directly in the cache. However, if there is a cache miss, the bitmap manager determines the number of bitmap blocks to traverse and calculates the corresponding latency: T_{pool}. If the latency T_{pool} is below the upper limit: T_{max}, the bitmap is accessed in the bitmap pool. Otherwise, the bitmap manager needs to swap the current bitmap with the cache that has the lowest occupancy before proceeding. The pseudo-code for bitmap updating and swapping is shown in Algorithm 1.

Thanks to the hardware's parallel capabilities and the synchronous circuits' assignment characteristics, the write operation of

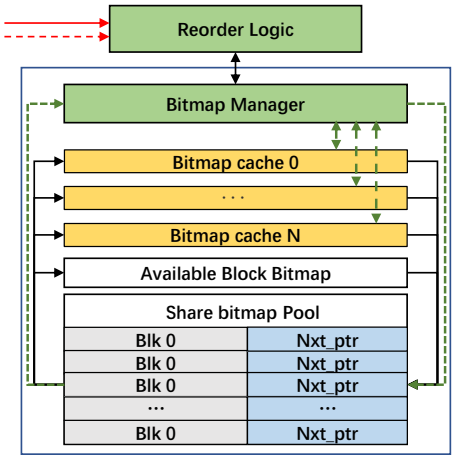

Figure 3: Dual-state bitmap pool

bitmaps from the cache to the bitmap pool and the read operation of bitmaps from the bitmap pool to the cache can be performed synchronously. The time overhead for switching bitmaps is determined by the maximum of the two bitmap switching times.

Algorithm 1: Bitmap manager

```
Get reorder op(type, shift, fbaddr);
if bitmap match cache=1 || empty cache=1 then
    Update bitmap in cache;
end
else
    T_pool = 5*(PSN-RCV_NXT)/blk_size;
    if T_pool <T_max then
        Rd block untill match;
        if need new bitmap then
            Pop block Avail_bitmap;
            Add address on Nxt_ptr;
        end
        Update bitmap in pool;
        if need release bitmap then
            Push block Avail_bitmap;
            Update start block;
        end
    end
    else
        Swap-bitmap with smallest cache;
        Update bitmap in cache;
    end
end
```

3.3 Gather queue

LEFT introduces gather queues to slightly increase the latency of large OoO connections to improve the bitmap cache hit rate. By doing so, it reduces the need for frequent switching between multiple connections in the cache, thereby enhancing the efficiency of packet reordering.

Specifically, the LEFT gathering module includes multiple FIFO queues for packet gathering. Each connection that requires gathering randomly occupies one FIFO queue. LEFT keeps track of the number of connection packets gathered in each FIFO queue and the enqueue time of the first packet from each gathered connection.

Once a certain number of packets (out_cnt) are gathered or the gathering timeout is reached, the packets are dequeued from the FIFO queues and passed to the reorder module for subsequent processing.

When there is a FIFO conflict among the connections that need to be gathered, the FIFO with the highest packet count will dequeue its current packets and allocate FIFO to the connection that does not have a FIFO. The pseudo code for bitmap updating and swapping is shown in Algorithm 2.

Algorithm 2: Packet Gather

```
if Connections match FIFO then
    Write to matched FIFO; Update(packet count);
end
else
    if Empty FIFO available then
        Allocate new FIFO;
    end
    else
        Dequeue highest packet count FIFO;
        Allocate FIFO;
    end
    Update(time,packet count);
end
if Timeout|| packet count >= out_cnt then
    Dequeue and set FIFO empty;
end
```

3.4 Reorder scheduler

LEFT uses a scheduler to perform fast and slow reorder path selection for incoming packets. **Fast path:** For packets with inorder and delay in the bitmap pool T_{pool} less than the upper tolerance limit T_{max}, the fast path will be selected to enter the reordering module directly. **Slow path:** For packets with larger T_{pool} will enter the FIFO for the gathering and will be processed by reordering in batches after the gathering is completed.

