SRNIC: A Scalable Architecture for RDMA NICs

Zilong Wang, *Hong Kong University of Science and Technology;* Layong Luo and Qingsong Ning, *ByteDance;* Chaoliang Zeng, Wenxue Li, and Xinchen Wan, *Hong Kong University of Science and Technology;* Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, Tianhao Wang, Weicheng Ling, Kejia Huo, Pingbo An, Kui Ji, Shideng Zhang, Bin Xu, Ruiqing Feng, and Tao Ding, *ByteDance;* Kai Chen, *Hong Kong University of Science and Technology;* Chuanxiong Guo

https://www.usenix.org/conference/nsdi23/presentation/wang-zilong

This paper is included in the
Proceedings of the 20th USENIX Symposium on
Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

Open access to the Proceedings of the
20th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

SRNIC: A Scalable Architecture for RDMA NICs

Zilong Wang[1*] Layong Luo[2] Qingsong Ning[2] Chaoliang Zeng[1*] Wenxue Li[1] Xinchen Wan[1*]
Peng Xie[2] Tao Feng[2] Ke Cheng[2] Xiongfei Geng[2] Tianhao Wang[2] Weicheng Ling[2]
Kejia Huo[2] Pingbo An[2] Kui Ji[2] Shideng Zhang[2] Bin Xu[2] Ruiqing Feng[2] Tao Ding[2]
Kai Chen[1] Chuanxiong Guo[3]
[1]*Hong Kong University of Science and Technology* [2]*ByteDance* [3]*Unaffiliated*

Abstract

RDMA is expected to be highly scalable: to perform well in large-scale data center networks where packet losses are inevitable (*i.e.*, high network scalability), and to support a large number of performant connections per server (*i.e.*, high connection scalability). Commercial RoCEv2 NICs (RNICs) fall short on scalability as they rely on a lossless, limited-scale network fabric and support only a small number of performant connections. Recent work IRN improves the network scalability by relaxing the lossless network requirement, but the connection scalability issue remains unaddressed.

In this paper, we aim to address the connection scalability challenge, while maintaining high performance and low CPU overhead as commercial RNICs, and high network scalability as IRN, by designing SRNIC, a Scalable RDMA NIC architecture. Our key insight in SRNIC is that, on-chip data structures and their memory requirements in RNICs can be minimized with careful protocol and architecture co-designs to improve connection scalability. Guided by this insight, we analyze all data structures involved in an RDMA conceptual model, and remove them as many as possible with RDMA protocol header modifications and architectural innovations, including cache-free QP scheduler and memory-free selective repeat. We implement a fully functional SRNIC prototype using FPGA. Experiments show that, SRNIC achieves 10K performant connections on chip and outperforms commercial RNICs by 18x in terms of normalized connection scalability (*i.e.*, the number of performant connections per 1MB memory), while achieving 97 Gbps throughput and 3.3 µs latency with less than 5% CPU overhead, and maintaining high network scalability.

1 Introduction

Datacenter applications are increasingly driving the demands for high-speed networks, which are expected to provide high throughput, low latency, and low CPU overhead, with a large number of connections (*a.k.a.*, connection scalability), over a large-scale network (*a.k.a.*, network scalability). Specifically, bandwidth-intensive applications like distributed machine learning training [13, 23] and cloud storage [16, 18], require 100 Gbps and beyond network bandwidth between servers; online services like search [9, 15] and database [25, 29], demand low latency to minimize query response time; most applications desire a network stack with low CPU overhead to reserve as many CPU cores as possible for computations; cloud storage like Alibaba Pangu [18] requires a large number of performant connections per host to provide mesh communications between chunk servers and block servers; last but not the least, high-speed networks tend to be deployed at larger scale as their application footprints expand [19].

Remote Direct Memory Access (RDMA) is emerging as a popular high-speed networking technique, thanks to its high throughput, low latency and low CPU overhead provided by architectural innovations including kernel bypass and transport offload. With these advantages, RoCEv2 (RDMA over Converged Ethernet Version 2) is becoming the de-facto standard for high-speed networks in modern data centers [4, 42].

Despite high performance and low CPU overhead, commercial RoCEv2 NICs (RNICs) suffer from both network scalability and connection scalability issues. On one hand, the network scalability issue arises from PFC (Priority-based Flow Control) which is required by RDMA to implement a lossless network fabric. PFC brings issues such as head-of-line blocking, congestion spreading, occasional deadlocks, and PFC storms in large-scale clusters [18, 19, 21, 34, 42]. As a result, datacenter operators tend to restrict the PFC configurations within a small network scope (*e.g.*, a moderate cluster). On the other hand, the connection scalability issue is the phenomenon that RDMA performance drops dramatically when the number of connections (*a.k.a.*, queue pairs (QPs)) exceeds a certain small threshold (*e.g.*, 256) [24, 28, 39]. Although commercial RNICs are blackbox, the root cause of this performance collapse phenomenon is explained as cache misses due to context switch between connections [24].

[*] This work is done while Zilong Wang, Chaoliang Zeng, and Xinchen Wan are interns with ByteDance.

To improve network scalability of RNICs, existing work IRN [33] advocates lossy RDMA that eliminates PFC, by replacing go-back-N with more efficient selective repeat (SR). However, the introduction of SR is non-trivial: it adds some SR specific data structures and thus increases memory consumption. To reduce the on-chip memory overhead, IRN makes some RoCEv2 header extensions, but still requires 3-10% more memory than existing RNIC implementations. As a result, IRN achieves high network scalability but leaves the connection scalability issue unaddressed.

In this paper, we propose SRNIC, a Scalable RDMA NIC architecture to address the connection scalability issue, while preserving high performance and low CPU overhead inherited from transport offload as commercial RNICs, and maintaining high network scalability originated from lossy RDMA as IRN. The major insight of SRNIC is that, most on-chip data structures and their memory requirements in RNICs can be eliminated with careful protocol and architecture co-designs, and the connection scalability of RNICs could be, as a result, significantly improved. Guided by this insight, we examine the typical data flow in a lossy RDMA conceptual model (§3.1), analyze all the involved data structures, classify them into two categories: *common data structures* required by RDMA in general, and *selective repeat specific data structures* brought by lossy RDMA, and finally take customized optimization strategies to minimize these two types of data structures respectively to improve the connection scalability (§3.2).

In particular, the cache-free QP scheduler proposed in §4.3 optimizes common data structures for RDMA designs no matter whether the underlying network is lossy or lossless. The optimizations of RDMA header extensions and bitmap onloading introduced in §4.4 are for memory-free selective repeat, hence specific for lossy RDMA.

We have implemented a fully functional SRNIC prototype with FPGA (§5) and evaluated SRNIC's scalability and performance through the testbed and simulations. Experiments (§6) show that SRNIC achieves high connection scalability, while preserving high performance and low CPU overhead as commercial RNICs, and high network scalability as IRN. Specifically, SRNIC supports 10K[1] connections/QPs without performance degradation, which outperforms Mellanox RNIC CX-5 by 18x in terms of normalized connection scalability (*i.e.*, the number of performant connections per 1MB memory). Meanwhile, SRNIC achieves 97 Gbps line-rate throughput and 3.3 µs latency, with only 5% CPU overhead, which are comparable with Mellanox RNICs. In addition, SRNIC shows its high network scalability via high loss tolerance (3x higher goodput than Mellanox RNICs under 1% loss rate) and predictable performance in large-scale lossy networks.

As a summary, Figure 1 shows the design space of RDMA NICs and makes a comparative analysis between different solutions. Although all RDMA hardware solutions provide high throughput, low latency, and low CPU overhead via transport offload and kernel bypass, their scalability varies. Commercial RNICs suffer from both the network scalability issue caused by the troublesome PFC, and the connection scalability issue caused by unknown blackbox implementations. IRN revisits the network supports for RDMA, and eliminates the need of PFC by introducing selective repeat with 3-10% extra memory overhead. As a result, the network scalability is significantly improved, but the connection scalability is left unsolved. SRNIC leverages the lossy RDMA approach of IRN to improve network scalability, and further addresses the connection scalability issue with the design guiding principle: minimize the on-chip memory requirements of RNICs in a simple yet performant way. As a result, SRNIC achieves both high network scalability and connection scalability.

