Fast and Scalable Selective Retransmission for RDMA

Peihao Huang[1,2], Guo Chen[2,*], Xin Zhang[2], Can Liu[2], Hongyu Wang[2], Huijun Shen[2], Ying Bian[2], Yuanwei Lu[3], Zhenyuan Ruan[4], Bojie Li[5], Jiansong Zhang[6], Yongfeng Liu[7], Zhigang Chen[1,*]
[1]*Central South University,* [2]*Hunan University,* [3] *StepFun,* [4] *MIT,* [5]*Huawei,* [6] *China Telecom,* [7] *Yunsilicon*

***Abstract*—RDMA, with its high throughput, ultra-low latency, and low CPU utilization, has been widely used in large-scale data centers. However, due to the limited RDMA RNIC on-chip memory, commodity RDMA usually implements the simple Go-Back-N (GBN) loss recovery mechanism, which consumes less memory but leads to a significant performance drop when encountering loss. Recent works try to improve RDMA performance under packet loss by introducing selective retransmission (SR) to it. Nevertheless, implementing efficient SR in RDMA remains challenging. Specifically, either it consumes too much memory for maintaining SR states which leads to poor connection scalability, or it incurs high CPU consumption and latency for onloading SR processing back to the CPU software.**

To this end, we propose FaSR, a fast and scalable RDMA selective retransmission. It is fast by processing the SR with 200Gbps+ line-rate fully on the RDMA NIC chip, and is scalable by introducing novel SR state management schemes thus consuming small memory even under high concurrency. Specifically, FaSR adopts a dynamically sharing SR structure among connections to reduce the memory footprint by orders of magnitude when the concurrency is high. Also, utilizing the loss recovery pattern, FaSR devises several techniques thus it can access the sharing structure with line-rate for different connections.

We have implemented FaSR in Xilinx FPGA board with ∼4000 lines of verilog code. Testbed evaluation demonstrates that FaSR can maintain 92%+ throughput at a packet loss rate of 1% under more than 5K concurrent connections, which is 16% and 12.6x higher compared to the latest RDMA SR solution and commodity RDMA NICs, respectively.

***Index Terms*—Datacenter networks, Loss recovery, Hardware memory.**

I. Introduction

Datacenter applications have progressively high requirements for network speed, latency, scale, concurrency, and fault tolerance. Specifically applications such as distributed machine learning training and online services [2], [11], [16], [25] require a network with extremely high bandwidth and low latency; cloud storage, such as Alibaba Pangu [4], requires each terminal to maintain thousands of concurrent connections to provide mesh communications between chunk servers and block servers; cloud migration, cloud disaster recovery, and multicenter collaborative computing necessitate a high level of fault-tolerant to adapt to lossy cross-datacenter transmission.

With low latency, high throughput, and bypass CPU, Remote Direct Memory Access (RDMA) has become a hot topic in large data center networks (DCNs) [3], [12], [21], [24], [29]. Also, with these advantages, RoCEv2 (RDMA over Converged Ethernet version 2) has become the de facto standard for high-speed networking in modern data centers [30].

To offer consistently high performance, RDMA must deal with out-of-order (OoO) packets caused by packet loss efficiently [15], [22], [23], [29] which is unavoidable for three reasons. Firstly, heterogeneous, ultra-long-distance networks between data centers are challenging to achieve losslessness. Secondly, although Priority Flow Control (PFC) was introduced to prevent congestion-caused loss in datacenters, it is a daunting task to make the PFC work correctly and efficiently in a large datacenter [3], [12], [21], [24], [30]. In a very large network, PFC can easily cause problems such as head-of-line blocking, congestion spreading, occasional deadlocks, and PFC storms [14], [30]. Lastly, packet loss caused by software/hardware failures or bugs still occurs frequently, which is inevitable even in well-engineered large datacenters [6], [13] (*e.g.*, a 2% random packet loss had lasted for two days at a spine switch in a Microsoft's production datacenter [13]).

Current RDMA, however, behaves very poor in case of packet loss. Due to the limited RNIC on-chip memory, commodity RDMA usually implements the simple Go-Back-N (GBN) loss recovery mechanism [8], [9], inefficient in retransmitting lost packets. The performance of RDMA loss recovery can be greatly improved by adopting SR [22], [23], [29]. However, how to implement efficient RDMA SR still remains unsolved, which is *hard to simultaneously reach the connection scalability (concurrency) and the packet processing speed*. Particularly, IRN [23] sacrifices the connection scalability for the loss recovery efficiency. To record the packets that arrived out-of-orderly and selectively retransmit the lost ones, IRN requires additional data structures including a 5 BDP[1]-sized bitmap (*e.g.*, ∼500B) and 20B SR state per connection. It significantly increases each connection's memory footprint on the NIC by about 2x, greatly decreasing the performance when the connection concurrency is high. On the contrary, SR-NIC [29] avoids the NIC memory consumption by onloading all SR processing back to the CPU software. In case of loss, SRNIC will pass all the lost packets to the CPU, and rely

∗: Corresponding author.

This work was done when Peihao Huang worked as a visiting student at Hunan University.

[1]Bandwidth-delay-product.

on software to record OoO packets, generate selective ACK (SACK) header options, and selectively retransmit the lost ones. However, the interaction between the NIC and the CPU software incurs significant delay, and the software processing is slow and has unstable latency. As such, it also loses throughput when loss happens even when the concurrency is not high (*e.g.*, a single SRNIC connection only reaches about 75% throughput at 1% loss rate).

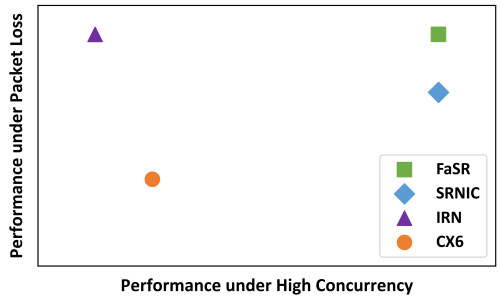


Figure 1. Comparison of Different RDMA Loss Recovery Solutions.

In this paper, we propose FaSR, a *Fa*st and *S*calable selective *R*etransmission for RDMA. As shown in Fig. 1, FaSR maintains comparable high concurrency performance to SRNIC while providing transmission performance as "fast" as IRN in a lossy network environment.

First, FaSR is fast by enhancing SR logic (§III-B) that eliminates the necessity for most SR operations to bypass bitmaps. As a result, FaSR can process the SR entirely on the RDMA NIC (RNIC) at line-rate. Second, to be also scalable, FaSR introduces a novel 2-level SR state management scheme (§III-C), which only consumes a small and constant value of memory (§III-D), regardless of the number of concurrent connections. Furthermore, FaSR designs the transport pipeline so the SR process will not interfere and delay other transport parts such as congestion control.

