NetRT: Enhancing RDMA with Retransmission Offloading in Data Center Networks

Wentao Wang*, Jiangping Han†‡, Kaiping Xue†‡, Jian Li†, Kunpeng Ding†, Ruidong Li§
*Department of EEIS, University of Science and Technology of China, Hefei, Anhui 230027, China
†School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China
§National Institute of Information and Communications Technology, Kanazawa University, Tokyo 184-0015, Japan
‡Corresponding author: J. Han, K. Xue (jphan, kpxue)@ustc.edu.cn

IEEE INFOCOM 2025 - IEEE Conference on Computer Communications | 979-8-3315-4305-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/INFOCOM55648.2025.11044493

***Abstract*—RDMA over Converged Ethernet (RoCE) enables the deployment of RDMA in Ethernet-based data centers and is becoming a mainstream solution. Due to the limited processing capability of RDMA NICs (RNICs), Priority-based Flow Control (PFC) is enabled to prevent decreased transmission efficiency caused by packet loss and retransmission. However, PFC brings performance impairments and cannot fully ensure reliability in the presence of various packet loss and out-of-order delivery cases in large-scale data centers. To achieve high resilience against lossy conditions and disorders while considering feasibility, this paper proposes NetRT, an innovative retransmission offloading solution. NetRT deploys selective retransmission between top-of-rack switches instead of RNICs, leveraging modern switches' hardware resources and programmable features for efficient and deployable in-network recovery. We develop key mechanisms to manage in-network retransmissions, including a ternary state machine and congestion avoidance, aiming for efficiency and transparency. Evaluations show that NetRT can handle diverse packet loss and out-of-order cases with acceptable overhead, reducing end-to-end retransmissions and decreasing flow completion times by up to 75%.**

***Index Terms*—Data Center Networks, RDMA, Programmable Switches, Selective Retransmission**

I. Introduction

Applications within data centers require high bandwidth and low latency [1]–[3]. Remote Direct Memory Access (RDMA) offers higher transfer performance and lower CPU overhead via bypassing the kernel and offloading the stack to Network Interface Cards (NICs), thus becoming the main transmission technology in high-performance computing clusters. With the ability to implement RDMA over existing Ethernet facilities instead of specific switching equipment, RDMA over Converged Ethernet (RoCE) becomes the canonical deployment method that balances performance and cost.

Despite the benefits of hardware offloading, RDMA suffers from limited hardware resources of RDMA NICs (RNICs). Specifically, RNICs are equipped with relatively limited on-chip memory, which presents a significant challenge in handling out-of-order packets and maintaining the data structures necessary for reordering. To illustrate, commonly used commodity RNICs like NVIDIA Mellanox ConnectX-5 (CX5) [4] have only 2 MB of memory and deploy a simple Go-Back-N (GBN) retransmission mechanism. As a result, even a small quantity of packet loss can cause significant performance degradation, e.g., 0.1% loss rate can lead to 25% drop in throughput [5]. To prevent such performance degradation caused by congestion loss, RoCE employs Priority-based Flow Control (PFC) [6] to construct lossless networks, thus avoiding queue overflow via pausing upstream ports.

However, PFC is problematic and inadequate for RoCE. For one thing, there are inherent performance impairments associated with PFC. Several studies have exposed that PFC can cause congestion spreading, unfairness, head-of-the-line blocking, and even deadlocks [7]–[9]. For another, PFC is not a comprehensive solution for all packet loss and out-of-order scenarios in data center networks. Link corruption is another significant source of packet loss in large-scale data centers, which provokes a higher loss rate and cannot be mitigated by flow control [10]. Besides, some fine-grained load balancing schemes may distribute packets across different paths, potentially causing packet disorder and prompting unnecessary retransmissions by RNICs [11]–[13]. Therefore, rather than relying solely on PFC, enhancing the transmission resilience for RoCE to cope with complex lossy conditions represents a crucial solution [14], as well as a significant challenge.

To address this challenge, the existing state-of-the-art solutions focus on improving architecture and deploying more efficient Selective Retransmission (SR) on memory-limited RNICs to eliminate the dependence on PFC. For instance, IRN [15] avoids high memory overheads by meticulously designing SR-specific data structures, and SRNIC [16] employs buffer uploading. Despite the improved resilience of these solutions, the modifications to RNIC hardware introduce deployment complexity. Unlike software updates, the existing commodity RNICs are challenging to modify functionally, making it difficult to implement these advanced solutions in actual data centers. Additionally, the costs and business interruptions associated with equipment replacement further limit feasibility. This raises the question: *is there a feasible improvement without reliance on RNIC hardware modifications?*

The advancements in modern programmable switches provide an answer to this question. Over the past decade, vendors have consistently increased the capacity of their commodity switches. The current generation of switches has reached a capacity of tens of megabytes, as exemplified by Intel Tofino switch [17], which has a 64 MB shared buffer. In contrast, data center networks with bandwidths of hundreds of Gbps and latencies of microseconds have a bandwidth-

delay product limited to O(100 KB) [18]. Thus, the switches have sufficient space to accommodate additional algorithms' data structures. Furthermore, these switches offer enhanced operational flexibility. Network operators can customize packet handling through data plane languages such as P4 [19]. These benefits allow switches to assume more transport functions and provide more in-network services.

In light of these observations, we propose NetRT, an in-**Net**work **R**e**T**ransmission offloading solution, deployed on programmable Top-of-Rack (ToR) switches. Our core idea is to leverage relatively abundant hardware resources (as opposed to RNICs) and programmable features to perform efficient ToR-to-ToR SR for enhanced transmission resilience and feasibility considerations. The design space of NetRT is shown in Fig. 1. Specifically, NetRT constructs a duplicate pool to mirror traffic at the source ToR (SrcToR) and checks the packet arrival order at the destination ToR (DstToR). In the event of packet loss, the DstToR tracks the arrival of out-of-order packets and performs a SACK-like retransmission request, based on which the SrcToR implements in-network recovery from the duplicate pool. In addition, to address the deployment challenges, we design mechanisms including duplicate pool admission policy, retransmission state machine, and in-network congestion avoidance to enable NetRT to efficiently utilize switch resources and achieve transparent in-order packet delivery.

We evaluate NetRT comprehensively through both testbed experiments and ns-3 [20] simulations, which demonstrate that NetRT enables significant performance improvements. Our testbed results show that NetRT can effectively cope with network packet loss, providing up to 75% reduction in Flow Completion Time (FCT) with acceptable overhead. In incast scenarios, we show that NetRT can work effectively in lossy networks, offering at most about 50% improvement. Additionally, NetRT enhances out-of-order tolerance and can be employed in conjunction with fine-grained load-balancing schemes, further reducing tail FCT by 30%.