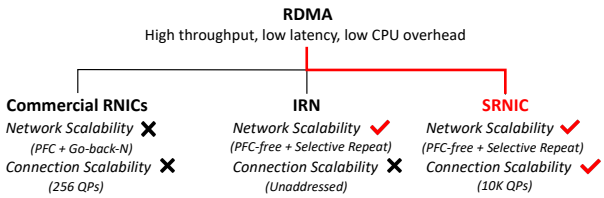


Figure 1: Design space of RDMA NICs.

This paper makes the following major contributions:

- We systematically study and quantify the memory requirements of RDMA NICs, by introducing an RDMA conceptual model (§3).
- We design SRNIC, a scalable and high-performance RDMA NIC architecture, that significantly improves the connection scalability, guided by an insight that the on-chip memory requirements in the conceptual model can be minimized with careful RDMA protocol modifications and architecture innovations, including cache-free QP scheduler and memory-free selective repeat (§4).
- We implement SRNIC using FPGA, with only 4.4 MB on-chip memory. The implementation achieves our design goals on scalability, performance, and CPU overhead (§5 and §6).

2 Background and Motivation

2.1 RDMA Overview

Unlike the traditional software transport TCP, RDMA is a hardware transport that implements the transport functionalities including congestion control and loss recovery entirely in NIC hardware, and provides kernel-bypass and zero-copy interfaces to the user applications. As a result, RDMA achieves high throughput, low latency, and low CPU overhead, compared with software transport TCP [42].

[1] Unless otherwise stated, K is 1024 in measuring the size of memory, data structures and messages, and 1000 in measuring the others.

RDMA was originally designed and simplified for lossless Infiniband [1]. To make RDMA work in Ethernet, RoCEv2 relies on PFC [22] to turn Ethernet into a lossless fabric. However, PFC brings management risks and network scalability challenges (*e.g.*, PFC storms and deadlocks) that affect the entire network's availability and also causes collateral damage to innocent flows due to head-of-line blocking [19, 42]. Besides, with PFC, the lossless network scale is also limited by the switch buffer size. Consequently, datacenters usually limit the scale of RDMA networks [18].

As the network scalability issue of RoCEv2 is mainly caused by PFC, IRN [33] takes the first step to rethink RDMA's network requirements, eliminates PFC and allows RDMA working well in lossy networks, by replacing the default lossy recovery mechanism go-back-N with more efficient selective repeat. However, it leaves the connection scalability challenge unsolved.

2.2 Connection Scalability Issue

Commercial RNICs face a well-known connection scalability issue [24, 27, 28, 39], *i.e.*, the RDMA performance drops significantly as the number of QPs increases beyond a small value (varies from 16 to 500 in different settings [28]). We demonstrate this issue using off-the-shelf commercial RNICs including Mellanox CX-5 and CX-6 [7, 8] with PFC enabled. As shown in Figure 2a, the aggregate throughput of Mellanox CX-6 drops 46% (from 97 to 52 Gbps) when the QP number increases from 128 to 16384, and there is no obvious improvement of connection scalability from CX-5 to CX-6.

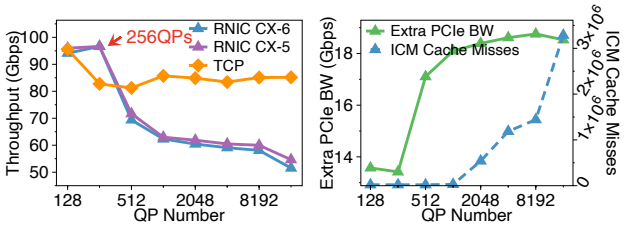
The root cause of RNIC's performance degradation is commonly explained as cache misses [24, 28, 38]. Commercial RNICs usually take a DRAM-free architecture, which does not have DRAM connected directly to the RNIC chip to reduce cost, power consumption, and area, but just has limited on-chip SRAM. As a result, RNICs can cache only a small number of QPs on chip, while storing the others in host memory. When the number of active QPs increases beyond the on-chip memory size, frequent cache misses and context switches between host memory and RNIC cause performance collapse. Our experiments in Figure 2b verify this in some sense. We observed significant extra PCIe bandwidth[2] and an increase in ICM cache miss[3] during the performance collapse. Both metrics reflect certain kinds of cache misses, causing extra PCIe traffic increase after 256 QPs.

Although on-chip SRAM is limited, it is abnormal in that the performance drops so early. Given the on-chip memory size and the QP Context (QPC) size for a QP, we can estimate the maximum number of performant QPs that could be supported without cache misses and performance collapse as:

$$max\_QPs = \frac{memory\_size}{sizeof(QPC)}. \quad (1)$$

[2]Extra PCIe throughput = PCIe throughput - network throughput.

[3]"ICM Cache Miss" is a counter provided by Mellanox Neohost tool [12].



(a) Aggregate throughput (b) ICM Cache misses and extra PCIe traffic

Figure 2: Connection scalability issue of current RNICs. Compared with TCP, the aggregate throughput of current RNICs collapses when the number of QPs exceeds 256.

Let's take Mellanox CX-5 as an example. Its on-chip memory size is ∼2 MB [24] and a QPC takes ∼375 B [24], so that the maximum number of performant QPs supported by CX-5 could be up to 5.6K (2 MB/375 B), which contradicts the fact shown in Figure 2a that CX-5 performance begins to collapse much earlier at 256 QPs. The contradiction implies that there is room to significantly improve the connection scalability.

Motivated by this contradiction, we systematically analyze the memory requirements of RNICs, and improve the connection scalability based on the insights derived from thorough memory analysis.

3 RNIC Memory Analysis

As commercial RNICs are blackbox, we are not able to use their micro-architectures as a reference. Instead, we leverage a lossy RDMA conceptual model with selective repeat to derive the involved data structures (§3.1). Then, we summarize and classify these data structures into two categories: *common data structures* required by RDMA in general, and *selective repeat specific data structures* brought by lossy RDMA, and discuss different optimization strategies to minimize them respectively to improve the connection scalability (§3.2).

3.1 RDMA Conceptual Model

Figure 3 shows an RDMA conceptual model, based on which, a typical RDMA data flow consists of the following steps:

1. Requester: the user posts a work queue element (WQE) into a send queue (SQ) to issue a SEND request. RNIC fetches the WQE from the SQ to a **WQE Cache**.

2. Requester: RNIC gets the virtual address of the data buffer by parsing the WQE, translates it into the physical address through a **Memory Translation Table** (MTT), and fetches data from the host data buffer using the physical address. RNIC then appends an appropriate RoCEv2 header onto the data and sends out the packet to the responder. The metadata of all outstanding requests is

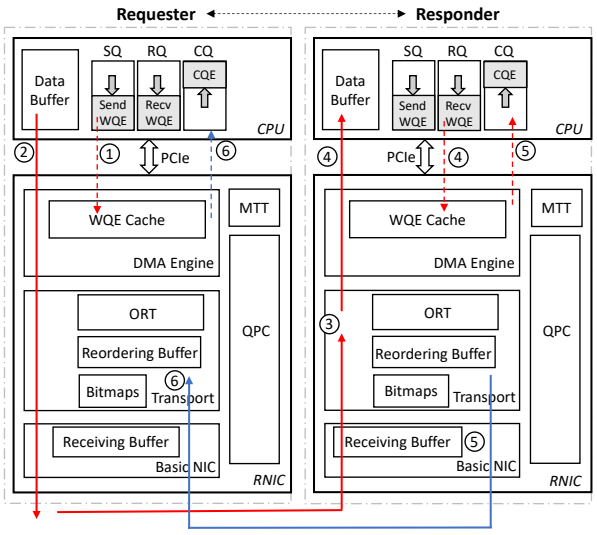


Figure 3: An RDMA conceptual model, and the RDMA data flow using a small SEND message as an example.

stored in an **Outstanding Request Table** (ORT) for fast retransmission in case of packet loss.

3. Responder: the incoming request is first queued in the **Receiving Buffer** and then gets verified. Out-of-order packets will be recorded in **Bitmaps** and reordered using the **Reordering Buffer**.