We have implemented FaSR in a 100Gbps Xilinx FPGA board with ∼4000 lines of Verilog code (Our implementation is open-sourced at [1]). With the aforementioned techniques, our FaSR implementation can reach full 100Gbps line-rate processing, and scales to 200Gbps+ line rate according to cycle-based calculation. Moreover, compared to GBN, FaSR introduces an average of only 10bits SR states per connection to efficiently record OoO packets for all connections at 1% packet loss rate. Testbed evaluation demonstrates that FaSR can maintain 92%+ throughput at a packet loss rate of 1% under different levels of concurrency, which is up to 16%, 31%, and 12.6x higher compared to SRNIC, IRN, and Nvidia CX6-DX commodity RNIC, respectively.

II. Background and Motivation

A. RDMA background

RDMA facilitates direct memory access to a remote server by fully offloading the entire network stack onto RNICs. An RDMA connection is identified by a queue pair (QP), which comprises a send queue (SQ) and a receive queue (RQ). Applications post RDMA messages to the RNIC QP via work-queue elements (WQEs). Subsequently, the RNIC encapsulates these messages into multiple packets and sends them to the remote side. Upon receipt, the remote RNIC decapsulates the packets, reassembles them into messages, and directly deposits them into the application buffer via DMA. During transmission, sending and decapsulating packets require relevant Queue Pair Context (QPC) information, such as congestion window size for sending, following packet sequence number (PSN) for receiving, and DMA-related status. Commercial network cards, such as CX3 [8], allocate 256B of memory overhead per QP to store the QPC, which gradually increases as more features are incorporated.

Loss recovery in commodity RDMA. To reduce memory footprint, traditional RDMA only implements GBN loss recovery mechanism, which retransmits all packets starting from the first lost packet, regardless of whether subsequent packets have been successfully received. Our evaluation on popular commodity RNICs, *i.e.*, Nvidia CX6-DX [9] shows that at a packet loss rate of 1%, throughput drops dramatically to less than 10%. Therefore, to maintain performance, the commodity RDMA is based on PFC to eliminate congestion-induced loss, which is hard to configure and operate in large-scale DCNs [11], [12], [30].

SR brings new problems. Recent works [22], [23], [29] have attempted to improve RDMA performance under packet loss by introducing SR to replace GBN. These approaches utilize bitmaps to keep track of both lost and successfully received packets [22]. As mentioned in IRN [23] and SRNIC [29], each QP requires five BDP-sized bitmaps (500 slots for a bitmap to fit the BDP cap of a network with bandwidth 100 Gbps and RTT 40 µs [17], [29]). Consequently, for 5K QPs, a total of 1.5 MB of memory is consumed by these bitmaps. In scenarios with larger RTT, such as cross-DCN communication, the overhead of bitmaps becomes even more significant. However, RNICs only have about 2 MB of Static RAM (SRAM) to store QPCs and other data structures and buffers [18].

The presence of bitmaps can have a noticeable impact on the connection scalability of the RNIC, potentially decreasing the number of QPs that can be supported. The connection scalability problem is a critical concern.

How to solve the connection scalability problem? This issue cannot be solved by simply increasing the SRAM capacity. This is because SRAM already accounts for a large portion of the RNIC's power consumption. Expanding SRAM capacity would require higher power consumption and better heat dissipation. Additionally, larger capacity would lead to larger circuit area, resulting in increased latency and limiting the RNIC's maximum bandwidth.

SRNIC [29] solved this issue by uploading bitmaps to host and relying on the CPU to perform SR, which indeed saves on-chip memory within the RNIC. However, its connection scalability is achieved at the expense of throughput in lossy networks. Because the receiver needs to pause the pipeline to wait for the CPU's bitmap query results after receiving the retransmitted packet. This latency exerts an adverse impact on throughput. We measured the latency of software SR query on

our testbed and found that the latency was as high as 1.4us at 1% packet loss. Among them, PCIe latency accounts for 1.2us of this latency and hard to be optimized. In addition, as shown in Fig. 2, the higher the packet loss rate, the higher the query frequency, the greater the impact of software SR latency on throughput. This latency has a greater impact on NICs with higher bandwidth. Since higher bandwidth causes packets to arrive faster. Every time the pipeline waits for the result of a software SR query, the backlog of packets waiting to be processed will also increase, and the loss of throughput at 1% packet loss on 400Gbps bandwidth can be as high as 40%.

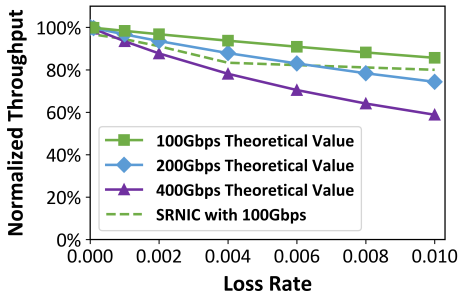


Figure 2. Impact of 1.4us SR latency on throughput at different RNIC bandwidths.

B. Challenges for a fast and scalable SR, and FaSR's intuition

To be both fast and scalable, FaSR on RNIC needs to address the following key challenges.

How to reduce the RNIC memory footprint for SR related states? Once a packet is lost, the number of OoO packets could be very large which is linear to the network BDP [22] (*e.g.*, hundreds of packets). Moreover, as the bandwidth rapidly grows from 100Gbps to 400Gbps, the DCN BDP becomes 4x larger than before. Apparently, maintaining a bitmap of such large size for each QP consumes too much RNIC memory that degrades the scalability. Simply reducing the size of each QP's bitmap does not work since the OoO packets that are not recorded still need to be retransmitted in GBN manner, which affects throughput [8].

*FaSR intuition.*We observe that the total number of OoO packets and lost packets of all QPs on a RNIC is limited by the network BDP. Particularly, the time from a lost packet is detected to its retransmitted one arrives at the receiver is bounded, which is about a round-trip time (RTT)[2]. At most, the number of OoO packets equal to BDP will reach the receiving RNIC as all concurrent QPs share its receive port bandwidth. Moreover, the number of lost packets is far less than the BDP. Even if each lost packet belongs to a different QP, it remains much smaller than the number of concurrent QPs on the RNIC (e.g., tens versus thousands). As such, FaSR takes a novel 2-level SR data share structure, which is shared by all the QPs on an RNIC to record each QP's SR states and OoO packets, thus greatly reducing the per-QP memory footprint.

[2]RDMA network usually has low send queue length due to precise congestion control [10], [21], [24], [30]. Also, the host processing delay is negligible due to bypassing the CPU.

How to access the shared bitmap in line-rate? Given the RNIC's limited processing power, the bitmap sharing technique must be simple and quick enough to finish the bitmap operation before the next packet arrives. Paging and segmentation aren't suitable for handling bitmaps in RNIC. The memory overhead of the host application remains consistent. But, QP disorder varies with each OoO packet, frequently relocating bitmap in the shared pool for memory use. This makes segmentation management inefficient and incurs significant hardware implementation costs and management overhead. Paged storage management is easy and quick yet requires a table to track shared memory page use. Despite bitmap paging size being mere bits, table overhead is too great. The database must be searched up every time the bitmap is utilized, adding delay. Also, all table entries for that QP need to be queried and modified when the bitmap is released during disordered packet recovery, which is expensive in latency.