In summary, the contributions of this paper are as follows:

- We propose NetRT to provide enhanced transmission resilience for RoCE using programmable switches. NetRT innovatively offloads retransmission to the near end, enabling efficient ToR-to-ToR SR without requiring any modifications to RNIC hardware.
- We design key components of NetRT with deployment considerations, including duplicate pool utilization, ternary retransmission state machine, and in-network congestion avoidance. These designs ensure efficient and transparent in-order packet delivery with acceptable overhead and in-network impact.
- We analyze the feasibility of NetRT and conduct comprehensive evaluations through real testbed experiments and simulations. The results show that NetRT can effectively enhance RoCE's handling of lossy conditions and disorders, providing benefits comparable to RNIC-based solutions while maintaining high practicality.

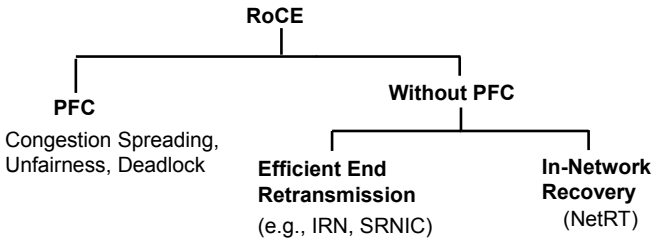


Fig. 1: Design space for transmission reliability of RoCE.

The rest of this paper is organized as follows. In Section II, we explain the motivation of our solution. Section III provides a detailed description of the design and implementation of NetRT. Comprehensive performance evaluations are presented in Section IV. In Section V, we introduce the related works aimed at improving the transmission robustness of RoCE. Finally, we conclude the paper in Section VI.

II. Motivation

In this section, we illustrate the necessity of enhancing loss and out-of-order tolerance for RoCE while also highlighting the challenges associated with the deployment of existing solutions. Subsequently, we present the insights gained from the development of programmable switches.

A. Flaws and Inadequacies of PFC

Since RNICs perform poorly when there is packet loss, PFC is enabled for RoCE to avoid congestion packet loss. However, PFC introduces configuration complexity and other performance issues. For instance, the backpressure mechanism employed by PFC can lead to the accumulation of upstream queues, causing congestion to spread. Coarse-grained port pauses can harm uncongested flows and introduce unfairness. In the presence of buffer loop dependencies, it can even result in network deadlocks. These issues have been extensively documented in numerous works [7]–[9], [21].

Furthermore, PFC is not adequately suited for handling all packet loss and out-of-order situations, such as packet corruption and out-of-order caused by load balancing across multiple paths. Packet corruption represents another significant contributor to loss in large-scale data centers. It occurs due to fiber bending and damage, connector contamination, etc., eventually resulting in data packet error and discarding. Zhuo *et al.* demonstrated that packet corruption can impose a higher packet loss rate than congestion [10]. Packet corruption has become a pervasive challenge in data centers [22], which cannot be mitigated by flow control mechanisms. Besides, the load balancing process may also result in transmitting packets in an incorrect sequence. In large-scale data centers, multiple links are typically established between nodes. Some fine-grained schemes may distribute the packets of the same flow across different paths to achieve a more balanced effect [11], [12]. These strategies can potentially cause packets to arrive out of order, leading RNICs to incorrectly interpret packet loss and perform unnecessary retransmissions. Thus,

better load balancing and transmission efficiency are dilemmas with the low out-of-order tolerance of RNICs.

In summary, relying solely on PFC is insufficient to address the transmission challenges in data centers. We believe that enhancing transmission resilience is a more pivotal strategy.

B. Deployment Complexity of RNIC-based Solutions

Due to the weak transmission robustness caused by the inefficient GBN, some solutions propose to replace it with more efficient Selective Retransmission (SR). To address the memory overhead associated with SR, these solutions optimize the algorithms and redesign the architecture [15], [16], [23]. While these solutions improve the transmission resilience of RNICs on their respective platforms, they face significant challenges in actual deployment. The primary obstacle is that the hardware offloading nature of RDMA forces these solutions to introduce RNIC hardware modifications for deployment, which is not feasible in most data centers. Existing data centers are equipped with commodity RNICs that typically do not offer primitives for customized modifications.

In recent years, some commodity RNICs have also enabled SR. NVIDIA's ConnectX series, for instance, has commenced support for SR in CX6 [24]. The advent of these commodities greatly reduces the complexity of deployment. However, for facilities that have already been established, upgrading to these new RNICs would require substantial replacement of network devices. It is uncommon to replace equipment in bulk in active data centers due to significant financial and operational considerations involved. Therefore, the feasibility and cost of deploying a solution are crucial factors to consider.

C. Development of Programmable Switches

Programmable switches have evolved in recent years, specifically in two aspects: the expansion of buffering space and the enhancement of capabilities, thus being able to take on more transmission functions within networks

In contrast to RNICs, which typically have a limited on-chip memory of only a few megabytes, programmable switches possess a significantly higher buffer space. The switch buffer size has grown by an order of magnitude, reaching levels in the tens of megabytes. For instance, Trident 3 has a shared buffer of 32 MB, and Intel Tofino 2 provides 64 MB. Table I summarizes the shared buffer sizes of popular commodity programmable switches [17], [25]–[27]. In contrast, despite increasing bandwidth in data centers, the Round-Trip Time (RTT) is reduced to microseconds, limiting the Bandwidth-Delay Product (BDP) to the range of O(10 KB) to O(100 KB). It can be concluded that the current switches have sufficient capacity to accommodate network traffic [28] and to maintain additional data structure overhead for in-network algorithms.

Along with the sufficient memory space, the switches' capabilities are also richer. Modern programmable switches support multiple queues per port and provide scheduling functions within the data plane. Users can independently pause and resume each queue using the provided primitives, enabling per-flow transmission management such as traffic backup [22], packet holding and in-network reordering [13]. Furthermore, the switches enable users to create customized metadata for additional in-network interaction. These features make in-network retransmissions possible.

TABLE I: Shared buffer size of mainstream programmable switches.

Switch Commodity	Buffer Size
Trident 3	32 MB
Spectrum 3	64 MB
Tofino 2	64 MB
Tomahawk 4	$>$ 64 MB

Summary and our design goal: We show the observations on RoCE transmission reliability: (i) The hop-by-hop flow control mechanism (typically PFC) has inherent defects and fails to fully guarantee reliability. (ii) The RNIC-based SR performs better but introduces hardware deployment complexity. Then, we show that the network can put SR to the near-end switch, leveraging the growing capabilities of programmable switches to provide retransmission offloading for end RNICs. Accordingly, our design objective is to provide transparent in-network packet loss recovery and in-order packet delivery with feasible overhead, addressing the challenges of transmission reliability requirements and deployability.