4. Responder: upon receiving a SEND packet, RNIC fetches a Receive WQE from a receive queue (RQ), queries MTT to get the physical address of the host data buffer, and DMAs the reordered data from the **Reordering Buffer** to the host data buffer.

5. Responder: RNIC replies an acknowledgment (ACK) packet to the requester, and notifies the user with a completion queue element (CQE) to indicate the Receive WQE is consumed.

6. Requester: RNIC receives the ACK, and generates a CQE to indicate the Send WQE is consumed.

Besides, RNIC leverages a **QPC** per QP to track QP/connection related contexts for all modules.

3.2 Data Structures

As concluded in Table 1, we classify the involved data structures into two categories: (1) *common data structures*, required by RDMA in general, and (2) *selective repeat specific data structures*, brought by lossy RDMA.

3.2.1 Common Data Structures

Common data structures are essential to RDMA in general, no matter whether the underlying network is lossy or lossless.

Receiving Buffer. The receiving buffer in the Basic NIC module is used to queue all incoming packets. Its major purpose is to absorb bursts caused by the temporal performance gap between the upstream Ethernet port and the whole downstream RNIC processing logic.

QPC. A QPC maintains for a QP all its contexts, including the DMA states (*e.g.*, the start and end addresses, read and write pointers of SQ & RQ), and connection states (*e.g.*, expected and next packet sequence numbers, window or rate for congestion control). The QPC size we allocate for each QP is 210 B, so the total size for 10K QPs is 2.0 MB.

MTT. RDMA uses virtual addresses in the packet while the PCIe system relies on physical addresses to perform DMA transactions. To perform address translation, RNIC maintains an MTT to map virtual pages of memory regions into physical pages. The size of MTT depends on the total size of memory regions and the page size, irrelevant to the number of connections. For example, considering the total memory region size of 4 GB, the page size of 4 KB, and an MTT entry size of 8 B, the MTT size is equal to $4\,GB/4\,KB * 8\,B = 8\,MB$.

WQE Cache. An SQ WQE cache could be used to cache the Send WQEs fetched from an SQ in host memory. Assuming each QP stores 8 WQEs (64 B*8) in a dedicated cache, 10K QPs consume 4.9 MB on-chip memory. Similarly, RNIC needs to fetch Receive WQEs from the RQ to process incoming SEND requests, and could allocate an RQ WQE cache to store the fetched Receive WQEs. The memory size of the RQ WQE cache is similar to that of the SQ WQE cache.

3.2.2 Selective Repeat Specific Data Structures

These data structures are all introduced by lossy RDMA using selective repeat as the loss recovery mechanism.

Bitmap. Bitmaps are used to track which packets are received or lost [31]. As mentioned in IRN [33], each QP requires five BDP (bandwidth-delay product)-sized bitmaps (500 slots for each bitmap to fit the BDP cap of a network with bandwidth 100 Gbps and RTT 40 µs [5]) and 10K QPs cost 3.0 MB memory in total.

Reordering Buffer. A reordering buffer is used to rearrange the out-of-order packets and ensure in-order delivery to the data buffer in host memory. The reordering buffer is required in a lossy RNIC implementation with the standard RoCEv2 header. As RoCEv2 is designed for the lossless network, its header lacks the necessary information to support out-of-order packet reception without extra reordering buffers.

One option is to allocate a separate reordering buffer for each QP. Each QP requires a BDP-sized (0.5 MB) reordering buffer, so it takes 4.9 GB memory to support 10K QPs. Another option is to maintain a shared reordering buffer for all QPs [31]. However, it does not scale. When multiple QPs experience out-of-order packets, it may soon run out of the shared buffer with limited on-chip SRAM. Hence, we choose the separate reordering buffer option in the analysis.

Category	Data structures	Typical sizes	Optimization ideas	Sizes after optimization
Common	Receiving Buffer	0.6 MB	None	0.6 MB
	QPC	2.0 MB	None	2.0 MB
	MTT	8 MB	Cache (§4.5)	1.2 MB
	WQE Cache	9.8 MB	Cache-free QP scheduler (§4.3)	0
SR Specific	Bitmap	3.0 MB	Bitmap onloading (§4.4.2)	0
	Reordering Buffer	4.9 GB	Header extensions (§4.4.1)	0
	Outstanding Request Table	114.4 MB	Header extensions (§4.4.1)	0

Table 1: Data structures in the RDMA conceptual model. The first three columns show the typical data structures and their memory requirements with 10K QPs. The last two columns summarize our ideas to minimize the on-chip memory requirements of these data structures, and show the memory size after optimization.

Outstanding Request Table. Outstanding request table is used to maintain the mapping between outstanding request packets and their metadata, which are used to quickly locate and retransmit the lost packets. These metadata include (1) packet sequence number (PSN), used to track packet sequences, (2) message sequence number (MSN), used to track message sequences and to locate the WQE associated with that message quickly, and (3) packet offset (PSN_OFFSET), used to locate the data offset inside the corresponding data buffer. With these fields, the outstanding request table size for each QP is 11.7 KB (given the entry size 24 B, entry number 500 sized to BDP), and 10K QPs consume 114.4 MB in total.

In summary, all the data structures derived from the RDMA conceptual model could be classified into two categories: *common data structures* required by RDMA in general, and *selective repeat specific data structures* brought by lossy RDMA. Table 1 summarizes the memory requirements of these data structures in the third column. Both categories require significant memory sizes, and thus need to be optimized to improve connection scalability.

To this end, we make different optimization strategies to minimize these two types of data structures respectively. In particular, all the common data structures required by RDMA should be optimized in a generic way, with architectural innovations that are not specific to lossless or lossy RDMA. The cache-free QP scheduler proposed in §4.3 falls into this strategy. On the other hand, all the selective repeat specific data structures brought by lossy RDMA, could be optimized based on the lossy network assumption. The header extensions and bitmap onloading approaches in the memory-free selective repeat architecture in §4.4 follow this strategy.

4 SRNIC Design

4.1 Design Goal and Guiding Principles

In the design space of RDMA NICs, Mellanox RNICs represent the state-of-the-art in terms of high performance and low CPU overhead, and IRN is the state-of-the-art in network scalability. The design goal of SRNIC is to maximize the connection scalability, while preserving high performance and low CPU overhead as Mellanox RNICs, and maintaining high network scalability as IRN.

To achieve this goal, we follow three design guiding principles:(1) keep as many RDMA functionalities as possible in hardware to achieve high performance and low CPU overhead; (2) handle packet loss as efficient as possible to allow discarding PFC and thus to support large-scale lossy networks; and (3) reduce the on-chip memory requirements as much as possible to support a large number of performant QPs with a limited amount of memory.

4.2 Architecture Overview

Guided by the above principles, we design a scalable RDMA NIC architecture SRNIC, as shown in Figure 4.

The server CPU allocates and manages QPs in the RNIC driver, and runs applications in user space over these QPs. Besides, a software retransmission module resides in user space to maintain the memory-consuming retransmission states collected by hardware and assist packet loss processing (§4.4). A pair of control queues (CtrlQs) is used as the communication channel between the software retransmission module and RNIC hardware.

RNIC hardware consists of three layers: DMA Engine, Transport, and Basic NIC. The *DMA Engine* layer leverages a *QP scheduler* to schedule tens of thousands of QPs from host memory, decides which QP to send data next, and then fetches WQEs and data from that SQ via *data mover*. The *Transport* layer realizes most of RDMA transport functionalities (except for the *software retransmission* in CPU), including a *congestion control* module that implements a hardware-friendly DCTCP [14], and a *hardware retransmission* module that implements the hardware part of selective repeat. The *Basic NIC* layer implements the primary functions of the Ethernet NIC, responsible for sending and receiving RoCEv2 packets via the *100GE MAC*. In addition to these three layers, there are two major data structures: QPC, which maintains all QP-related contexts, and MTT, which stores the mapping between virtual and physical addresses.