FaSR intuition. We observed that paging constitutes a beneficial technique; however, a more appropriate approach for bitmap management is required. We discovered that there exist regularities in both the arrival pattern of OoO packets and the number of packet losses within a QP. Firstly, based on the observation that OoO packets access only the tail of the bitmap and retransmitted packets generally access only the head of the bitmap, we devised an efficient and low-latency bitmap management mechanism aimed at handling a large number of diverse connections and interleaved packets. Secondly, we found that as the number of concurrent QPs increases, the probability of a QP losing precisely one packet rises significantly. In response to this, we designed fast and slow SR paths, enabling SR to bypass the bitmap with a high probability and ensuring the low latency.

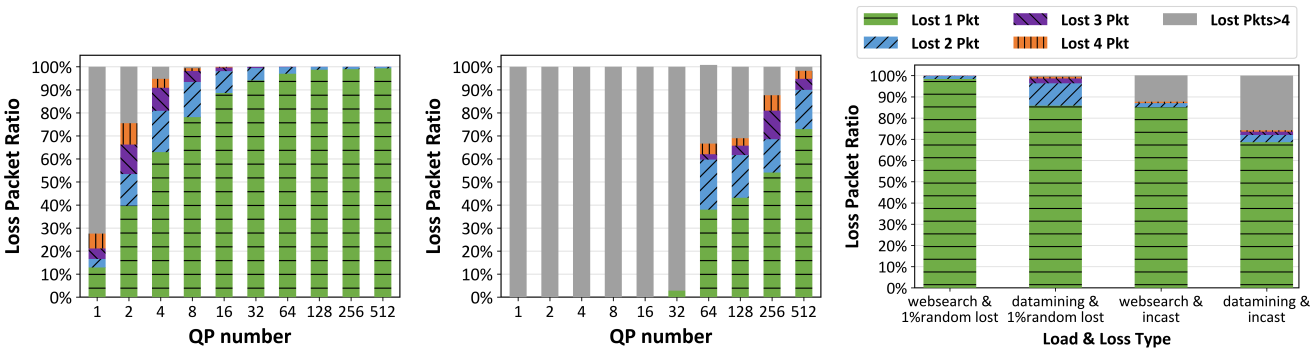
How the SR logic interacts with other parts in the existing transport? After introducing the new shared structure of SR states, it will inevitably impact the original RNIC transport processing. With time constraints, complex operations such as retransmission, packet reception, and packet transmission must be processed through multiple stages in a pipeline to complete the overall operation within 10s of ns (100Gbps+ and 1K packet size). Designing the pipeline to avoid interference among the three functions of packet processing, packet retransmission, and packet transmission and achieving high processing bandwidth is also challenging.

FaSR intuition. To avoid state machine blocking from affecting performance, FaSR has rearchitected the transport layer structure to allow for parallel packet receiving, sending, and OoO packet recording and retransmitting without interference.

III. FaSR Design

A. Overview

Fig. 4 shows the overview of FaSR. The blue arrow represents the data path where OoO packet data will be directed DMA to the reorder buffer according to the packet header information(§IV-D) and then submitted together to the application after the lost packets are retransmitted. The red solid line and dashed line arrow represent the fast and slow paths



(a) Packet loss pattern under random loss (b) Packet loss pattern under incast (c) Packet loss pattern under real workload

Figure 3. Pattern of Packets lost number per flow under different lossy scenarios.

we designed. This allows the recording of OoO packets and the retransmission of lost packets to bypass the bitmap with a high probability. As a result, FaSR can effectively reduce complexity and latency while providing improved bandwidth extensibility (§III-B). We also introduce a novel 2-level Shared SR pool to control the SR state overhead of each QP. This pool is dynamically allocated based on the degree of QP out-of-order, which helps minimize the metadata required to track OoO packets (§III-C).

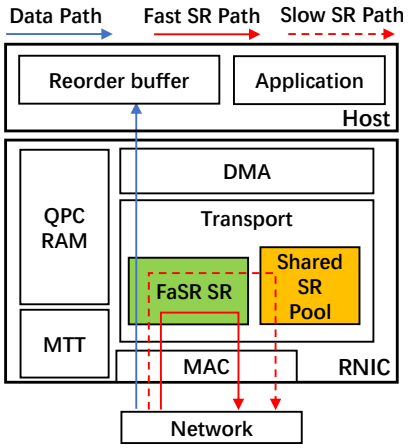


Figure 4. FaSR Overview.

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

B. FaSR SR logic

We have done comprehensive experimental analysis and found that not all lost packets need bitmaps. We built a basic unit of a typical fattree topology in NS3, with all bandwidth of 200Gbps, maximum transmission unit (MTU) set to 1024B, BDP size of 500, and the sender load set to 90% bandwidth. At the receiver, we measured the impact of 1% packet loss and incast (3V1) loss on the distribution of OoO packets of a single QP under different concurrency. As shown in Fig. 3 (a) and (b), with the number of concurrent QPs increased, The probability of QP losing only one packet during each recovery period for packet loss exceeds 70%. As shown in Fig. 3 (c), this pattern also holds for actual traffic loads. If QP loses only one packet, it means the receive next (RCV-NXT) is lost, and all subsequent packets are in order. Therefore, there is no need to utilize bitmaps for the recover of lost packets. Instead, the highest selective ACK(sack-high), can be employed to record such information.

FaSR designed a loss counter (lost-cnt) per QP, which records the lost packets at the receiver and determines whether it is data loss or SACK loss at the sender.

1) Receiver Fast & Slow Path: Similar to previous work [22], [23], [29], FaSR receiver detects packet loss by comparing the PSN to the RCV-NXT. In the normal state, lost-cnt will increase according to the interval between PSN and RCV-NXT. In the loss recovery state, lost-cnt will increase according to the interval between PSN and sack-high, and will decrease after receiving the retransmitted data packet.

Receiver fast path. When lost-cnt equals 1, only RCV-NXT is lost and fast path is selected. Only sack-high will be updated if FaSR receives an OoO packet. When a retransmission packet (PSN=RCV-NXT) is received, the loss-recovery state is exited, and RCV-NXT is directly updated to sack-high + 1.The fast path is fast as it does not rely on bitmap and can be executed through simple logic.

Receiver slow path. If lost-cnt is greater than 1, slow path is selected, and the bitmap is enabled for OoO recording and querying. After receiving a retransmission packet, the bitmap is still queried to obtain the updated value of RCV-NXT. The updated RCV-NXT is then sent back to the sender through an ACK (AACK).