This strategy ensures that latency-sensitive small flows are not affected by gathering. Moreover, filtering out a significant amount of connections effectively reduces the frequency of aggregation FIFO conflicts and allows the number of gathering packets closer to the value of out_cnt.

3.5 Other design details

Structural configuration: Previous work has proven that the number of OoO packets received by RNIC will theoretically not exceed BDP_{max} [21, 23]. We consider congestion and switch queue

Algorithm 3: Path selection

```
if new pkt then
    T_pool=(PSN - RCV_NXT)/ blk_size;
    if T_pool < T_max then
        select fast path;
    end
    else
        select slow path;
    end
end
```

length and, through simulation, set the longest path BDP size bitmap pool. This is completely sufficient as LEFT currently considers OoO packets caused by multi-path transmission under lossless networks, and the theoretical upper limit is only 0.5BDP size. At the same time, 3 bitmap caches with a size of 0.5BDP and 8 gathering FIFOs with a gathering number of 8 are set up, which can meet the sorting performance under hard conditions4.3.

Packet lost: Although LEFT operates in a lossless network, to achieve broader application, it still takes into account the possibility of a lossy environment. Therefore, LEFT uses the RTO mechanism and connection bitmap size [7] to trigger selective retransmission.

Bitmap exhaustion: Although a large enough bitmap has been reserved, there is still a minimal probability that the bitmap will be exhausted. In this case, subsequent random packets with bitmap overflow will be discarded and a NACK will be returned. At the same time, new bitmaps will not be allocated to the connections before this disorder recovery so as to avoid occupying an excessively large bitmap while waiting for retransmission, leaving other connections with no bitmaps available, and also returning NACK, and finally, the entire system is paralyzed.

4 EVALUATION

We demonstrate the functionality and characteristics of LEFT through NS3 simulations. Through simulations, we show the advantages of LEFT compared to single-path transmission. It effectively supports multipath transmission, resulting in improved throughput and reduced tail latency. Furthermore, we conduct targeted experiments to demonstrate that LEFT maintains excellent reordering performance across different levels of packet disorder.

4.1 LEFT benefits FCT

Topology: We use 4k-scale Dradonfly topology [17], which contains 80 groups, each group has 26 switches, each switch is connected to two hosts, and the group is directly connected to two links. All link bandwidths are set at 200Gbps, link latency is set to 1us, MTU is set to 1KB, bitpool size is set to 2Kb (8b*128*2), the bitmap cache is set to 3, and the number of aggregate queues is set to 8.

Setup: In this experiment, we employed actual workloads obtained from real data center, sampled with High-Performance Linpack (HPL) [18, 28] benchmark, mode across various matrix sizes (P and Q) under LONG+LONG model. We set up five sets of comparative experiments: per-packet reroute transmission, per-flowlet reroute transmission with LEFT and IRN respectively, and per-flow as the single path transmission.

Results: As shown in Fig.4, 5, in the case of LEFT enhancement, per-packet reroute transmission demonstrates a significant improvement over per-flow transmission in terms of average FCT (28.1% reduction) and 99% tail latency (48.2% reduction). Furthermore, the PSN is no longer used as a trigger for retransmissions, in the per-packet transmission mode, resulting in a 20% improvement in average FCT and a 26.7% reduction in 99% tail latency compared to the IRN approach. In the per-flowlet transmission mode, there is a 12% improvement in average FCT and an 8.1% reduction in 99% tail latency.