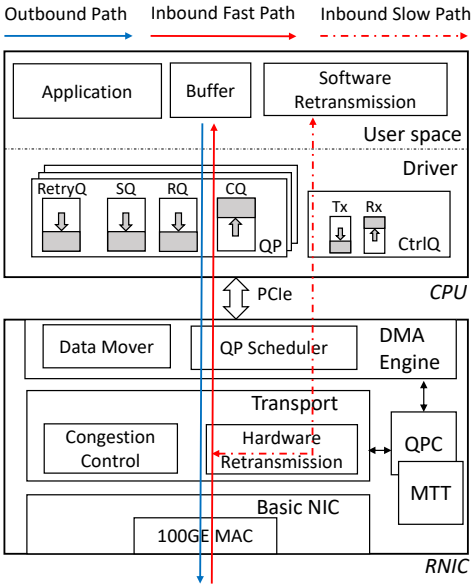


Figure 4: SRNIC architecture.

In order to balance performance and scalability, the data path of SRNIC is divided into a fast path and a slow path (§4.4), which handle sequential and out-of-order (OOO) packets, respectively. The fast path wholly implemented in RNIC processes the majority of traffic consisting of sequential packets, and thus provides hardware-level high performance with low CPU overhead for most packets. The slow path implements software retransmission, processes very little traffic consisting of OOO packets, and onloads bitmaps to host memory for connection scalability.

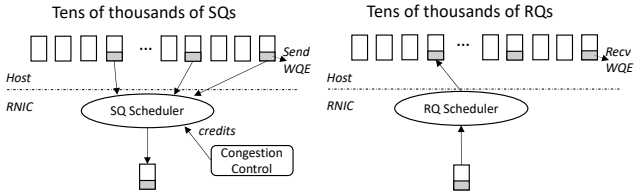
The overhead of the data path separation is very low for two reasons. First, the average packet loss rate in data centers is low (less than 0.01% [20, 41, 43]), and the resulting OOO packets form a very small fraction of traffic. Second, SRNIC only transmits loss events (*i.e.*, metadata of the OOO packets) over PCIe, further reducing the PCIe overhead. For example, the extra PCIe overhead is only 2.46% even with 1% loss rate.

Based on the above architecture, we further make two critical design optimizations: cache-free QP scheduler (§4.3) and memory-free selective repeat (§4.4) to optimize RDMA common data structures and lossy RDMA specific data structures, respectively, in order to address the scalability issues while preserving high performance.

4.3 Cache-free QP Scheduler

4.3.1 SQ Scheduler

An SQ is either *active* when it contains WQEs or *inactive* otherwise. The SQ scheduler (as modeled in Figure 5a) chooses one active SQ each time from tens of thousands of SQs in host memory to send messages next. The design challenges



(a) SQ scheduler model. (b) RQ scheduler model.

Figure 5: The QP scheduler models.

of the SQ scheduler are as follows:

- **Challenge #1**: Active SQs cannot be scheduled blindly, as they are also subject to congestion control, as shown in Figure 5a. Once an SQ is scheduled, if it is not allowed to send messages due to the lack of credits granted by congestion control, the scheduling does not take effect but just wastes time and degrades performance.

- **Challenge #2**: The PCIe round-trip latency between RNIC and host memory is high (around 1 µs in FPGA based RNIC), and it takes at least two PCIe transactions (one WQE fetch and one message fetch), to execute one scheduling decision. Without careful design, the high latency between scheduling iterations will significantly degrade the performance.

- **Challenge #3**: There are tens of thousands of SQs in host memory but very limited on-chip memory within RNIC. It is prohibitive to have separate WQE caches for different SQs in the RNIC.

To address these challenges, SQs should be scheduled when they are both active and have credits (to address Challenge #1), with appropriate batch transactions to hide PCIe latency (to address Challenge #2), and in a WQE-cache-free way (to address Challenge #3).

Guided by these principles, we propose a cache-free SQ scheduler (as shown in Figure 6) that can do fast scheduling among tens of thousands QPs with minimal on-chip memory requirements. It consists of three major components:
Event Mux (EMUX): The EMUX module handles all scheduling related events, including (1) SQ doorbell[4] from the host to indicate which SQ has new WQEs and messages to send; (2) credit update from the *congestion control* module to indicate window or rate adjustment for a connection/SQ; and (3) dequeue event from the *schedule queue* to indicate an SQ is scheduled.

Upon receiving an event, EMUX changes the scheduling states in QPC. There are three scheduling states: an *active* state indicating the SQ has WQEs; a *credit* value indicating

[4]Doorbell is the mechanism for the driver to notify RNIC that a SEND WQE has been posted into an SQ [26]. It is usually implemented by updating the write pointer of the SQ into an RNIC register.

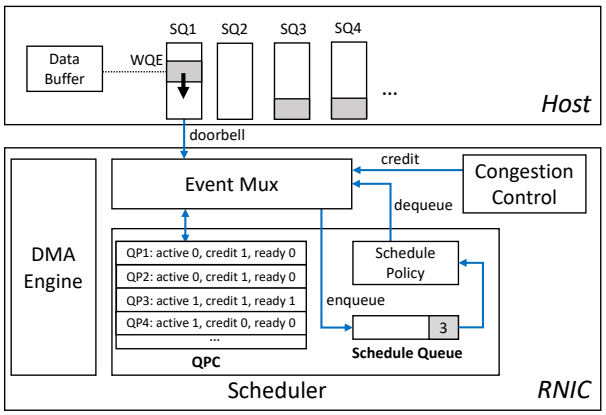


Figure 6: The cache-free SQ scheduler.

the bytes of messages allowed to send, and a *ready* state indicating the SQ is in the *schedule queue* and ready for scheduling. An SQ is ready for scheduling only when it is both active and has available credits, which addresses **Challenge #1**.

Scheduler: The scheduler leverages a *schedule queue* to maintain a list of SQs ready for scheduling. The scheduler implements a round-robin strategy in the *schedule policy* module, by popping a single ready SQ from the head of the *schedule queue* each time, and fetching from that SQ a given amount of WQEs and messages. After this scheduling iteration, if the SQ is still ready for scheduling, it will be pushed back into the *schedule queue* by the EMUX. Other scheduling strategies (*e.g.*, weighted round-robin and strict priority) can be implemented by modifying the *schedule policy* module.

DMA Engine: When an SQ is being scheduled, the DMA engine fetches from that SQ up to n WQEs and $min(burst\_size, credit)$ bytes of messages to address **Challenge #2**. After a scheduling iteration, there could be unused WQEs left in RNIC, if the total message size associated with the n WQEs is over $min(burst\_size, credit)$ bytes. Unused WQEs are dropped instead of being cached in RNIC, and they will be fetched again next time when its SQ is scheduled. This *fetch-and-drop* strategy enables us to achieve cache-free scheduling to address **Challenge #3**.

There are two critical parameters (n and $burst\_size$) to balance tradeoffs. n is the maximum number of WQEs, and $burst\_size$ is the maximum bytes of messages allowed to fetch in each scheduling iteration. n reflects the tradeoff between PCIe bandwidth usage and PCIe latency hiding. A smaller n would lead to less PCIe bandwidth waste in the *fetch-and-drop* strategy, but be harder to hide the PCIe latency or saturate the PCIe bandwidth with small messages, while a larger n would perform inversely. In SRNIC, n is set to 8 to balance the PCIe bandwidth utilization and latency hiding. With this setting, the maximum message rate of a single QP is 8 million requests per second (Mrps) (*i.e.*, 8 messages per 1 µs). As for $burst\_size$, it reflects the tradeoff between PCIe bandwidth utilization and scheduling granularity. A smaller $burst\_size$ would enable finer scheduling granularity and hence less HoL, but be harder to saturate PCIe bandwidth, while a larger $burst\_size$ would perform inversely. Based on this analysis, we set $burst\_size$ to the PCIe BDP, *i.e.*, 16 KB, to balance performance and scheduling granularity.

In summary, the SQ scheduler adopts a cache-free architecture to do fast scheduling among a large number of SQs with minimal on-chip memory. Specifically, the width of the schedule queue is 2 bytes, *i.e.*, the QPN (QP Number) size, and a schedule queue of 19.5 KB can support 10K SQs.

4.3.2 RQ Scheduler

The RQ scheduler is modeled as shown in Figure 5b. Upon receiving a packet, RNIC gets its QPN by parsing the packet header, fetches a Receive WQE from the RQ indicated by that QPN, and places the packet payload into the data buffer associated with that Receive WQE.