III. NetRT Design

We propose NetRT, an in-network retransmission offloading solution designed to enhance RoCE resilience through near-end SR. In this section, we start by describing the design principles of NetRT. Subsequently, we present a general overview, followed by a detailed description of each component. Finally, we analyze the overhead and feasibility of NetRT.

A. Design Principles

NetRT is designed to implement in-network loss recovery and reordering on programmable ToR switches to provide packet order guarantees without modifying RNIC hardware and transport protocol. The following three design principles guide the development of NetRT.

1) **Effectiveness.** Due to the negative impact of end retransmissions on transmission efficiency, NetRT must possess the capacity to handle various loss and disorder cases. When packet loss occurs, NetRT needs to accurately identify the lost packet and promptly execute in-network recovery and reordering operations.
2) **Transparency.** To ensure the deployability of the solution, NetRT is transparent to RNICs. NetRT involves no modifications to the transport protocol and no additional interaction with RNICs. Furthermore, the solution should avoid impacting other traffic.
3) **Best Efforts.** Given that the primary function of a switch is forwarding, NetRT employs the principle of best-effort services. Once faced with resource constraints, switches prioritize original packet buffering and forwarding, and

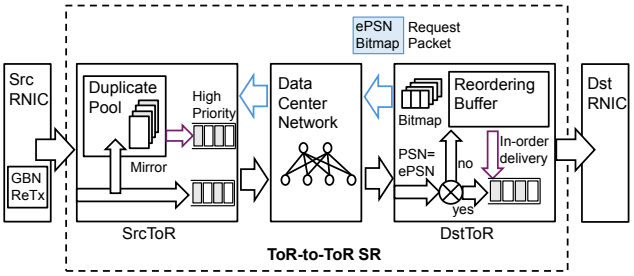


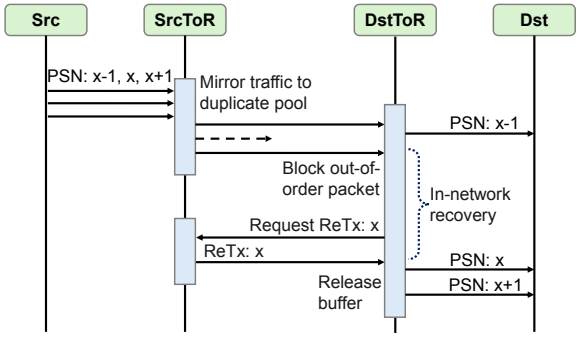
Fig. 2: Overview of NetRT.

NetRT is required to have the ability to trigger retained end retransmission.

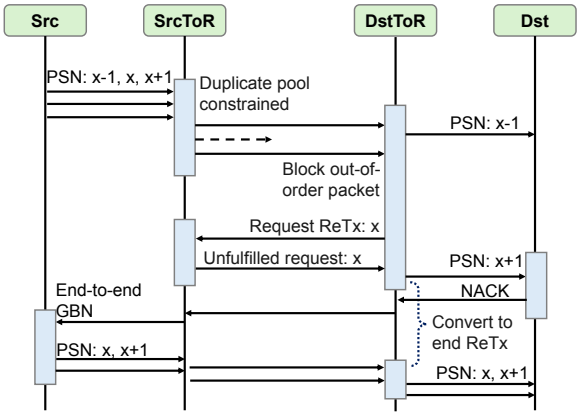
B. Overview

Based on the above principles, we design NetRT, which is overviewed in Fig. 2. To balance transmission robustness with solution feasibility, NetRT implements ToR-to-ToR SR at the near end instead of end-RNICs. Therefore, NetRT comprises two components deployed on the source ToR (SrcToR) and the destination ToR (DstToR), interconnected through the data center network. At SrcToR, NetRT performs two key functions: (i) utilizing available buffer space to create a duplicate pool serving as an in-network backup for traffic, and (ii) retrieving the duplicate pool upon receiving a retransmission request from the opposite end and performing in-network retransmission. At DstToR, NetRT handles: (i) checking the packets' arrival order and ensuring that packets are delivered in order, (ii) blocking and tracking out-of-order packets during packet loss, executing retransmission requests, and (iii) managing and orderly releasing the reordering buffer based on transmission state.

The key interaction design of NetRT between the ToRs is illustrated in Fig. 3. For packets arriving in sequence, NetRT conducts regular forwarding and enqueuing operations. In the event of a packet loss, the DstToR intercepts out-of-order packets in the reordering buffer and requests retransmission using the maintained expected Packet Sequence Number (ePSN) and a bitmap data structure, similar to TCP's SACK. Upon receiving the retransmission request, the SrcToR can determine the request sequence through the ePSN and the hole in the bitmap. If the duplicate pool contains the requested data, SrcToR assigns a high priority and selects a rate for the packets to perform in-network retransmission. DstToR then receives and delivers the retransmitted packets, sequentially releasing the reordering buffer, thus achieving transparent packet loss recovery for RNICs as shown in Fig. 3(a). In cases where resources are constrained, the SrcToR may not be able to retain duplicates of the requested data. Afterward, it responds with an unfulfilled request notification, instructing DstToR to cease in-network reordering. DstToR then switches to end retransmission mode, actively releasing out-of-order packets to trigger end-to-end retransmission, and waits for the end RNIC to complete the recovery process, as shown in Fig. 3(b).



(a) In-network loss recovery and reordering.



(b) Converting to end retransmission mode in resource-constrained situations.

Fig. 3: NetRT's main interaction process design.

C. Deployment Considerations

The deployment challenges of NetRT include efficient utilization of the limited remaining SrcToR buffers, tracking and managing per-flow retransmission with low overhead, and mitigating the impact of retransmitted traffic on networks. To address these challenges, we design a duplicate pool admission algorithm, a ternary state machine at DstToR, and an in-network congestion avoidance mechanism.

1) Duplicate Pool: For each packet, its duplicate remains in the duplicate pool for about one RTT to ensure that the original packet is delivered to the receiver. Thus, the optimal size for the duplicate pool associated with each port is one BDP. In resource-limited scenarios, the switch prioritizes the buffering requirements of regular flows, thereby reducing the space available in the duplicate pool, which may prevent SrcToR from maintaining a complete duplicate of all flows.