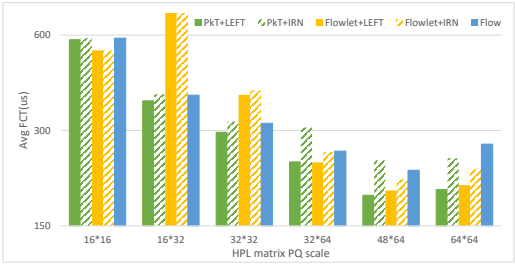

Figure 4: LEFT reduced average FCT

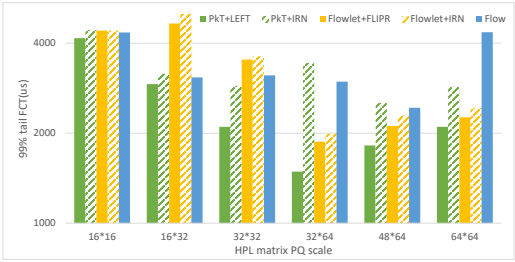

Figure 5: LEFT reduced 99% tail FCT

4.2 LEFT improved throughtput

Setup: We selected two servers under different switches in the same group and introduced a link with varying latency between the switches. Switches will evenly distribute the packets across these two links. One server set 5 connections sent traffic to the other server at full speed, and we measured the connection's throughput. We compared four scenarios: LEFT, MELO+ (MELO with PSN trigger retransmissions disabled, using only the bitmap pool), LEFT- (non-gather version of LEFT), and an ideal reorder. All bitmap delays are derived from the FPGA (clock frequency 200MHz) test. The ideal is set to 10ns, and the bitmap cache is set to 10ns. The first access of each block in the bitmap pool requires 15ns, and each subsequent block requires 5ns.

Results: As shown in Fig.6, it is evident that as the Round-Trip Time (RTT) difference between the two links increases, the shared bitmap delay of MELO+ also increases exponentially. Consequently, the RNIC processing experience a sharp drop and the throughput

rate decreases to only 33%. The issue with LEFT- is attributed to the absence of gathering, leading to frequent cache misses and causing LEFT- to maintain only 50.5% of the effective bandwidth due to frequent bitmap switching. On the other hand, complete LEFT benefits from gathering, reducing cache misses, and allowing access to bitmaps with low latency from the fast path for reordering. Even when the path RTT differs by 32 times, the throughput can be maintained at 94%, which is 83% and 180% higher than LEFT- and MELO+, respectively.

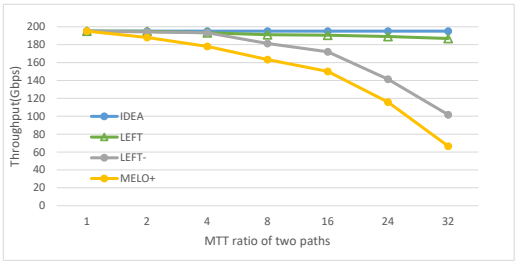

Figure 6: LEFT improved throughtput

4.3 Deep Dive

Baseline setting each connection requires 5BDP (512bit*5) [27] size bitmap, and LEFT setting consistent with section 3.5. As shown in Fig.7, when the number of concurrent connections is small, LEFT's overhead is greater than MELO+ and LEFT- due to the addition of bitmap cache and gather queue, but it is still far less than the overhead of baseline. As the number of connections increases, the advantages of LEFT become obvious. When the number of RNIC connections reaches 5000, LEFT memory overhead is 27KB, an increase of 9.1% compared to MELO+, an increase of 5.5% compared to LEFT-, and only 1.7% of baseline.