This process seems straightforward, but there is one design decision affecting connection scalability: *do we prefetch and cache Receive WQE in RNIC before the packet arrives?*

If Receive WQEs are prefetched and cached, the incoming packet could hit the WQE cache, reducing the latency by one PCIe round-trip time (*i.e.*, around 1 µs). However, it is hard to predict from which RQ to prefetch Receive WQEs before packets arrive, and thus the cache hit ratio largely depends on the traffic pattern and the cache size. Therefore, we decide to take the cache-free approach without prefetching or caching Receive WQEs, thus improving the connection scalability. Given that the typical RDMA network latency for small messages is tens of microseconds in data centers(*e.g.*, for 1KB messages, RDMA P50 and P99 latency is 24us and 40us, respectively [5]), the increased 1 µs latency is generally negligible. For latency-sensitive scenarios where 1 µs matters, like in rack-scale deployments, a shared Receive WQE cache can be brought back to optimize the latency.

4.4 Memory-free Selective Repeat

The introduction of selective repeat into RNICs increases the challenge to achieve high connection scalability. As analyzed in §3.2.2, the extra data structures brought by selective repeat include outstanding request tables, reordering buffers, and bitmaps, whose memory requirements in total exceed the typical on-chip SRAM sizes of RNICs.

To minimize the memory requirements introduced by selective repeat, SRNIC eliminates the need for outstanding request tables and reordering buffers via RDMA protocol header extensions (§4.4.1), and onloads bitmaps into host memory without sacrificing performance via careful software-hardware co-designs (§4.4.2).

4.4.1 Header Extensions

As described in §3.2, the *outstanding request table* is used to maintain for each QP the mapping between outstanding request packets and their metadata including PSN, MSN, and PSN_OFFSET for fast selective retransmission. We eliminate the need for this data structure, by carrying these per-packet metadata on packet headers, instead of storing them in the on-chip memory. Specifically, we let all outstanding request packets carry these metadata on their headers, and let their response packets echo the same metadata back. In this way, the requester can locate the WQE and its message quickly with metadata in the response packet header.

The *reordering buffer* is used by each QP to rearrange the OOO packets and ensures in-order delivery to the data buffer of user applications. To get rid of the per-QP reordering buffer, our approach is *in-place reordering*, *i.e.*, leveraging the user data buffer pinned in host memory as the reordering buffer. To achieve this, all incoming packets should be placed directly into the user buffer at correct addresses. We make the following header extensions so that RNIC can derive the address for each packet by parsing its header: (1) all SEND packets carry send message sequence number (SSN) and the aforementioned PSN_OFFSET, which can be used by the RNIC responder to locate the corresponding receive WQE and the offset in its associated receive buffer. (2) all WRITE packets carry their target remote addresses [33].

As to RDMA READ, we add acknowledgements to READ requests and responses respectively to add self-clocking for RDMA READ, and schedule RDMA READ at the responder side similar to RDMA WRITE. By doing so, we can apply similar header extensions of SEND and WRITE for READ request and response packets, and more importantly, we can apply window-based congestion control for RDMA.

With these modifications, both sequential and out-of-order packets can be placed directly into the user buffer at the correct address, thus achieving *in-place reordering* and eliminating per-QP reordering buffer in the on-chip memory.

The aforementioned extensions add 8 to 20 bytes of headers to packets. In particular, the header is increased from 58 to 66 bytes for SEND and from 58 to 78 bytes for WRITE, which will decrease the application goodput by 0.7% and 1.8%, respectively, given 1024 byte RoCE MTU.

4.4.2 Bitmap Onloading

As mentioned in §3.2.2, each QP requires five BDP-sized bitmaps, and 10K QPs need 3.0 MB memory to store bitmaps, which alone may exceed the RNIC on-chip memory size (*e.g.*, 2 MB in Mellanox RNIC [24]), thus increasing the challenge to achieve high connection scalability.

We observe that, when there is no packet loss, packets from the same QP are sent and received in order, and an expected PSN (ePSN) in the responder and a last acknowledged PSN

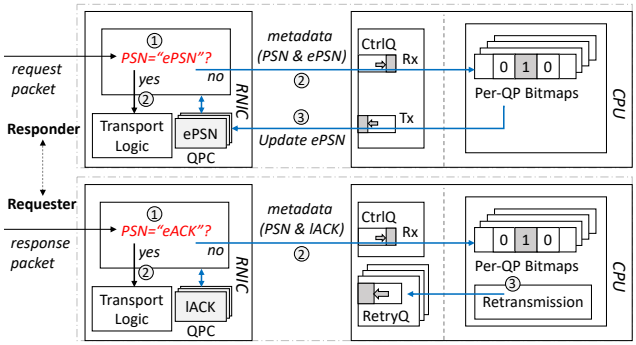


Figure 7: Selective repeat with bitmap onloading.

(lACK) in the requester are enough to track the sequential reception of request and response packets, respectively, without the need of bitmaps; when there is packet loss, OOO packets appear, and bitmaps are only required to track OOO packets.

Based on the above observation, for each QP we maintain an ePSN and a lACK in QPC to process sequential packets in hardware, and onload all bitmaps into host memory to track OOO packets. Assume packet loss rate is low and sequential packets are the majority, most traffic is handled by hardware directly, and little traffic containing the OOO packets is handled by software with the memory-consuming bitmaps in host memory. In this way, we achieve a balance between high performance and high connection scalability.

Figure 7 shows the software-hardware co-designed selective repeat architecture with bitmap onloading. On the responder side, the PSN of an inbound request packet is compared against the ePSN (①). If they match (②), it is a sequential packet and will be handled in the RNIC; otherwise (②), it is an OOO packet and the responder enters into the loss recovery state. In this state, the metadata (PSN and ePSN) of all incoming OOO packets is sent to software, which then fills the bitmaps in host memory to track received packets. After lost packets are received and bitmaps are filled accordingly, a new ePSN is updated (③), and the RNIC exits from the loss recovery state. On the requester side, the PSN of an inbound response packet is compared against an eACK (*i.e.*, a coalesced ACK greater than the lACK) (①). If they match (②), the lACK is updated in hardware; otherwise (*e.g.*, upon receiving NACK or SACK) (②), the requester enters into the loss recovery state. In this state, the metadata of all incoming OOO response packets including PSN and lACK is sent to the software retransmission module, which then manipulates the bitmaps in host memory to track which packets are received by the responder, and makes retransmission decision accordingly. The retransmitted requests are submitted through a Retry Queue (RetryQ) associated with each QP (③). After all retransmitted packets are successfully delivered (indicated by ACKs), the requester exits from the loss recovery state. Another option is to keep bitmaps only in the responder and make the requester stateless. Then, the responder should notify the requester exactly which packets to be retransmitted.

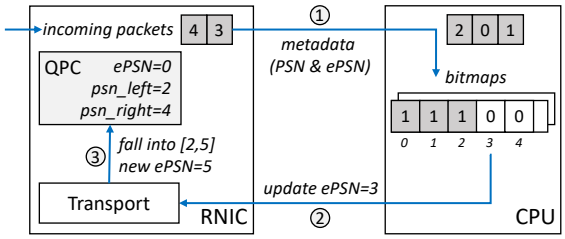


Figure 8: Fast exit from the loss recovery state.

A race condition may arise in the responder when exiting from the loss recovery state. Specifically, when the software updates the new ePSN, there might be inflight metadata of OOO packets with newer PSN between RNIC and CPU. In this case, the updated ePSN is not the latest, and thus the exit fails. To address the race condition problem while preserving high performance, RNIC records the range of the most advanced sequential packets (via $[psn\_left, psn\_right]$) after it enters the loss recovery state. A QP can exit from the loss recovery state if the updated ePSN falls into $[psn\_left, psn\_right + 1]$ range, and the ePSN in QPC will be updated to $psn\_right + 1$, as illustrated in Figure 8.

4.5 Other Design Considerations

With the cache-free QP scheduler and memory-free selective repeat, all data structures shown in Table 1 are eliminated, except for the receiving buffer, QPC, and MTT.

Receiving Buffer is a shared packet buffer among all QPs and its size is small, so it is not optimized in this paper.