Fast and slow path switching. Once lost-cnt decreases to 1 again, FaSR switches from the slow path to the fast path, and the bitmap is disabled. FaSR incorporates a bitmap compression acceleration mechanism(§III-C) in conjunction with the shared bitmap to mitigate the delay associated with switching between the fast and slow paths.

Finally, lost-cnt will be piggy-backed in SACK packet (extended header) and returned to the sender. This ensures that the sender and receiver remain on the same path, avoiding the sender's inaccurate estimation of packet loss owing to SACK loss and lowering unnecessary bitmap overhead.

2) Sender Fast & Slow Path: The information contained in the SACK makes the sender more likely to maintain the SR operation on the fast path and no bitmap is required. The

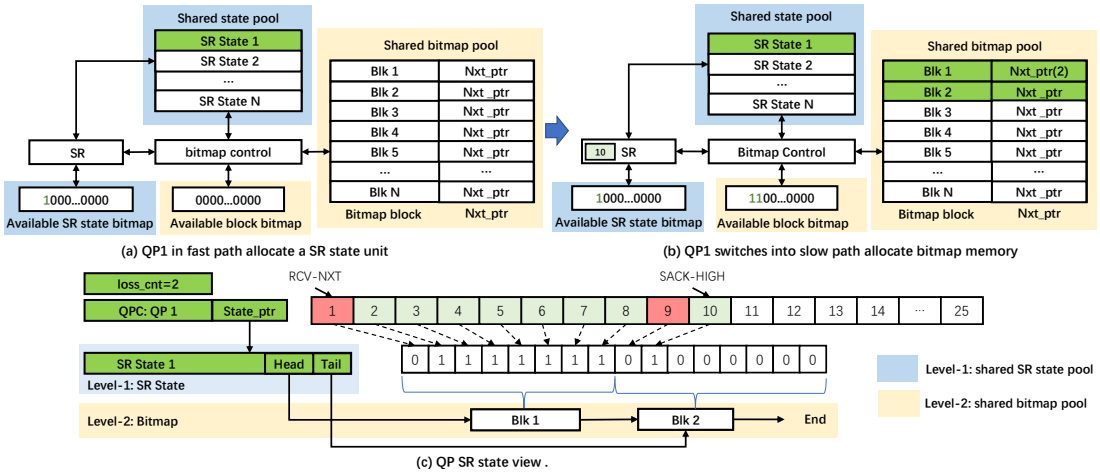


Figure 5. Shared SR Pool Structure.

sender does not locally update its lost-cnt. Rather, it relies on the lost-cnt transmitted by the receiver, comparing it to the previously received lost-cnt before reaching a decision. By comparing the spacing between SACK and sack-high and the change of lost-cnt, it can be determined whether the packet is lost or SACK is lost.

Selective retransmission is in fast path. Upon receiving the SACK, the sender will enter the loss recovery state and immediately retransmit the packet between the unacknowledged PSN (UNACK) and the SACK, regardless of the value of lost-cnt. Also, during the loss recovery state, the sender will retransmit the packet between SACK and sack-high only when there is a discontinuity between the received SACK and sack-high and the lost-cnt returned by the receiver increases.

Updating UNACK is "basically" fast path. The bitmap is also unnecessary for the UNACK update. This is because the ACK received from the receiver already includes the most recent UNACK value. So, if the bitmap is already in use, it's updated in parallel.

RTO can be either a fast or slow path. When retransmission timeout (RTO) is triggered, the behavior will depend on the value of lost-cnt. If lost-cnt is greater than 1, the UNACK will be retransmitted immediately, and the bitmap will be consulted to identify any zeros between the UNACK and sack-high for retransmission. If lost-cnt equals 1, only the UNACK will be retransmitted without querying the bitmap. However, if lost-cnt is 0, it indicates that all packets within the entire window of the QP have been lost. In this case, the sender will resend starting from UNACK.

C. Two-level shared SR Pool

As shown in Fig. 5(a), FaSR divides the additional memory overhead required for SR into two levels of shared pools: the first level is the shared SR state pool managed by SR module, and the second level is shared bitmap pool managed by bitmap control module. By implementing this two-level shared SR state pool, FaSR optimizes memory usage by allocating resources selectively to QPs that are in specific need.

1) Level-1: shared SR state pool: As shown in Figure 5(a) and (c), the Level-1 pool is divided into two parts. An SR state pointer needs to be added to the QPC of each QP. When there is no packet loss in the QP, this pointer is set to null. When the QP has packet loss and is in the fast SR path, SR states will be allocated to it, but bitmaps will not be assigned. At this time, the pointer points to the starting address of the SR states allocated to the QP in the Level-1 pool.

Shared state pool. Consists of several fixed-sized SR state units, each of which consists of a complete set of SR-required states [23] and pairs of bitmap start and end pointer and lost-cnts. These pointers are used to indicate the start and end addresses of the bitmaps allocated to the QP in the level-2 bitmap pool when lost-cnt is greater than 1.

Available SR state bitmap. Initialized with all 0, indicating that all SR state units are available. When an SR state unit is assigned to a specific QP, the corresponding position in the bitmap is set to 1. This setting indicates that the unit is occupied and cannot be used by other QPs. When QP exits the loss-recovery state, the corresponding position in the SR states bitmap is set to 0, and can be used by other QPs.

2) Level-2: shared bitmap pool: As shown in Fig. 5(b) and (c), the Level-2 bitmap sharing pool is divided into three parts: Bitmap Blocks, Pointers, and Available block bitmaps.

The bitmap block is the smallest allocation unit for recording OoO packets. When a QP is in the slow SR path, FaSR allocates one or more bitmap blocks from the available block bitmap according to the OoO status of QP and connects them in a single linked list using pointers. When a retransmitted packet is received, the bitmap block is released from the head of the linked list, and the corresponding available block bitmap is marked as 0 for use by other QPs.

D. Other design Considerations

Memory size optimization. To optimize the memory consumption, we derive a theoretical model to analyze the lower limit of the shared bitmap's size. For space limitation, we only show the results here.

Various factors, such as congestion, switch queues, and the potential increase in OoO packets due to retransmission packet loss, need to be considered. In a scenario where the theoretical maximum BDP is 500 and the packet loss rate is 0.02, 20

SR state units with a size of 38B (SR state overhead 20B, 5 pairs of 1B bitmap tail pointer overhead 10B, 5*10b size compression-cnt and 5*3bit size lost-cnt overhead) and 70 bitmap blocks with a size of 10 bits can meet the needs of all concurrent QPs on the RNIC for OoO packets recording. The total memory overhead is 0.92 KB. Note that this is the total size shared by all connections on the RNIC.

Bitmap acceleration. When the SR module recognizes a QP transition to a slow path, it activates the bitmap to record OoO packets. If there is a significant offset between RCV-NXT and sack-high, it can affect the recording latency of the bitmap module. FaSR adds a 10-bit compression-cnt in the SR state, which can record up to 1k consecutive OoO packets starting from RCV-NXT. It records the offset value before the sack-high update, and the bitmap starts recording from the previous RCV-NXT to RCV-NXT plus compression-cnt.