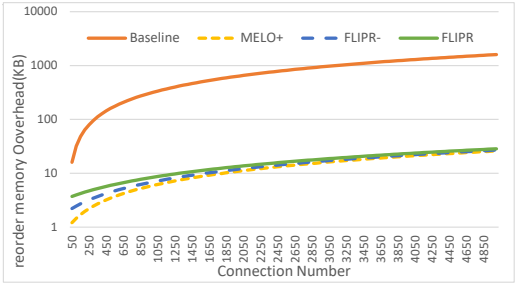

Figure 7: Onchip memory overhead

5 CONCLUSION

We present LEFT, a high-performance and scalable design for RDMA packet reordering. By leveraging an enhanced shared bitmap pool and fast and slow reordering path settings, LEFT achieves exceptional speed, enabling line-speed out-of-order packet reordering under per-packet reroute transmission. Additionally, a shared bitmap pool is utilized to ensure scalability, allowing it to operate efficiently even under high concurrency situations while consuming minimal memory resources. Simulation results demonstrate that LEFT maintains a remarkable throughput rate of 94% even when the path RTT difference is 32 times, surpassing the performance of the original shared bitmap pool by 180%.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 62222204 and Grant 62172148, in part by the National Key Research and Development Program of China under Grant 2023YFB3002203, and in part by the Major special project of Changsha Science and Technology Plan under Grant kh2103016 and Grant kh2401005.

REFERENCES

[1] 2017. RDMA in Data Centers: Looking Back and Looking Forward. https://conferences:sigcomm:org/events/apnet2017/slides/cx:pdf,2017.

[2] 2020. InfiniBand Trade Association. In *2020. InfiniBand Architecture Specification Release 1.4 Annex A17: RoCEv2.*

[3] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA*. USENIX Association, 281–296. http://www.usenix.org/events/nsdi10/tech/full_papers/al-fares.pdf

[4] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, Fabián E. Bustamante, Y. Charlie Hu, Arvind Krishnamurthy, and Sylvia Ratnasamy (Eds.). ACM, 503–514. https://doi.org/10.1145/2619239.2626316

[5] Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. 2020. Enabling Programmable Transport Protocols in High-Speed NICs. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, Ranjita Bhagwan and George Porter (Eds.). USENIX Association, 93–109. https://www.usenix.org/conference/nsdi20/presentation/arashloo

[6] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. 2023. Empowering Azure Storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 49–67. https://www.usenix.org/conference/nsdi23/presentation/bai

[7] Guo Chen, Yuanwei Lu, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, and Thomas Moscibroda. 2019. MP-RDMA: Enabling RDMA With Multi-Path Transport in Datacenters. *IEEE/ACM Trans. Netw.* 27, 6 (2019), 2308–2323. https://doi.org/10.1109/TNET.2019.2948917

[8] cx3. 2018. Mellanox ConnectX-3 Product Brief. (2018). arXiv:cs.CL/2100.08172 https://lenovopress.lenovo.com/tips0897-mellanox-connectx-3.

[9] cx5. 2020. Mellanox ConnectX-5 Product Brief. (2020). arXiv:cs.CL/2300.08774 https://network.nvidia.com/files/doc-2020/pb-connectx-5-en-card.pdf.

[10] Advait Abhay Dixit, Pawan Prakash, Y. Charlie Hu, and Ramana Rao Kompella. 2013. On the impact of packet spraying in data center networks. In *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*. IEEE, 2130–2138. https://doi.org/10.1109/INFCOM.2013.6567015

[11] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. 2021. When Cloud Storage Meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, James Mickens and

Renata Teixeira (Eds.). USENIX Association, 519–533. https://www.usenix.org/conference/nsdi21/presentation/gao

[12] Soudeh Ghorbani, Zibin Yang, Philip Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. DRILL: Micro Load Balancing for Low-latency Data Center Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*. ACM, 225–238. https://doi.org/10.1145/3098822.3098839

[13] 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 ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, Marinho P. Barcellos, Jon Crowcroft, Amin Vahdat, and Sachin Katti (Eds.). ACM, 202–215. https://doi.org/10.1145/2934872.2934908

[14] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John B. Carter, and Aditya Akella. 2015. Presto: Edge-based Load Balancing for Fast Datacenter Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye (Eds.). ACM, 465–478. https://doi.org/10.1145/2785956.2787507

[15] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2019. Datacenter RPCs can be General and Fast. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, Jay R. Lorch and Minlan Yu (Eds.). USENIX Association, 1–16. https://www.usenix.org/conference/nsdi19/presentation/kalia