To optimize the utilization of the duplicate pool, we employ the duplicate replacement strategy. Our core idea is that flows requested multiple times are likely to experience greater congestion and are more prone to subsequent retransmission requests. Consequently, maintaining more copies of these flows is more effective for meeting retransmission needs. Therefore, we adopt the Least Frequently Used (LFU) replacement strategy. NetRT tracks the number of retransmission requests per flow at SrcToR. Upon receipt of a packet at the ToR switch, it

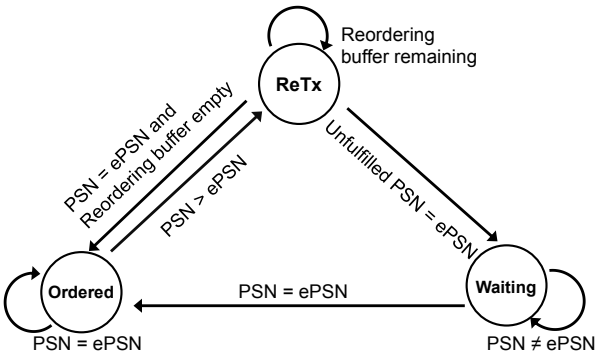


Fig. 4: State machine for retransmission, which transfers at every packet received.

is copied and added directly to the pool if the duplicate pool is not at capacity. If the duplicate pool is full, NetRT removes the oldest duplicate of the flow with the fewest requests and replaces it with the newly arriving packet.

2) Ternary State Machine for Retransmission: NetRT verifies the order of incoming packets on DstToR and selectively requests retransmission in case of packet loss. We primarily use two data structures: ePSN, representing the expected PSN for a flow, and bitmap which tracks packet arrivals following the ePSN. The bitmap is set to 128 bits, which allows it to accommodate the BDP of networks. Each bit in the bitmap is set to 1 to indicate an arrival.

DstToR must accurately determine whether to forward or block a packet. We thus design a ternary state machine, shown in Fig. 4, to guide the actions of the switch. The transmission states include the `Ordered` state, which indicates that traffic arrives in the correct order; the `ReTx` state, which indicates that in-network recovery is in progress; and the `Waiting` state, which indicates that NetRT is unable to achieve in-network recovery, thus switching to end-retransmission mode.

In the `Ordered` state, an incoming packet is directly forwarded if its PSN matches the ePSN. If the PSN is greater than the ePSN, a disorder is detected, and the state transitions to `ReTx`. Subsequently, the DstToR buffers the out-of-order packet, updates the bitmap, and sends a retransmission request containing the ePSN and the bitmap to the DstToR. In the `ReTx` state, NetRT continuously blocks incoming out-of-order packets until the ePSN arrives. It then attempts to release the reordering buffer in sequence, as illustrated in Algorithm 1.

Algorithm 1: Release Reordering Buffer

```
while bitmap & 0X01 do
    forward(buffer.pop());
    ePSN ← ePSN + 1;
    bitmap = bitmap >> 1;
if bitmap = 0 then
    state ← Ordered;  ▷ Reordering buffer is empty.
```

Once the reordering buffer is emptied (i.e., the bitmap value of 0), the in-network recovery process is complete, and the state transitions back to `Ordered`. If SrcToR lacks a duplicate of the requested packet, the DstToR receives an unfulfilled request message and transitions to the `Waiting` state. NetRT in the `Waiting` state first forwards the out-of-order packet with the smallest PSN in the reordering buffer to trigger end-to-end GBN and then flushes the reordering buffer. In this state, NetRT does not block out-of-order packets and waits until the sender re-sends the expected packet, then transitions the state to `Ordered`. The entire deployment logic of the DstToR algorithm is detailed in Algorithm 2.

Algorithm 2: The Ternary State Machine Algorithm

```
Function HandleDataPacket(pkt):
    if pkt.PSN > ePSN and state ≠ Waiting then
        state ← ReTx;                          ▷ Out-of-order
        bitmap ← bitmap &
        1 << (PSN − ePSN − 1);
        Buffer pkt and request retransmission at time
         intervals;
    else if pkt.PSN = ePSN then
        ePSN ← ePSN + 1;
        forward(pkt);
        if state = ReTx then
            Release reordering buffer;
        else if state = Waiting then
            state ← Ordered;
Function HandleNetRTMessage(pkt):
    if pkt.unfulfilledPSN = ePSN then
        state ← Waiting;
        forward(buffer.pop());   ▷ Triggering end ReTx.
        buffer.clear();
        bitmap ← 0;
```

3) Retransmission Event and In-network Congestion Avoidance: When an incast occurs, it often results in a large number of consecutive packet losses. If these lost packets are retransmitted continuously, it can trigger further congestion, leading to significant performance degradation. To address this, we design in-network congestion avoidance to select the retransmission rate R_r. The congestion avoidance logic of NetRT can be shown in Fig. 5. We define two thresholds: K_l and K_h. When the number of retransmitted packets N is less than K_l, indicating that the current congestion level is insignificant, NetRT sends the requested packet at the line rate R_l. When the retransmission data exceeds K_l, it indicates large congestion at the bottleneck and continuous packet loss. In this case, NetRT performs in-network congestion avoidance using the following formula:

$$R_r = \max\{R_a^{max} - \frac{(N - K_l)(R_a^{max} - R_a^{min})}{K_h - K_l}, R_a^{min}\},$$

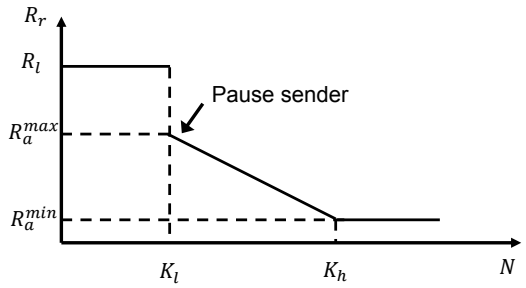


Fig. 5: Logic of NetRT's retransmission rate adjustment.

where R_a^{max} and R_a^{min} are the upper and lower bounds for retransmission rate adjustment. NetRT pauses the corresponding sender and sends retransmission packets at a low rate that decreases linearly with the amount of requested data. We use the original PFC pause frame, which is utilized only between senders and ToRs, thus avoiding the defects of PFC.

D. Overhead and Feasibility Analysis

In this subsection, we analyze the memory overhead of NetRT, illustrate its implementation details, and discuss its feasibility.