QPC is essential to maintain the per-QP states, and is involved in per-packet processing. To support a large number of performant QPs, we have to store their QPCs entirely in on-chip memory. Therefore, this part is not eliminated, and we preserve as much on-chip memory as possible for QPC to maximize the number of performant QPs.

MTT is memory-consuming as analyzed in §3.2 (*e.g.*, 4 GB memory region requires 8 MB MTT size). Therefore, MTT is maintained in the host memory, and an MTT cache is implemented inside the RNIC by leveraging traffic locality. The cache size does not increase with the number of QPs, and its performance is highly related to traffic patterns. In addition, adopting hugepages (*e.g.*, 2MB/1GB) is a classical optimization to reduce the memory size of address translation tables [24, 40], but requires modification to the applications.

4.6 Design Summary

The last two columns of Table 1 summarize our ideas to minimize the RDMA related data structures, and show the memory requirements after optimizations. Specifically, we eliminate the WQE cache through a cache-free QP scheduler, eliminate all SR-related data structures in on-chip memory through SR-friendly header extensions and bitmap onloading, and minimize the on-chip memory requirements of MTT with a cache, while keeping the large MTT table in host memory.

Resource Usage			
LUT	Register	BRAM	URAM
101102	140816	621	48

Memory Breakdown (MB)				
QPC	MTT	Receiving Buffer	SQ Scheduler	Total
2.3	1.2	0.6	0.3	4.4

Table 2: Resource usage of the SRNIC prototype.

5 Implementation

We build a fully functional prototype of SRNIC using a Xilinx FPGA board with a PCIe Gen3 x16 interface and a 100 Gbps Ethernet port, running at a clock frequency of 300 MHz.

Congestion Control. Since SRNIC introduces ACK based self-clocking for RDMA READ, we therefore can use window-based congestion control for RDMA. Window-based approach in general is more friendly for hardware implementation than rate-based congestion control due to its self-clocking mechanism. More specifically, window-based design is event-driven: congestion window update events are triggered by inbound acknowledgement packets, and window based congestion control for each flow is applied at QP scheduling events. These events are naturally serialized and can be processed one by one. On the other hand, rate-based congestion control is timer-driven. It is challenging to support a large number of timer-based rate limiters in parallel for many concurrent flows. In SRNIC, we use DCTCP.

Memory Consumption. We realize 10K QPs in SRNIC and the resource consumption is broken down in Table 2. SRNIC consumes 4.4 MB on-chip SRAM in total. The QPC table, whose size increases linearly with the QP number, occupies 2.3 MB[5] for 10K QPs. The remaining memories are used by QP-irrelevant data structures, including MTT cache, receiving buffer, and SQ scheduler, which consume constant memories when the QP number increases.

Per Table 2, the precious on-chip SRAM of SRNIC is mainly partitioned between the two most memory-consuming data structures: the QPC table and the MTT cache. A larger QPC table would support more performant QPs, while a larger MTT cache could provide a higher cache hit rate during address translation thus better performance. The best on-chip memory partition strategy between the QPC table and the MTT cache highly depends on scenarios, and it's an interesting problem to explore in the future.

[5] This is slightly larger than 2 MB calculated in Table 1 due to memory alignment overhead, *e.g.*, each memory depth should be a power of 2 in FPGA implementation.

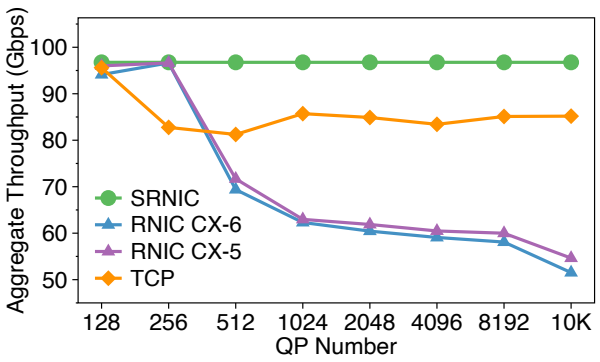


Figure 9: Connection scalability. SRNIC maintains constant high throughput as the number of QPs increases, while the performance of commercial RNICs (Mellanox CX-5 & 6) drops dramatically when the QP number exceeds 256.

6 Evaluation

We evaluate SRNIC using both testbed experiments and large-scale ns-3 simulations [10], and compare it with Mellanox RNICs, IRN, and TCP. Our results reveal that:

- SRNIC achieves high connection scalability: it supports 10K performant QPs, outperforming Mellanox RNIC CX-5 by 18x in terms of normalized connection scalability.
- SRNIC achieves high throughput (97 Gbps), low latency (3.3 µs), and low (5%) CPU overhead.
- SRNIC achieves high network scalability: it is loss-tolerant (up to 75 Gbps goodput under 1% loss rate) and maintains predictable performance over large-scale lossy networks.

6.1 Connection Scalability

We compare SRNIC with Mellanox RNIC CX-5, CX-6, and TCP in terms of connection scalability. The settings of the testbed experiments are as follow. We connect two RNICs directly and launch 16 threads on each side, with each thread executing 512 B send operations. We set the RoCE MTU to 1024 bytes, and use the standard *perftest* benchmarks [11] in all experiments. With the above settings, we measure the aggregate throughput of these solutions while increasing the number of QPs from 128 to 10K, as shown in Figure 9.

SRNIC preserves the highest aggregate throughput almost unchanged at around 97 Gbps when the QP number increases from 128 to 10K. This is expected, as SRNIC keeps the QPC of 10K QPs entirely in the on-chip memory while eliminating or minimizing all other data structures.

TCP also preserves relatively high performance (from 81 to 96 Gbps), as it maintains the contexts of 10K connections in the large host memory, demonstrating high connection scalability but lower and unpredictable performance.

In contrast, the aggregate throughput of Mellanox RNICs CX-5 and CX-6 drops dramatically when the QP number exceeds 256 due to frequent cache misses, as explained in §2.2.

In summary, SRNIC provides much higher connection scalability than commercial RNICs. Specifically, SRNIC realizes 10K QPs with 4.4 MB memory, while Mellanox CX-5 supports 256 QPs with 2 MB memory. To make a fair comparison, we define *normalized connection scalability* as the number of performant connections per 1 MB on-chip memory. SRNIC outperforms Mellanox CX-5[6] by 18x (10 K QPs/4.4 MB vs. 256 QPs/2 MB) in terms of normalized connection scalability.

6.2 Performance and CPU Overhead

We compare SRNIC with CX-6[7] and TCP in terms of throughput, latency, and CPU overhead using a single connection, with the same settings as above (*i.e.*, 1024-byte RoCE MTU, two NICs are connected directly).

Throughput. The throughput comparison is shown in Figure 10a. When the message size exceeds 4 KB, SRNIC and CX-6 both achieve line-rate throughput (97 Gbps), whereas TCP can only achieve up to 37 Gbps since the single CPU core becomes the bottleneck. In our experiments, the maximum message rate that SRNIC can achieve is 6.6 Mrps, comparable to that of the CX-6 (6.3 Mrps). This confirms that RNIC can achieve a high message rate without WQE cache. As mentioned in §4.3.1, the message rate of SRNIC depends on the batch size of the SQ scheduler. In our implementation, the SQ scheduler can request at most 8 WQEs at a time and the average PCIe RTT we measured is 1.1 µs, therefore our result is close to the upper bound of 7.2 Mrps.

Latency. We measure the latency for transmitting 64 B small messages. As Figure 10b shows, the latency of SRNIC is about 3.3 µs, slightly higher than that of CX-6 (1.16 µs). We believe this gap comes from the extra 1 µs added by the cache-free QP scheduler and the clock frequency difference between FPGA (300MHz) and ASIC (GHz) implementations. The latency would be decreased if SRNIC adopts the shared Receive WQE cache or is implemented in ASIC. In contrast, TCP has the highest latency of 24 µs, indicating that bypassing kernel and offloading transport in RDMA is vital for significant latency reduction.