[16] Naga Praveen Katta, Mukesh Hira, Aditi Ghag, Changhoon Kim, Isaac Keslassy, and Jennifer Rexford. 2016. CLOVE: How I learned to stop worrying about the core and love the edge. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks, HotNets 2016, Atlanta, GA, USA, November 9-10, 2016*, Bryan Ford, Alex C. Snoeren, and Ellen W. Zegura (Eds.). ACM, 155–161. https://doi.org/10.1145/3005745.3005751

[17] John Kim, William J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *35th International Symposium on Computer Architecture (ISCA 2008), June 21-25, 2008, Beijing, China*. IEEE Computer Society, 77–88. https://doi.org/10.1109/ISCA.2008.19

[18] Shuhei Kudo, Keigo Nitadori, Takuya Ina, and Toshiyuki Imamura. 2020. Implementation and Numerical Techniques for One EFlop/s HPL-AI Benchmark on Fugaku. In *11th IEEE/ACM Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA@SC 2020, Atlanta, GA, USA, November 13, 2020*. IEEE, 69–76. https://doi.org/10.1109/SCALA51936.2020.00014

[19] Petr Lapukhov, Ariff Premji, and Jon Mitchell. 2016. Use of BGP for Routing in Large-Scale Data Centers. *RFC* 7938 (2016), 1–35. https://doi.org/10.17487/RFC7938

[20] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, Jianping Wu and Wendy Hall (Eds.). ACM, 44–58. https://doi.org/10.1145/3341302.3342085

[21] Yuanwei Lu, Guo Chen, Zhenyuan Ruan, Wencong Xiao, Bojie Li, Jiansong Zhang, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2017. Memory Efficient Loss Recovery for Hardware-based Transport in Datacenter. In *Proceedings of the First Asia-Pacific Workshop on Networking, APNet 2017, Hong Kong, China, August 3-4, 2017*, Kai Chen and Jitendra Padhye (Eds.). ACM, 22–28. https://doi.org/10.1145/3106989.3106993

[22] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily R. Blem, Hassan M. G. Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye (Eds.). ACM, 537–550. https://doi.org/10.1145/2785956.2787510

[23] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting network support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*, Sergey Gorinsky and János Tapolcai (Eds.). ACM, 313–326. https://doi.org/10.1145/3230543.3230557

[24] 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, ACM SIGCOMM 2023, New York, NY, USA, 10-14 September 2023*, Henning Schulzrinne, Vishal Misra, Eddie Kohler, and David A. Maltz (Eds.). ACM, 816–831. https://doi.org/10.1145/3603269.3604849

[25] 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, ACM SIGCOMM 2023, New York, NY, USA, 10-14 September 2023*, Henning Schulzrinne, Vishal Misra, Eddie Kohler, and David A. Maltz (Eds.). ACM, 816–831. https://doi.org/10.1145/3603269.3604849

[26] 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 2017, Boston, MA, USA, March 27-29, 2017*, Aditya Akella and Jon Howell (Eds.). USENIX Association, 407–420. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini

[27] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, Tianhao Wang, Weicheng Ling, Kejia Huo, Pingbo An, Kui Ji, Shideng Zhang, Bin Xu, Ruiqing Feng, Tao Ding, Kai Chen, and Chuanxiong Guo. 2023. SRNIC: A Scalable Architecture for RDMA NICs. In *23th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, April 20-23, 2023*, J (Ed.). USENIX Association, 519–533. https://www.usenix.org/conference/nsdi21/presentation/gao

[28] Gen Xu, Huda Ibeid, Xin Jiang, Vjekoslav Svilan, and Zhaojuan Bian. 2020. Simulation-Based Performance Prediction of HPC Applications: A Case Study of HPL. In *IEEE/ACM International Workshop on HPC User Support Tools and Workshop on Programming and Performance Visualization Tools, HUST/ProTools@SC 2020, Atlanta, GA, USA, November 18, 2020*. IEEE, 81–88. https://doi.org/10.1109/HUSTPROTOOLS51951.2020.00016