1) Overhead Analysis: For SrcToR, the primary overhead is attributed to the duplicate pool. From a pipeline perspective, the data transmitted by a port within an RTT is one BDP, regardless of the number of Queue Pairs (QPs). Therefore, a port with 100 Gbps bandwidth and 5 μs RTT requires less than 100 KB for duplicates. Another memory overhead arises from recording request frequencies for the LFU strategy. We use a 32-bit data structure, resulting in an overhead of 40 KB for 10K QPs. At DstToR, the main overhead is the reordering buffer. Out-of-order packets are forwarded or dropped after at most one RTT, so similar to the duplicate pool, the reordering buffer typically requires no more than 100 KB. Additionally, because NetRT interactions between ToRs involve shorter paths and retransmitted packets travel with high priority, DstToR consumes less memory. For SR-specific data structures maintained by NetRT, the ePSN is identical to the PSN field of the RDMA transport header, using 24 bits, with a bitmap of 128 bits and a retransmission state identifiable by 2 bits. With 10K QPs, the overhead is less than 200 KB. Table II summarizes the overhead of NetRT. In conclusion, the primary overhead of NetRT involves traffic replication and storage. Existing programmable switches typically have sufficient space to accommodate the data overhead of a BDP. Therefore, the overhead of NetRT is tolerable.

TABLE II: Memory overhead of data structures required by NetRT under 100 KB BDP and 10K QPs.

Component	Data Structures	Memory Overhead
SrcToR	Duplicate Pool	100 KB
	Request Frequency	40 KB
DstToR	Reordering Buffer	100 KB
	ePSN	30 KB
	bitmap	160 KB
	state	2.5 KB

2) Implementation Details and Feasibility: We implement NetRT in 800 lines of code on our testbed based on DPDK. For NetRT interaction messages, we insert customized fields. The retransmission request message adds 20 bytes, containing one byte for the message type, 24 bits for the ePSN, and 128 bits for the bitmap. The unfulfilled request message adds only 4 bytes for the message type and unfulfilled request number. The messages traverse the network at the address of the flow served and are dropped at the opposite ToR, thus not affecting the end RNICs.

The SrcToR and DstToR algorithms are deployed in the switch Memory Management Unit (MMU). Modern data center switches typically have multiple queues; we reserve one queue for the duplicate pool. When a packet passes through the ingress queue, the switch creates a replica using the multicast function and adds it to the duplicate pool queue. For duplicate retention, existing programmable switches provide advanced flow control primitives [17], [22]. The main feature of the DstToR algorithm is reordering. The reordering buffer uses a separate queue similar to the duplicate pool and is blocked based on flow control. The position of out-of-order packets corresponds to the bitmap, thus realizing low-overhead sequential release of the buffer as shown in Algorithm 1.

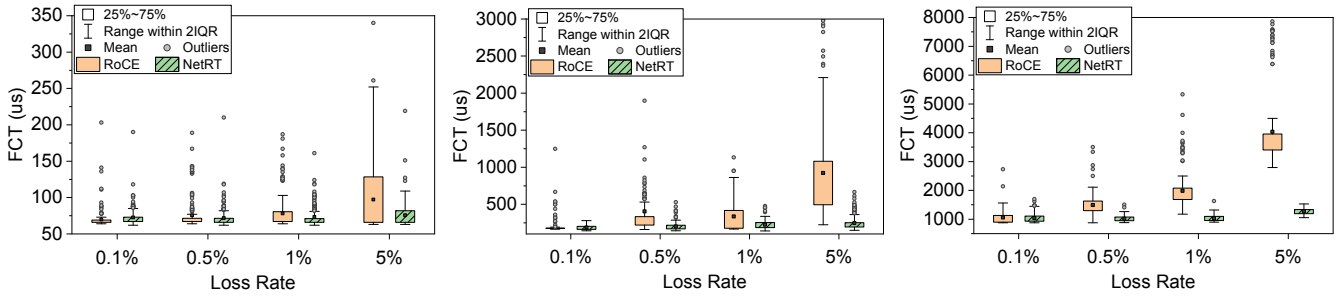
IV. Performance Evaluation

In this section, we evaluate the performance of NetRT through our testbed and ns-3 [20] simulations. We focus on validating that NetRT can effectively achieve in-network packet loss recovery with acceptable overhead, provide high out-of-order tolerance, and ultimately reduce end retransmissions and shorten Flow Completion Time (FCT).

We compare NetRT with GBN, GBN combined with PFC, and IRN [15], which are the default retransmission mechanism for RNICs, the mainstream RoCE solution, and the advanced SR-based solution, respectively. For NetRT, the high and low thresholds associated with congestion avoidance are set to 20 KB and 100 KB, respectively, based on our experience. The rate at which retransmission is limited varies from 1% to 50% of the bandwidth. The parameters for other solutions are set following their respective papers.

A. Testbed Evaluation

1) Testbed Setup: Our testbed consists of five nodes arranged in a chain topology. We designate the nodes at the two ends as a sender and a receiver connected to the core switch in the middle via the ToR nodes. Each node is equipped with a Core i5-11500 processor, 64 GB of DDR4 RAMs, and Intel E810 10GbE NICs [29]. We deploy the stack and implement NetRT based on DPDK [30]. At the end nodes, the GBN retransmission mechanism is employed.



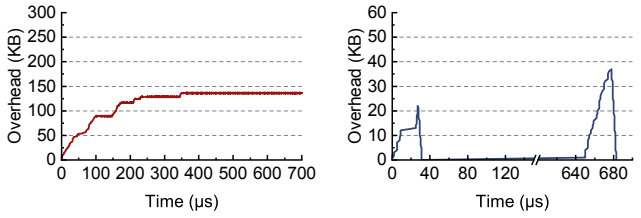
(a) Short flow (10 KB). (b) Medium flow (100 KB). (c) Long flow (1 MB).

Fig. 6: FCTs for different sized flows with different packet loss rates.

2) Evaluations: We set up random packet loss at the core switch. We evaluate the solution at four packet loss rates: 0.1%, 0.5%, 1%, and 5%. The sender transmits data to the receiver in sizes including 10 KB, which is smaller than a BDP, 100 KB, which lasts for multiple RTTs, and the larger 1 MB. We perform 200 experiments for each group; the results are shown in Fig. 6.