If the offset between RCV-NXT and sack-high is less than the size of one bitmap block when transitioning to the slow path, the compression-cnt is set to zero, meaning bitmap acceleration is disabled.

Exhausting of shared pool. FaSR supports OoO records with a packet loss rate of 0.02. Although (§V-D3) proves that even if multiple QPs trigger RTO, it will not cause bitmap overflow, but in extreme cases, there is a small probability that the bitmap may be exhausted. When the bitmap is exhausted, FaSR automatically switches to GBN. After the QP returns to GBN, the recorded OoO will not be deleted, but QP will not be returned to SR until the loss recovery is completed.

Reducing bitmap random access. Linking lists are not ideal for random access to the middle block, which requires traversal from the front to locate the bitmap block. Fortunately, packet loss OoO makes bitmap access regular. OoO packets only access the linked list's end, whereas retransmitted packets access the bitmap from the head. FaSR holds bitmap head and tail addresses for low-latency access.

When the QP loses the initial retransmission packet, successive retransmission packets will access intermediate bitmap blocks, which increases bitmap latency. FaSR made corresponding by FNACK mechanism: if the receiver receives a packet with a PSN smaller than sack-high and larger than RCV-NXT, one retransmitted packet is considered lost. This packet will be discarded, and the PSN value is carried back in FNACK to notify the sender to trigger RTO in advance and start retransmission from RCV-NXT again. The sender only responds to an FNACK once before receiving a new ACK, although the receiver may send multiple.

Lost-cnt overflows. In our experiments, a 3-bit lost-cnt can satisfy over 90% of QP loss records. However, the lost-cnt may exceed its capacity. Once the receiver's lost-cnt overflows, it remains unchanged until the packet loss recovery is completed. The RCV-NXT value must be updated through bitmap queries. Concurrently, the receiver informs the sender about the lost-cnt overflow, and the sender no longer needs lost-cnt to increase to trigger retransmission.

IV. IMPLEMENTATION

As the commercial RDMA NIC protocol stack is fixed and unmodifiable, we implemented FaSR on the Xilinx FPGA board model XCVU13P. It features a PCIe Gen3 x16 interface, as well as two 100 Gbps Ethernet ports.

A. Hardware implementation

FaSR enables unmodified RDMA applications to submit transmission tasks to the SQ and RQ in RNIC, and get the completion notifications from Completion Queues(CQ).

Fig. 6 illustrates the implementation structure of FaSR, which is implemented at the transport layer.The DMA engine consists of a scheduler [29] and a DMA core. WQE will not be cached on the NIC but is acquired in advance through the cooperation of Schedule and Congestion Control (CC) before transmission, and the MAC serving as the basic NIC.

B. Transport layer structure

As shown in the bottom right part of Fig. 6, FaSR transport layer mainly consists of seven modules.

QPC MANAGER. Responsible for managing the on-chip QPC storage, checking the legality of the packets and hashing the received packets to corresponding QPC addresses.

SR. Distinguishes the packet type based on op_type and packet_type, and then performs corresponding actions. Upon receiving a new packet, it returns an ACK to ACKQueue. When a SACK is received, the retransmitted packet is sent to RetxQueue.

CC. Gets the latest QPC&WQE from the schedule and determine which packet to send.

Update Queue. Processes the write requests from QPC MANAGER, CC, and SR to QPC RAM in priority order.

RTO. QPC RAM is traversed to determine whether to trigger RTO. If an RD request times out, it will be resubmitted to QPC MANAGER. For other requests, the timeout information will be sent to the SR module for retransmission.

Send Queue. It is responsible for packing/unpacking, checking, and sending/receiving packets with priority.

QPC RAM. FPGAs only have dual-port RAM, which allows only one read or write operation on each port per cycle, thus necessitating several backups and incurring high storage cost to enable concurrent access. To prevent conflicts in four modules concurrent read-write operations, we partition each modules 's state variables into three distinct groups. In this way, the four modules can access the QPC RAM equally in each clock cycle.

C. Transport layer pipeline

As shown in the upper right of Fig. 6, to prevent inefficient mixing of different types of WQEs and packets within a single pipeline, as well as a scenario where a single module blocks all functionalities, FaSR establishes separate FIFO pipelines between the transport layer modules based on the specific functionalities required for various operation types and packet types. For example, RD operations initiated by the host and received packets do not need to go through DMA again. They

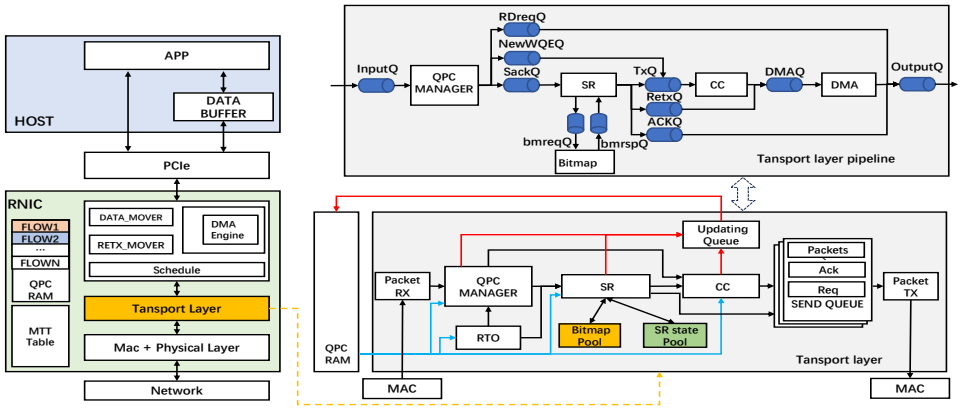


Figure 6. Implementation Structure.

can bypass the CC and DMA modules and directly enter the SendQueue through RDreqQueue and ACKQueue. Similarly, WRITE and SEND operations that are ready to be sent can bypass SR module and enter CC through NewWQEQueue.

By guiding operations and packets to the most suitable path, FaSR enables the transport layer to handle different types of operations and packets more efficiently according to specific needs. It reduces unnecessary processing and delays, consequently boosting overall transmission performance and reliability.

D. Packet format

To correctly DMA OoO packets into the buffer, we make the following extensions to the RDMA header so that the RNIC can infer the address of each packet by parsing the header of each packet: (1) All SEND packets carry the 3 bytes size send message sequence number (MSN) and 3 bytes for the payload offset [29], which can be used by the RNIC receiver to locate the offset in the corresponding receive WQE and its associated receive buffer. (2) All WRITE packets carry their target remote memory address, which costs 8 bytes [23], [29].