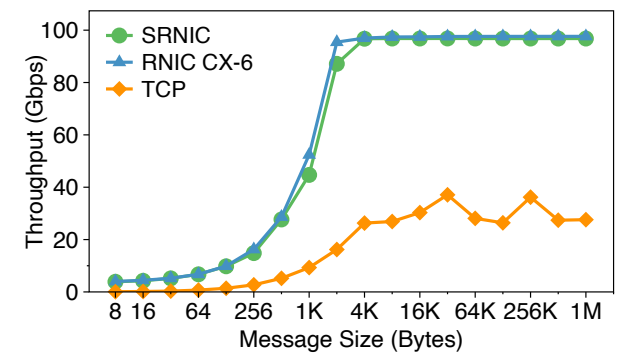
CPU overhead. As shown in Figure 10c, the CPU overhead of SRNIC and CX-6 both maintains at a low level (< 5%) thanks to transport offload and kernel bypass. TCP consumes much more CPU cycles at both the client and server sides (around 100% CPU utilization, not shown in the figure).

6.3 Network Scalability

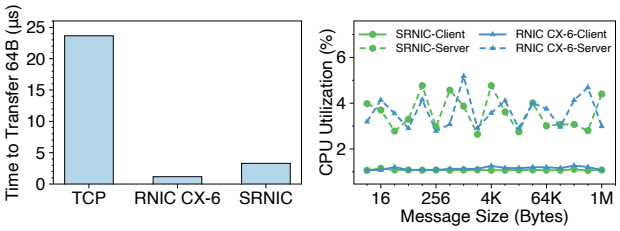
Finally, we evaluate the network scalability of SRNIC. We show the efficiency of loss recovery in SRNIC with testbed

[6] We know the on-chip memory size (*i.e.*, 2 MB) of CX5 [24] but not CX6, so we only compare with CX-5 in terms of normalized connection scalability.

[7] CX-5 and CX-6 behave similarly, so we only show CX-6 thereafter.



(a) Throughput



(b) Latency (c) CPU overhead

Figure 10: Performance and CPU overhead. SRNIC achieves high throughput, low latency, and low CPU overhead, similar to CX-6.

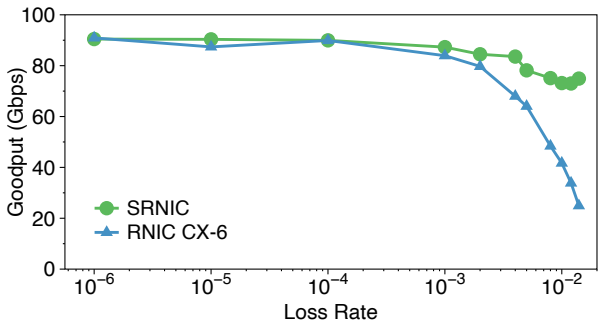
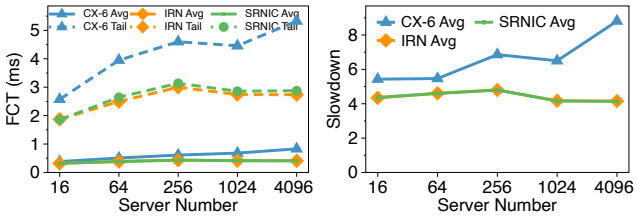


Figure 11: Loss tolerance. SRNIC achieves higher goodput than CX-6 when loss rate increases, as the number of retransmitted packets with selective repeat is much fewer than that with go-back-N.



(a) Average and tail FCT (b) Average slowdown

Figure 12: Performance at different network scales.

experiments, and the performance of SRNIC over large-scale lossy networks via simulations.

Loss tolerance. We compare the goodput of SRNIC with CX-6 at different packet loss rates, which are emulated by placing an FPGA between two RNICs and letting the FPGA randomly drop packets at given rates. We use *perftest* to generate 4 KB messages continuously. We disable congestion control here to exclude the influence of congestion control on loss tolerance, and only compare the loss recovery efficiency between selective repeat in SRNIC and go-back-N in CX-6.

Figure 11 compares SRNIC with CX-6 in terms of goodput under different loss rates. The goodput of CX-6 drops rapidly when the loss rate exceeds 0.1%. In particular, the CX-6 goodput is down to 25 Gbps when the loss rate exceeds 1%. Meanwhile, we monitor the MAC statistics counters in CX-6 and get its raw throughput of ∼97 Gbps, which indicates that most of the RNIC bandwidth is wasted on retransmission caused by go-back-N. The goodput of SRNIC drops much slower than that of CX-6. When the loss rate exceeds 1%, the goodput is still 75 Gbps, 3x higher (75 vs. 25 Gbps) than that of CX-6.

The good loss tolerance of SRNIC comes from both the efficiency of selective repeat and its careful software-hardware co-designs in §4.4.2.

Performance in large-scale lossy networks. We use ns-3 to simulate the transport behavior of SRNIC, and compare it with CX-6 and IRN in large-scale lossy networks. We simulate the fat-tree topologies with the server number ranging from 16 to 4096, with the (ToR, Aggregate, Core) switch number varying among five settings: (1, 0, 0), (4, 4, 0), (8, 8, 0), (64, 64, 16) and (128, 128, 64). The subscription ratio is 1:1 in all topologies. We equip each server with one 100 Gbps NIC connected to one ToR. ToR, Aggregate, and Core switches are connected via 400 Gbps links. The propagation delay of each link is 1 µs.

PFC is enabled for CX-6 but disabled for SRNIC and IRN. We use the traffic trace in Cache_Follower [36], where 53% of the flows are sized between 0 - 100 KB, 18% between 100 KB - 1 MB, and the rest are larger than 1 MB. We set the network load at 0.7 utilization, and configure other algorithm parameters based on their papers.

We primarily focus on three metrics, *i.e.*, average FCT, P99 tail FCT, and average slowdown [33]. The average FCT and tail FCT describe the performance of throughput-intensive flows, while the average slowdown shows the performance of latency-sensitive flows.

As shown in Figure 12, the performance of SRNIC is 1.9 - 2.2x better than CX-6 across all three metrics. As the cluster scale increases, SRNIC maintains stable performance, while the performance gap between SRNIC and CX-6 widens. Meanwhile, SRNIC and IRN perform similarly well as they use the same loss-recovery mechanism (selective repeat) and similar congestion control schemes (DCTCP vs. DCQCN).

7 Discussion

RDMA Protocol for lossy Ethernet. The RDMA protocol was originally designed and simplified for lossless Infiniband, and it *"does not support selective packet retransmission nor the out-of-order reception of packets"*, written in the Infiniband RDMA specification [1]. As a result, the current RDMA, by design, requires a lossless fabric to perform well.

Based on this requirement, when RDMA is introduced into Ethernet-based data centers, Ethernet is turned from lossy to lossless by introducing PFC, rather than re-designing an Ethernet-native or loss-friendly RDMA protocol.

A lossless Ethernet network, however, is inherently difficult to scale and hard to maintain for high availability. It is therefore desirable to look into the other end of the design spectrum: revising the RDMA protocol for a lossy network. This is the path taken by the pioneering work of IRN [33], and SRNIC. We hope these early attempts can inspire the re-design of a new RDMA specification for lossy network, which supports out-of-order packet reception and selective packet retransmission natively and efficiently, and ensures compatibility and interoperability among different protocol versions and RNIC vendors.

SRNIC vs. RoCEv2, iWARP and ToE. There exists a long debate [3, 6] between RoCE and iWARP [35] (ToE [2] is similar to iWARP in the sense of TCP offload). The former takes a bottom-up strategy: start from a minimal, hardware-friendly yet working transport (*e.g.*, go-back-0, no congestion control) and incrementally add more advanced mechanisms (*e.g.*, go-back-N/selective repeat, DCQCN/DCTCP) to make RoCE work better over various networks. The latter takes a top-down strategy: offloading the fully-compatible TCP/IP stack (which is already proven to work well over various networks at scale), and gradually reduce unnecessary complexity to improve hardware friendliness.

SRNIC takes a more balanced approach: it inherits the hardware friendliness (and thus high performance) from RoCE, and introduces only necessary features from TCP such as selective repeat and DCTCP.

SRNIC demonstrates that high network scalability and hardware friendliness can be achieved simultaneously with careful architecture and protocol co-designs. We believe that the best of both RoCE (hardware friendliness) and iWARP/TCP (high network scalability) can coexist as we have shown in SRNIC.