For short flows, packet loss primarily affects the tail FCT, as shown in Fig. 6(a). For example, at a 1% packet loss rate, retransmission occurs in approximately one out of every ten short flows on average. The victimized flow is forced to undergo more transmission time, raising more outliers. As the flow size increases, the occurrence of packet loss in each transmission becomes more frequent, resulting in an overall increase in average delay. When a connection experiences multiple packet losses, e.g., 100 KB of data at a 5% loss rate and 1 MB with over 0.1% loss rate, the GBN retransmission mechanism cannot effectively cope with out-of-order packets, triggering multiple backtracks and wasting bandwidth resources. As shown in Figs. 6(b) and 6(c), the average FCT doubles with the increase in packet loss. In contrast, NetRT provides in-network packet loss recovery that greatly improves performance. NetRT significantly reduces tail FCT for short flows and average FCT by up to nearly 75% for medium and long flows. In particular, as the packet loss rate increases, NetRT shows consistent performance and does not exhibit a significant increase in FCT as observed with GBN.

To evaluate the memory overhead of NetRT, we also track the occupancy of the duplicate pool and the reordering buffer (shown in Fig. 7). The duplicate pool grows with traffic input and then reaches a steady state. The overhead at steady state is around 128 KB. As for the reordering buffer, its overhead only appears when packet loss occurs. The peak is 37 KB, and the average is 15 KB. Therefore, the overhead of NetRT is acceptable in real deployments.



(a) Duplicate Pool. (b) Reordering Buffer.

Fig. 7: Memory overhead of NetRT duplicate pool and reordering buffer.

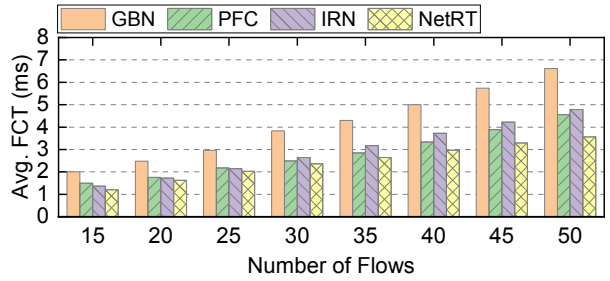
B. Micro-benchmarks

We further use simulations to build more complex topologies to validate NetRT's handling of congestion loss and cooperation with fine-grained load balancing.

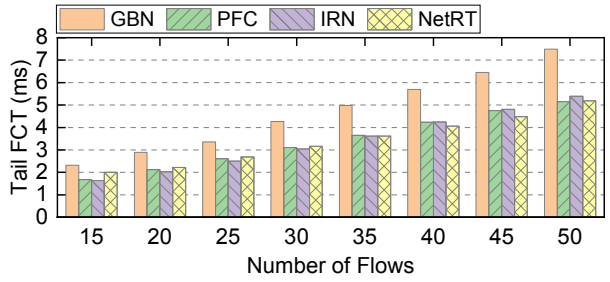
1) Congestion Loss: We build a many-to-one scenario to generate network congestion to verify the effectiveness of NetRT in lossy networks. In this scenario, each server is connected to a core switch via its ToR switch. The link bandwidth is set to 50 Gbps, and the RTT is about 5 μs. Senders transmit 500 KB of data sequentially to a single receiver at 10 μs intervals, creating continuous traffic bursts. We use DCQCN [5] as the congestion control algorithm. We vary the number of senders from 15 to 50 and measure the FCT for each solution.

Fig. 8 shows the average and tail FCT. As the number of concurrent flows increases, the network congestion escalates, leading to a high loss rate in lossy networks. The end-to-end GBN retransmission mechanism becomes inefficient, causing a significant drop in effective throughput and resulting in high FCT. By constructing a lossless network, PFC avoids the impact of congestion loss, offering an average FCT reduction of 25% to 35%. Due to the performance impairments of PFC, IRN chooses to deploy efficient SR in lossy networks. IRN significantly improves FCT and maintains stable performance as the number of flows increases. From feasibility considerations, NetRT deploys SR between programmable ToRs and benefits from a shorter packet loss recovery path. Due to the in-network congestion avoidance design, NetRT adapts to the increased network congestion and enables consistent high performance. As illustrated in Fig. 8, NetRT provides 34% to 47% improvement in the average FCT and 14% to 34% in the tail FCT compared to GBN.

2) Multipath Disorder: In data centers, multiple viable paths exist between servers, and load balancing may result in out-of-order packets by spreading packets from the same flow



(a) Average FCT versus number of concurrent flows.



(b) Tail FCT versus number of concurrent flows.

Fig. 8: Average and tail FCT with a different number of flows.

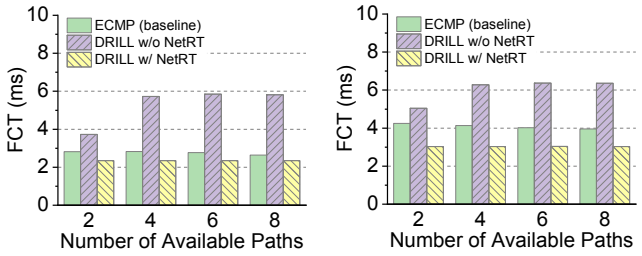
over different paths. We thus evaluate NetRT's ability to handle packets arriving out of order. We set up 15 senders connected to multiple core switches via a ToR switch, transmitting 1 MB of data to a single receiver. Consequently, there are several available paths for each flow to reach the receiver. We use ECMP [31] as the baseline and DRILL [11] that uses per-packet switching as the fine-grained load-balancing algorithm. We vary the number of available paths and compare the behavior of RoCE with and without NetRT.

Fig. 9 illustrates the average and tail FCT for each solution. As a benchmark solution, ECMP exhibits decreasing FCT with increasing available paths due to the reduced load on each path. However, when a fine-grained scheme is enabled, the out-of-order packets caused by switching paths result in undesired performance degradation. As the number of available paths increases, the network conditions become more complex, leading to increased performance impairments, which ultimately doubles the average FCT and increases the tail FCT by 61%. To further understand the impact, we count the number of load-balancing switching paths and the number of triggered retransmissions of RNICs, as shown in Table III. With the adoption of DRILL, an average of 68.8 path switches occur, resulting in approximately five retransmissions per 100 packets. Due to the lack of reordering capability of GBN, constant retransmissions cause a drop in effective throughput and longer FCT. In contrast, NetRT enables strong disorder tolerance and ensures that switching does not trigger end retransmission, enhancing transmission efficiency. Compared to the DRILL without NetRT, it reduces the average FCT by 37%∼60% and the tail FCT by 40%∼52%. Additionally, this result also proves that NetRT can work better with fine-grained load-balancing schemes for better link utilization. Compared to ECMP, DRILL with NetRT provides 11%∼17% average FCT reduction and up to 29% tail FCT reduction.