For READ, an acknowledgment mechanism is introduced to the READ requests; the requester adds a timestamp to the request that has not been ACKed, and R-READ (remote read), similar to the WRITE operation, is scheduled on the responder. In this way, SEND and WRITE header extensions can be applied to READ request and response packets.

For SACK, a 3-bit lost-cnt overhead is added. The lost-cnt overflow flag is marked using the unused op-type value.

The above extension adds a 6 to 14 bytes header to the packet, which results in a 0.58% and 1.3% decrease in application goodput with a 1024 bytes RoCE MTU.

E. FPGA resource overhead

We evaluate the resource consumption of implementing GBN and FaSR respectively. Compared with GBN, the FPGA resource overhead (flip-flops (FFs), look-up tables (LUTs), BRAMs) of FaSR increases by less than 0.7%.

V. Evaluation

We use testbed experiments to compare FaSR with Mellanox (Nvidia) RNICs, IRN, and SRNIC. Our results indicate the following key points: 1) FaSR achieves high efficiency on loss recovery, maintaining over 92% throughput at a 1% loss rate; 2) FaSR maintains high performance under high concurrency (5K concurrent connections); 3) FaSR's fixed-size shared SR pool is able to handle many concurrent connections and can record OoO packets in various concurrent scenarios with minimal delay.

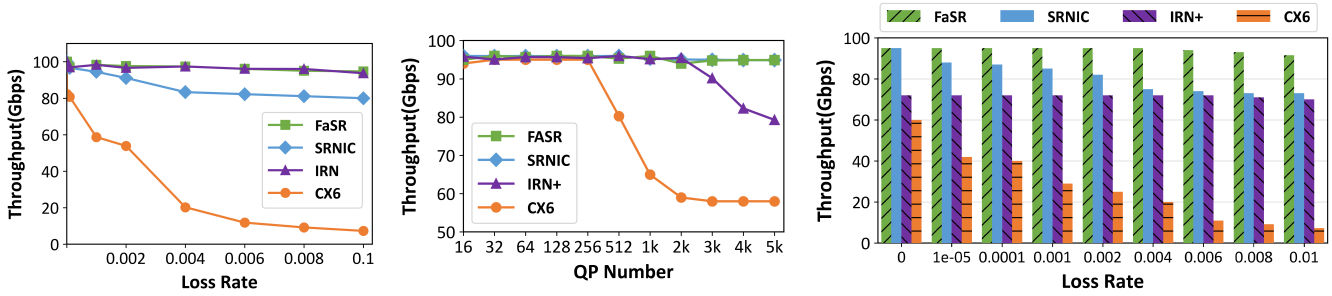
A. Testbed settings

Methods compared. Besides Commercial RNIC Mellanox CX6-DX, we compare FaSR to IRN, and SRNIC, which are the state of the art hardware-based SR and software-based SR.

We implement IRN and SRNIC based on the FaSR protocol stack on the same FPGA. To focus on the loss recovery part, we optimize IRN's connection scalability utilizing the architecture of SRNIC, that is, storing WQEs on host memory and dynamically fetching them using smart scheduling policies without storing them on the NIC board. The IRN's scalability is mainly limited by the QPC, where the SR states and bitmap consume the most. In the rest of this section, we call the optimized IRN as IRN+.

Memory overhead. 3.2MB SRAM space on the FPGA was allocated (1.2MB for MTT, 0.6MB for receiver buffer, 1.4MB for QPC [29]). For maintaining basic functionality, each QP on FaSR, IRN, and SRNIC on FPGA requires 256B QPC overhead. For SR, IRN+ adds 20B of SR state overhead per QP and 5 times the BDP size of the bitmaps. SRNIC adds 12B of state overhead per QP to avoid contention and a 1.2us delay in accessing the bitmaps, due to the interaction between the RNIC and the CPU for SR processing. FaSR adds 1B of SR state pointer per QP and 1600B of SR state share overhead (divided into 16-bit bitmap blocks).

Testbed topology. We construct a small cluster comprising four physical machines. Each machine has one socket of Intel Xeon E5-2650 CPU 64GB of memory and a 100Gbps FPGA board connected to two Tofino S9180-32F programmable switches. We deliberately generate packet loss on the switch



(a) Fault Tolerance under low concurrency. (b) Throughput with Various QPs, lossless. (c) Throughput under 5K QPs at Various Loss Rates.

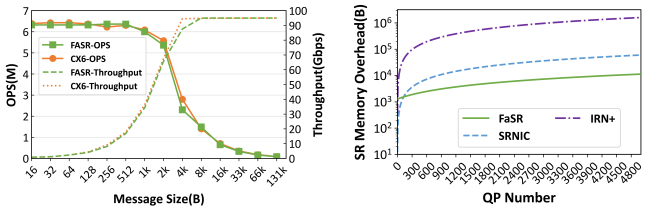
Figure 7. Throughput under different packet loss and concurrent QP.

through injecting a dedicated P4 data-plane program. The base RTT is about 6us, and the MTU is 1024 bytes. The standard perf-test benchmark [26] is used in all the experiments.

We turned off DCQCN in our experiments as we observed that when packet loss is detected, DCQCN will cut the sending speed in half regardless of whether congestion occurs, which will affect the accuracy of the experiment.

B. FaSR under low concurrency

We evaluate the throughput of a single QP continuously sending 8KB between two servers with varying packet loss rates. As shown in Fig. 7(a), the throughput of CX6 drops significantly when the loss rate exceeds 0.1% and only achieves about 10% throughput when the loss rate exceeds 1%. The throughput of SRNIC drops much slower than that of CX6, but it still experiences a 25% loss when the loss rate exceeds 1%. Thanks to the low-latency on-chip state access, IRN and FaSR maintain a stable throughput loss of under 7%, even at a 1% packet loss rate, which is 3x better (7% loss vs. 25% loss) than that of SRNIC, 12X better than CX6.



(a) Throughput and OPS performance. (b) SR Memory Overhead.

Figure 8. Throughput and Memory Overhead.

C. FaSR under high concurrency

1) Lossless: To evaluate connection scalability, we connected 1 server with 3 clients through a P4 switch, and the server issues RD/WR requests from clients in concurrent QPs from 16 to 5K. As shown in Fig. 7(b), SRNIC and FaSR maintain the highest throughput with little change when the number of QPs increases from 16 to 5K, as they keep the QPC of 5K QPs entirely in on-chip memory without cache misses. However, when the number of QPs exceeds 256, CX6 suffers a drop in throughput due to frequent cache misses. IRN experiences a more extensive QPC storage overhead than other solutions due to excessive bitmap overhead, when concurrent QPs exceed 2K, IRN suffers throughput performance slippage due to cache misses.

Although FaSR has a larger SRAM than CX6, the 1.6-fold increase in storage capacity (2MB vs. 3.3MB) has allowed us to achieve 19.5 times higher concurrency than CX6 (256 vs. 5K). FaSR has better connection scalability compared to CX6 and IRN, with the same concurrency performance as SRNIC.