8 Related Work

Several works [30, 32, 42] aim at improving RDMA's network scalability via bringing advanced congestion control algorithms to RNICs.They control the queue length at switches and thus improve RDMA's performance at scale. Note that these works are orthogonal to ours and can be integrated into SRNIC if they are hardware-friendly.

Mellanox tries to improve RNIC's connection scalability via DCT [17] technology, which restricts the number of active connections and avoids QP exhaustion via dynamically creating and destroying QPs. However, such behavior may cause frequent flips of connections, resulting in increased latency and bandwidth waste [27]. StaR [39] improves RNIC's connection scalability at one side by letting the other side save states for it. However, this strategy highly relies on the asymmetric communication pattern, where the client with low concurrency can share its resources with the server with high concurrency, to improve the overall connection scalability.

Other software based transport solutions or DPDK-style NICs, *e.g.*, eRPC [24], FaSST [27], 1RMA [38], and Nitro [37], expect NICs to provide scalable connection-less service including packet transmission and reception, and leverage CPU to implement connection-related semantics. In these solutions, it is the CPU's responsibility to handle most of the transport-related tasks, including packet order maintenance, congestion control, and loss recovery. Though the scalabilities of these approaches are comparable to the software transport TCP, the heavy involvement of CPU results in higher CPU overhead, higher latency, and higher jitter than that of hardware-based transport. In contrast, SRNIC handles almost everything in hardware but leaves only part of retransmission in software, resulting in hardware-level performance in most cases when there is no packet loss, and software-level loss tolerance when packet loss happens.

9 Conclusion

This paper presents the design and implementation of SRNIC, a scalable RDMA NIC architecture, which addresses the connection scalability challenge, while achieving high network scalability, high performance, and low CPU overhead at the same time. Our key insight in SRNIC is to minimize RNIC's memory requirement, by eliminating as many on-chip data structures as possible in a simple yet performant way. Guided by this insight, we make a few RDMA protocol header extensions and architectural innovations to achieve the design goal. Our experiences in SRNIC tell us that existing RDMA header formats originally designed for a lossless environment, are not suitable for much large-scale, lossy data center networks. SRNIC therefore is our first attempt towards more scalable and performant, next-generation RoCE/RDMA designs.

Acknowledgments

We would like to thank our anonymous reviewers and shepherd Yashar Ganjali for their valuable comments. This work is supported in part by the Key-Area Research and Development Program of Guangdong Province (2021B0101400001), the Hong Kong RGC TRS T41-603/20-R, GRF-16215119, GRF-16213621, ITF ACCESS, the NSFC Grant 62062005, and a joint HKUST-ByteDance research project.

References

[1] Infiniband architecture volume 1, general specifications, release 1.2.1. www.infinibandta.org/specs, 2008.

[2] Information about the TCP Chimney Offload, Receive Side Scaling, and Network Direct Memory Access features in Windows Server 2008. https://docs.microsoft.com/en-us/troubleshoot/windows-server/networking/information-about-tcp-chimney-offload-rss-netdma-feature, 2008.

[3] The pitfalls in RoCE answered with respect to iWARP. https://www.chelsio.com/wp-content/uploads/2011/05/RoCE-FAQ-1204121.pdf, 2011.

[4] Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2 (IP routable RoCE). https://www.infinibandta.org/specs, 2014.

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

[6] RoCE vs. iWARP competitive analysis. https://network.nvidia.com/sites/default/files/pdf/whitepapers/WP_RoCE_vs_iWARP.pdf, 2017.

[7] Mellanox ConnectX-5 Product Brief. https://network.nvidia.com/files/doc-2020/pb-connectx-5-en-card.pdf, 2020.

[8] Mellanox ConnectX-6 Product Brief. https://network.nvidia.com/sites/default/files/doc-2020/pb-connectx-6-en-card.pdf, 2020.

[9] Microsoft Bing. https://www.bing.com/, 2020.

[10] Network Simulator 3. https://www.nsnam.org/, 2021.

[11] OFED Perftest. https://github.com/linux-rdma/perftest/, 2021.

[12] Mellanox NEO-Host. https://support.mellanox.com/s/productdetails/a2v50000000N2OlAAK/mellanox-neohost, 2022.

[13] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *Proc. OSDI*, 2016.

[14] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proc. SIGCOMM*, 2010.

[15] Luiz André Barroso, Jeffrey Dean, and Urs Holzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 2003.

[16] Wei Cao, Yang Liu, Zhushi Cheng, Ning Zheng, Wei Li, Wenjie Wu, Linqiang Ouyang, Peng Wang, Yijing Wang, Ray Kuan, et al. POLARDB meets computational storage: Efficiently support analytical workloads in Cloud-Native relational database. In *Proc. FAST*, 2020.

[17] Diego Crupnicoff, Michael Kagan, Ariel Shahar, Noam Bloch, and Hillel Chapman. Dynamically-connected transport service, July 3 2012. US Patent 8,213,315.

[18] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. When cloud storage meets RDMA. In *Proc. NSDI*, 2021.

[19] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proc. SIGCOMM*, 2016.

[20] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proc. SIGCOMM*, 2015.

[21] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. Tagger: Practical pfc deadlock prevention in data center networks. In *Proc. CoNEXT*, 2017.

[22] IEEE. 802.1 qbb—priority-based flow control. 2008.

[23] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *Proc. OSDI*, 2020.

[24] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter rpcs can be general and fast. In *Proc. NSDI*, 2019.

[25] Anuj Kalia, Michael Kaminsky, and David G Andersen. Using rdma efficiently for key-value services. In *Proc. SIGCOMM*, 2014.

[26] Anuj Kalia, Michael Kaminsky, and David G Andersen. Design guidelines for high performance RDMA systems. In *Proc. ATC*, 2016.

[27] Anuj Kalia, Michael Kaminsky, and David G Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram rpcs. In *Proc. OSDI*, 2016.

[28] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. Collie: Finding performance anomalies in RDMA subsystems. In *Proc. NSDI*, 2022.

[29] Feng Li, Sudipto Das, Manoj Syamala, and Vivek R Narasayya. Accelerating relational databases by leveraging remote memory and rdma. In *Proc. SIGMOD*, 2016.

[30] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proc. SIGCOMM*. 2019.

[31] Yuanwei Lu, Guo Chen, Zhenyuan Ruan, Wencong Xiao, Bojie Li, Jiansong Zhang, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Memory efficient loss recovery for hardware-based transport in datacenter. In *Proc. APNet*, 2017.

[32] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. In *Proc. SIGCOMM*, 2015.

[33] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for rdma. In *Proc. SIGCOMM*, 2018.

[34] Kun Qian, Wenxue Cheng, Tong Zhang, and Fengyuan Ren. Gentle flow control: avoiding deadlock in lossless networks. In *Proc. SIGCOMM*. 2019.

[35] Renato Recio, Bernard Metzler, Paul Culley, Jeff Hilland, and Dave Garcia. A remote direct memory access protocol specification. Technical report, RFC 5040, October, 2007.

[36] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proc. SIGCOMM*, 2015.

[37] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. A cloud-optimized transport protocol for elastic and scalable hpc. *IEEE Micro*, 2020.

[38] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F Wenisch, Monica Wong-Chan, Sean Clark, Milo MK Martin, Moray McLaren, Prashant Chandra, Rob Cauble, et al. 1rma: Re-envisioning remote memory access for multi-tenant datacenters. In *Proc. SIGCOMM*, 2020.

[39] Xizheng Wang, Guo Chen, Xijin Yin, Huichen Dai, Bojie Li, Binzhang Fu, and Kun Tan. Star: Breaking the scalability limit for rdma. In *Proc. ICNP*, 2021.

[40] Jian Yang, Joseph Izraelevitz, and Steven Swanson. FileMR: Rethinking RDMA networking for scalable persistent memory. In *Proc. NSDI*, 2020.

[41] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proc. IMC*, 2017.

[42] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *Proc. SIGCOMM*, 2015.

[43] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. Understanding and mitigating packet corruption in data center networks. In *Proc. SIGCOMM*, 2017.