C. Large-scale Simulations

We further use large-scale simulations to evaluate NetRT. The topology used is a two-level fat-tree [32]. We set up 4 core switches with 16 ToR switches fully connected. Each ToR switch connects to 32 servers, totaling 512 servers across the topology. The links between the core switches and the ToRs are 100 Gbps, and the ToRs have 50 Gbps links connecting to the servers. DCQCN is the congestion control algorithm at the ends, and we use ECMP as the load-balancing algorithm. Traffic generation follows a Poisson distribution, with each sender randomly selecting its destination. The flow size distribution is based on $WebSearch$ traffic [33].

We measure average and 90^{th} percentile FCT, and the results are shown in Fig. 10. Due to the inefficient GBN retransmission mechanism, RNICs have a low tolerance for network packet loss. Once network congestion and packet loss occur, it leads to a decrease in effective throughput and an increase in the FCT. As shown in Figs. 10(a) and 10(b), it exhibits the worst average FCT when no other mechanism is adopted. PFC prevents queue overflow and enables lossless networking by suspending the upstream queue, therefore effectively avoiding low effective throughput due to frequent backoffs. In this experimental scenario, PFC reduces the average FCT for the traffic by a maximum of 16.7%. However, the back-pressure feature of PFC and the congestion spreading problem lead to queue buildup, which particularly affects the performance of short flows. For instance, for the short flows of 10 KB, PFC increases the average transmission time by 10%. In addition, PFC primarily affects the tail delay performance. As shown in Fig. 10(c), short flows with PFC exhibit high tail delays, up to more than twice that of GBN. IRN and NetRT deploy efficient selective retransmission, enabling high packet loss recovery capability. IRN has advantages in shortening the FCT of short

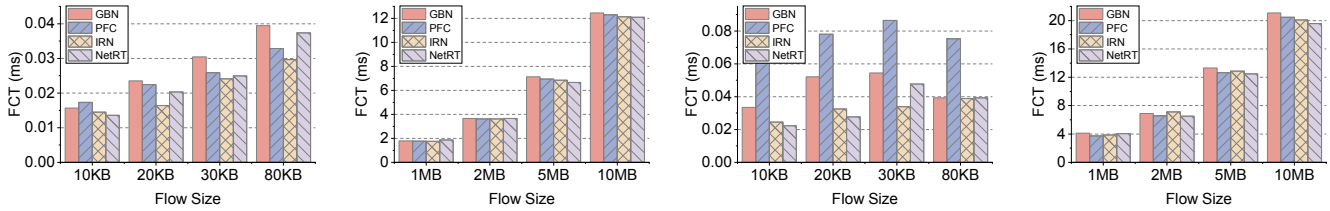


(a) Average FCT. (b) Tail FCT.

Fig. 9: Average and tail FCT under multipath disorder.

TABLE III: Average number of switching paths and end retransmissions per flow per 100 KB of data for each solution.

	Switching Path	**End GBN**
RoCE w/o NetRT	68.8	5.1
RoCE w/ NetRT	59.3	0



(a) Average FCT of short flow. (b) Average FCT of long flow. (c) 90^{th} percentile FCT of short flow. (d) 90^{th} percentile FCT of long flow.

Fig. 10: Average and 90^{th} percentile FCT of $WebSearch$ traffic load under GBN, PFC, IRN, and NetRT.

flows and offers up to 30% improvement. However, IRN uses a BDP-based flow control mechanism to limit the number of in-flight packets, which may result in impaired throughput in long flows, and thus slightly lower performance than NetRT in long flows. NetRT reduces the number of end retransmissions and enhances transmission efficiency with high feasibility through its in-network recovery strategy. NetRT reduces average FCT by 13.5% to 24.5% for short flows and reduces tail FCT by up to 46.8%. As the flow size increases, the transmission time spans more RTTs, and congestion control takes effect, so the impact of packet loss decreases, but NetRT still maintains optimal FCT performance.

V. Related Work

In this section, we present works aimed at enhancing the transmission robustness of RoCE.

Constructing Lossless Networks. Given that RNICs struggle with out-of-order packets, RoCE introduces PFC [6] to construct lossless in the initial deployment phase. With the revelation of PFC's performance impairments, some solutions aim to reduce PFC triggering. These solutions target suppressing switch queue lengths by utilizing precise congestion signals, shortening control loops, and performing accurate rate adjustments, as exemplified by HPCC [34], RoCC [35], and Bolt [36]. Besides, some solutions attempt to refine the flow control mechanism to avoid PFC's drawbacks. GFC [9] manipulates the port rate at a fine granularity to solve network deadlocks. BFC [37] provides per-hop per-flow flow control to mitigate congestion spread.

Enhancements for RNICs. Rather than relying on PFC, several solutions explore the possibility of running RDMA over lossy networks. Their core idea is to replace GBN with efficient SR on RNICs with limited hardware resources. MELO [38] separates data and metadata and stores data off chips. It avoids high on-chip overheads by placing buffers in user space. IRN [15] carefully designs the data structures required for retransmission, paired with BDP-based flow control to bound the number of inflights, ultimately deploying SR with less than 10% additional overhead. Flor [23], a framework for heterogeneous networks, develops a software SR mechanism regarding reliability, i.e., memory overhead uploading. SRNIC [16] co-designs transport protocols and architectures to address the overheads of SR deployment. Moreover, vendors like NVIDIA are gradually introducing commodity RNICs with SR support (e.g., ConnectX-6 [24]).

While these solutions significantly improve transmission robustness, the complexity and expense caused by updating architectures and replacing devices show significant problems.

In-network Assistance for Reliability. Noting the potential for in-network assistance, some solutions enable switches to carry out reliability functions. SQR [39] and LinkGuardian [22] guarantee link-level reliability through retransmissions between switches upstream and downstream of links. SQR focuses on link failure detection and packet in-network recovery. LinkGuardian addresses random packet loss due to link corruption. Both solutions lack end-to-end performance guarantees. SLR [14] switch accelerates end retransmissions by actively sending NACKs to reduce flow completion time. ConWeave [13] performs reordering after rerouting on ToR switches to achieve better load-balancing results without out-of-order. However, ConWeave assumes a lossless network and relies on PFC.

VI. Conclusion

In this paper, we discussed the necessity of enhancing RDMA resilience and the deployment limitations of existing solutions for improving RNICs. To solve this dilemma, we proposed NetRT, an innovative in-network retransmission offloading solution deployed on Top-of-Rack (ToR) switches. By leveraging the relatively abundant memory resources and programmable features of modern switches, NetRT performs transparent and efficient ToR-to-ToR selective retransmission to enhance RDMA transmission efficiency without requiring hardware modifications to RNICs. We implemented NetRT on our testbed, demonstrating its feasibility and acceptable overhead. Extensive simulation experiments show that NetRT effectively copes with various packet loss and out-of-order cases, significantly reducing flow completion time.