2) Lossy: We tested FaSR and other methods at 5K concurrent QPs under different packet loss. As shown in Fig. 7(c), CX6 has extremely low throughput due to frequent cache misses and inefficient retransmissions. IRN has efficient retransmissions, but concurrency performance became a bottleneck, leading to a 30% throughput loss. SRNIC does not have a backup in concurrency performance but lost 25% throughput due to deficiency of software SR. FaSR performs well in concurrency and fault tolerance, with a throughput rate of above 92%. This is approximately 13 times higher than CX6 and 16% higher than SRNIC.

D. Deep Dive

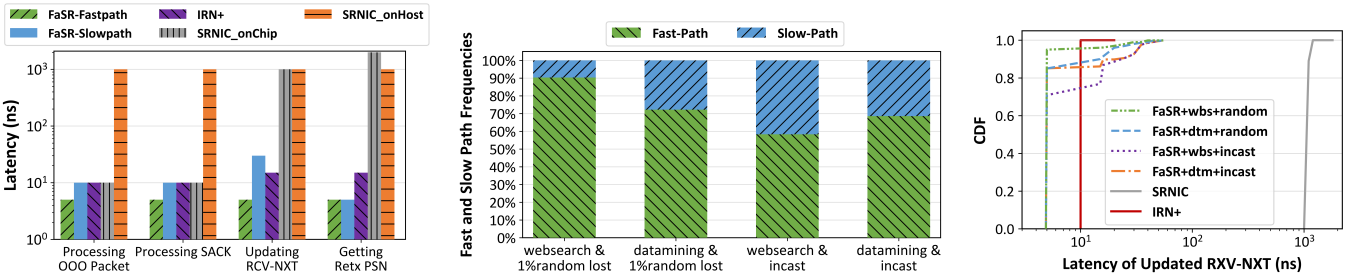
1) Throughput performance: We compared FaSR and CX6 in terms of throughput and operation per second (OPS) using a single connection, single core, and two directly connected NICs with a 1024-byte RoCE MTU.

As shown in Fig. 8(a), FaSR achieves a maximum message rate of 6.2 Mops, which is near to the 6.3 Mops achieved by CX6. The ops rate of FaSR is limited by the batch size of the WQE scheduler. In the implementation of FaSR, the scheduler batch size is set to 8 WQEs, with a PCIe delay of approximately 1.2us. Result theoretical upper limit for FaSR is 6.8 Mops, and the performance of FaSR aligns closely with this theoretical limit. Additionally, when the message size surpasses 4KB, both FaSR and CX6 almost reach their respective maximum sending rates.

2) Memory overheads and latency : We compare FaSR with SRNIC and IRN in terms of latency, and memory overhead with the same settings as above.

SR memory overhead. As shown in Fig. 8(b), due to the fixed overhead of the shared pool, the memory overhead of FaSR is highest when the concurrent QP is small. However, as the number of QP increases, the advantages of FaSR become apparent. When the number of QPs exceeds 145, FaSR has the smallest SR storage overhead, and this advantage becomes more pronounced as the number of QPs increases further.

SR action latency: SR delay is the time between the SR module receiving a new PSN or SACK and completing its



(a) Average Latency of SR Functions.

(b) Ratio of the fast and slow path.

(c) bitmap delay CDF.

Figure 9. SR latency under different lossy scenarios.

SR activities (*e.g.*, recording OoO packet, returning SACK, updating RCV-NXT, retransmitting the lost packet).

In actions like processing SACK or OoO packets, SRNIC, IRN, and FaSR may sustain nanosecond processing delays without waiting for the host's software bitmap result (Fig. 9(a)). Operations may be done in two clock cycles (10ns). Due to PCIe delay, SRNIC has much greater latency than IRN and FaSR when host response is needed, such as updating RCV-NXT or collecting retx PSN. In such cases, SRNIC operation latency rises by two orders of magnitude.

FaSR's SR logic is entirely implemented on the NIC, which enables it to achieve similar bitmap latency as IRN. Moreover, thanks to the fast path in FaSR, obtaining the retransmission PSN is even faster than with IRN. FaSR updates the RCV-NXT by querying the bitmap blocks one by one in the order of the chain table, and the latency is proportional to the number of queried bitmap blocks. However, the probability of this delay occurring is very low, and this delay can be buffered through the cache queue on the RNIC.

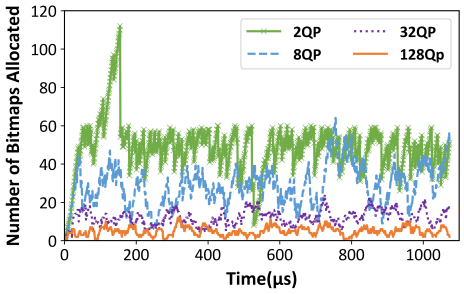


Figure 10. Bitmap Usage under 1% Loss Rate.

Latency of updating RCV-NXT. We examined FaSR, IRN, and SRNIC's RCV-NXT update delay under random or incast packet loss with websearch and datamining loads. Fig. 9(c) reveals that IRN and FaSR have much lower latency than SRNIC. The linked list structure prevents FaSR from querying all bitmaps at once. Due to the fast path, RCV-NXT updates often bypass the bitmap, making FaSR the lowest latency. Under different load and packet loss scenarios, the proportion of fast and slow paths is also different, but the median delay of FaSR is the lowest at only 5ns.

From the Fig. 9. It is clear that FaSR effectively reduces dependency on bitmaps via the fast path, resulting in an SR delay is highly likely to be 5ns. We estimate the SR delay expectation to be between 5.5 and 7ns, considering the predicted delay of 12ns for the slow path. Theoretically, the SR module's single pipeline packet processing per second (PPS) can reach a maximum of 181M, enabling it to support a bandwidth expansion of over 200Gbps.

3) Usage of FaSR SR state pool: We evaluate the shared pool's support for OoO records under packet loss and different concurrency scenarios through NS3 simulations.

Setup: we set a three-tier fat tree topology in NS3, with four backbone switches and 32 leaf switches, each connecting ten servers. The inter-switch link is 400Gbps, and the server's link is 100Gbps; we set the basic RTT to 40us, and MTU to 1KB. The bitmap pool size is 270B.

Result: FaSR's shared bitmap pool design can support all QP OoO records. As shown in Fig. 10, when the concurrency is 2QP and 8QP, retransmission packet loss occurs, but the bitmap pool covers it, preventing overflow. In addition, bitmap consumption reduces as the number of QPs increases. This is because the BDP of each QP lowers as the number of QPs increases, resulting in fewer OoO packets caused by packet loss. Furthermore, a smaller BDP increases the likelihood of a QP losing only one packet, implying that FaSR does not need to assign a bitmap to this QP, further lowering bitmap consumption.

VI. Related Works

MELO [22] is an early work designed for hardware-based SR using the same idea of shared bitmap. However, it has no system implementation and there is no associated low-latency design for line-rate processing. In addition, MELO requires the network with PFC, and is only intended to improve the robustness under random loss networks.

FLOR [20] implements SR at the message level within software layer. It splits large messages into smaller ones for transmission. However, when encountering packet loss, the trade off between retransmission efficiency and network utilization causes serious performance degradation.

Besides the RDMA loss recovery works [22], [23], [29] which we have discussed and compared in details, there are several works try to improve RDMA's scalability, by improving congestion control [5], [21], [23], [24], [30] or dynamically managing different connections [7], [19], [28]. And the latest fine-grained load balancer: ConWeave [27] can achieve high bandwidth utilization while ensuring that packets arrive at the

receiver in order under fine-grained rerouting. These works do not address the loss recovery, which are orthogonal and complimentary to FaSR.

VII. Conclusion

We propose FaSR, a fast and scalable RDMA selective retransmission. FaSR processes 200Gbps+ line-rate SR on the RNIC chip using a fast SR path and dual SR state management, using little memory even under high concurrency. In addition, with the Lost Recovery Model, FaSR has designed several technologies to access wire-speed shared structures for different QPs. The testbed evaluation shows that FaSR can maintain 92%+ throughput at a packet loss rate of 1% under more than 5K concurrent QPs, which is 16% and 12.6x higher compared to the latest RDMA SR solution and commodity RDMA NICs, respectively.

VIII. ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant U24B20150, 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 kh2401005, and in part by the Major special project of Shenzhen Science and Technology Plan under Grant KJZD20240903100813017.

References

[1] Fasr repository. https://github.com/hph101001/fasr.

[2] Martín Abadi and Paul Barham et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 265–283. USENIX Association, 2016.

[3] Wei Bai, Shanim Sainul Abdeen, and Ankit Agrawal et al. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 49–67, Boston, MA, April 2023. USENIX Association.

[4] Wei Cao, Yang Liu, and Zhushi Cheng et al. POLARDB meets computational storage: Efficiently support analytical workloads in cloud-native relational database. In *18th USENIX Conference on File and Storage Technologies, FAST 2020, Santa Clara, CA, USA, February 24-27, 2020*, pages 29–41. USENIX Association, 2020.

[5] Guo Chen, Yuanwei Lu, and Bojie Li et al. MP-RDMA: enabling RDMA with multi-path transport in datacenters. *IEEE/ACM Trans. Netw.*, 27(6):2308–2323, 2019.

[6] Guo Chen, Yuanwei Lu, and Yuan Meng et al. FUSO: fast multi-path loss recovery for data center networks. *IEEE/ACM Trans. Netw.*, 26(3):1376–1389, 2018.

[7] Diego Crupnicoff, Michael Kagan, and Ariel Shahar et al. Dynamically-connected transport service. In *US Patent*, pages 8,213,315, 2012.

[8] cx3. Mellanox ConnectX-3 Product Brief., 2018. https://lenovopress.lenovo.com/tips0897-mellanox-connectx-3.

[9] cx6. Mellanox ConnectX-6 Product Brief, 2020. https://nvdam.widen.net/s/qpszhmhpzt/networking-overal-dpu-datasheet-connectx-6-dx-smartnic-1991450.

[10] Mallesham Dasari, Kumara Kahatapitiya, Samir R. Das, Aruna Balasubramanian, and Dimitris Samaras. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 103–118, Renton, WA, April 2022. USENIX Association.

[11] Yixiao Gao, Qiang Li, and Lingbo Tang et al. When cloud storage meets RDMA. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, pages 519–533. USENIX Association, 2021.

[12] Chuanxiong Guo and Haitao Wu et al. RDMA over commodity ethernet at scale. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, pages 202–215. ACM, 2016.

[13] Chuanxiong Guo, Lihua Yuan, and Dong Xiang et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, pages 139–152. ACM, 2015.

[14] Shuihai Hu, Yibo Zhu, and Peng Cheng et al. Tagger: Practical PFC deadlock prevention in data center networks. *IEEE/ACM Trans. Netw.*, 27(2):889–902, 2019.

[15] Peihao Huang, Xin Zhang, and Zhigang Chen et al. LEFT: lightweight and fast packet reordering for RDMA. In *Proceedings of the 8th Asia-Pacific Workshop on Networking, APNet 2024, Sydney, Australia, August 3-4, 2024*, pages 67–73. ACM, 2024.

[16] Yimin Jiang, Yibo Zhu, and Chang Lan et al. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 463–479. USENIX Association, 2020.

[17] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Rdma in data centers: Looking back and looking forward. In . *https://conferences:sigcomm:org/events/ apnet2017/slides/cx:pdf*, pages 1–16. USENIX Association, 2017.

[18] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 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*, pages 1–16. USENIX Association, 2019.

[19] Pengfei Li, Yu Hua, and Pengfei Zuo et al. ROLEX: A scalable rdma-oriented learned key-value store for disaggregated memory systems. In *21st USENIX Conference on File and Storage Technologies, FAST 2023, Santa Clara, CA, USA, February 21-23, 2023*, pages 99–114. USENIX Association, 2023.

[20] Qiang Li, Yixiao Gao, and Xiaoliang Wang et al. Flor: An open high performance RDMA framework over heterogeneous rnics. In *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023, Boston, MA, USA, July 10-12, 2023*, pages 931–948. USENIX Association, 2023.

[21] Yuliang Li, Rui Miao, and Hongqiang Harry Liu et al. HPCC: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, pages 44–58. ACM, 2019.

[22] Yuanwei Lu, Guo Chen, and Zhenyuan Ruan et al. 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*, pages 22–28. ACM, 2017.

[23] Radhika Mittal and Alexander Shpiner et al. 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*, pages 313–326. ACM, 2018.

[24] Radhika Mittal, Vinh The Lam, and Nandita Dukkipati et al. 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*, pages 537–550. ACM, 2015.

[25] OpenAI. Gpt-4 technical report, 2023. https://arxiv.org/abs/2303.08774.

[26] Perftest. Ofed perftest. 2021.

[27] Cha Hwan Song and Xin Zhe Khooi et al. 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*, pages 816–831. ACM, 2023.

[28] Xizheng Wang, Guo Chen, and Xijin Yin et al. Star: Breaking the scalability limit for RDMA. In *29th IEEE International Conference on Network Protocols, ICNP 2021, Dallas, TX, USA, November 1-5, 2021*, pages 1–11. IEEE, 2021.

[29] Zilong Wang, Layong Luo, and Qingsong Ning et al. Srnic: A scalable architecture for rdma nics. In *23th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, April 20-23, 2023*, pages 519–533. USENIX Association, 2023.

[30] Yibo Zhu, Haggai Eran, and Daniel Firestone et al. Congestion control for large-scale RDMA deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, pages 523–536. ACM, 2015.