Acknowledgment

We would like to thank the anonymous reviewers for their constructive comments. This work is supported in part by the National Natural Science Foundation of China under Grant No. 62302472, Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant No. Y202093, and JSPS KAKENHI under Grant No. JP19H04105.

REFERENCES

[1] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "ATP: In-network aggregation for multi-tenant learning," in *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021, pp. 741–761.

[2] W. Wang, M. Moshref, Y. Li, G. Kumar, T. S. E. Ng, N. Cardwell, and N. Dukkipati, "Poseidon: Efficient, robust, and practical datacenter CC via deployable INT," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023, pp. 255–274.

[3] Y. Dong, Y. Dai, M. Xie, K. Lu, R. Wang, J. Chen, M. Shao, and Z. Wang, "Faster and scalable MPI applications launching," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 2, pp. 264–279, 2024.

[4] "NVIDIA Mellanox ConnectX-5," https://www.nvidia.com/en-us/networking/ethernet/connectx-5/, accessed on July 2024.

[5] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale RDMA deployments," in *Proceedings of the 2015 ACM Special Interest Group on Data Communication (SIGCOMM)*, 2015, pp. 523–536.

[6] "IEEE 802.1 Qbb - Priority-based Flow Control," https://1.ieee802.org/dcb/802-1qbb/.

[7] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, "Re-architecting congestion management in lossless ethernet," in *Proceedings of the 17th Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, 2020, pp. 19–36.

[8] Y. Zhang, Y. Liu, Q. Meng, and F. Ren, "Congestion detection in lossless networks," in *Proceedings of the 2021 ACM Special Interest Group on Data Communication (SIGCOMM)*, 2021, pp. 370–383.

[9] K. Qian, W. Cheng, T. Zhang, and F. Ren, "Gentle flow control: avoiding deadlock in lossless networks," in *Proceedings of the 2019 ACM Special Interest Group on Data Communication (SIGCOMM)*, 2019, pp. 75–89.

[10] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy, and T. Anderson, "Understanding and mitigating packet corruption in data center networks," in *Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2017, pp. 362—-375.

[11] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro load balancing for low-latency data center networks," in *Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2017, pp. 225—-238.

[12] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 407–420.

[13] C. H. Song, X. Z. Khooi, R. Joshi, I. Choi, J. Li, and M. C. Chan, "Network load balancing with in-network reordering support for RDMA," in *Proceedings of the 2023 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2023, pp. 816––831.

[14] Q. Meng, Y. Zhang, S. Zhang, Z. Wang, T. Zhang, H. Luo, and F. Ren, "Switch-assistant loss recovery for RDMA transport control," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2069–2084, 2024.

[15] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting network support for rdma," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018, pp. 313—-326.

[16] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng, T. Wang, W. Ling, K. Huo, P. An, K. Ji, S. Zhang, B. Xu, R. Feng, T. Ding, K. Chen, and C. Guo, "SRNIC: A scalable architecture for RDMA NICs," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023, pp. 1–14.

[17] "Intel Tofino 2," https://www.intel.com/content/www/us/en/products/sku/218647/intel-tofino-2-6-4-tbps-4-pipelines/specifications.html, accessed on July 2024.

[18] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019, pp. 1–16.

[19] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[20] "Network Simulator 3," https://www.nsnam.org/about/, accessed on July 2024.

[21] S. Hu, Y. Zhu, P. Cheng, C. Guo, K. Tan, J. Padhye, and K. Chen, "Deadlocks in datacenter networks: Why do they form, and how to avoid them," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNet)*, 2016, pp. 92–98.

[22] R. Joshi, C. H. Song, X. Z. Khooi, N. Budhdev, A. Mishra, M. C. Chan, and B. Leong, "Masking corruption packet losses in datacenter networks with link-local retransmission," in *Proceedings of the 2023 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2023, pp. 288–304.

[23] Q. Li, Y. Gao, X. Wang, H. Qiu, and et.al., "Flor: An open high performance RDMA framework over heterogeneous RNICs," in *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2023, pp. 931–948.

[24] "NVIDIA Mellanox ConnectX-6," https://www.nvidia.com/en-us/networking/ethernet/connectx-6-dx/, accessed on July 2024.

[25] "Trident-3," https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56870-series, accessed on July 2024.

[26] "NVIDIA Mellanox Spectrum-3," https://network.nvidia.com/files/doc-2020/pb-spectrum-3.pdf, accessed on July. 2024.

[27] "Tomahawk 4," https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series, accessed on July 2024.

[28] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019, pp. 1–16.

[29] "Intel Ethernet Network Adapter E810," https://www.intel.com/content/www/us/en/products/details/ethernet/800-network-adapters/e810-network-adapters/products.html, accessed on July 2024.

[30] "DPDK," https://www.dpdk.org/about/, accessed on July 2024.

[31] C. Hopps, "RFC2992: Analysis of an equal-cost multi-path algorithm," 2000.

[32] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, 1985.

[33] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proceedings of the 2010 ACM Special Interest Group on Data Communication (SIGCOMM)*, 2010, pp. 63–74.

[34] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "HPCC: High precision congestion control," in *Proceedings of the 2019 ACM Special Interest Group on Data Communication (SIGCOMM)*, 2019, pp. 44–58.

[35] P. Taheri, D. Menikkumbura, E. Vanini, S. Fahmy, P. Eugster, and T. Edsall, "RoCC: Robust congestion control for RDMA," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2020, pp. 17–30.

[36] S. Arslan, Y. Li, G. Kumar, and N. Dukkipati, "Bolt: Sub-RTT congestion control for Ultra-Low latency," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023, pp. 219–236.

[37] P. Goyal, P. Shah, N. K. Sharma, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022, pp. 779–805.

[38] Y. Lu, G. Chen, Z. Ruan, W. Xiao, B. Li, J. Zhang, Y. Xiong, P. Cheng, and E. Chen, "Memory efficient loss recovery for hardware-based transport in datacenter," in *Proceedings of the First Asia-Pacific Workshop on Networking (APNet)*, 2017, pp. 22—-28.

[39] T. Qu, R. Joshi, M. C. Chan, B. Leong, D. Guo, and Z. Liu, "SQR: In-network packet loss recovery from link failures for highly reliable datacenter networks," in *Proceedings of the IEEE 27th International Conference on Network Protocols (ICNP)*, 2019, pp. 1–12.