Flor: An Open High Performance RDMA Framework Over Heterogeneous RNICs

Qiang Li, *Alibaba Group;* Yixiao Gao and Xiaoliang Wang, *Nanjing University;* Haonan Qiu, *Alibaba Group;* Yanfang Le, *AMD;* Derui Liu, *Alibaba Group;* Qiao Xiang, *Xiamen University;* Fei Feng, Peng Zhang, Bo Li, Jianbo Dong, Lingbo Tang, Hongqiang Harry Liu, Shaozong Liu, Weijie Li, Rui Miao, Yaohui Wu, Zhiwu Wu, Chao Han, Lei Yan, Zheng Cao, and Zhongjie Wu, *Alibaba Group;* Chen Tian and Guihai Chen, *Nanjing University;* Dennis Cai, Jinbo Wu, Jiaji Zhu and Jiesheng Wu, *Alibaba Group;* Jiwu Shu, *Xiamen University*

https://www.usenix.org/conference/osdi23/presentation/li-qiang

This paper is included in the Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation.

July 10–12, 2023 • Boston, MA, USA

978-1-939133-34-2

Flor: An Open High Performance RDMA Framework Over Heterogeneous RNICs

Qiang Li$^\diamond$*, Yixiao Gao*‡*, Xiaoliang Wang*‡*, Haonan Qiu*$^\diamond$*, Yanfang Le*$^\sharp$*, Derui Liu*$^\diamond$*, Qiao Xiang***,*
Fei Feng$^\diamond$*, Peng Zhang*$^\diamond$*, Bo Li*$^\diamond$*, Jianbo Dong*$^\diamond$*, Lingbo Tang*$^\diamond$*, Hongqiang Harry Liu*$^\diamond$*, Shaozong Liu*$^\diamond$*,*
Weijie Li$^\diamond$*, Rui Miao*$^\diamond$*, Yaohui Wu*$^\diamond$*, Zhiwu Wu*$^\diamond$*, Chao Han*$^\diamond$*, Lei Yan*$^\diamond$*, Zheng Cao*$^\diamond$*, Zhongjie Wu*$^\diamond$*,*
Chen Tian‡*, Guihai Chen*‡*, Dennis Cai*$^\diamond$*, Jinbo Wu*$^\diamond$*, Jiaji Zhu*$^\diamond$*, Jiesheng Wu*$^\diamond$*, Jiwu Shu**
$^\diamond$*Alibaba Group,* ‡*Nanjing University,* $^\sharp$*AMD,* **Xiamen University*

Abstract

Datacenter applications have been increasingly applying RDMA for ultra-low latency and low CPU overhead. However, RDMA-capable NICs (RNICs) of different vendors or different generations of the same vendor do not cooperate well, which could cause bandwidth imbalance in the production network and introduce new root causes of the PFC storms. Our key observation is that although the data path functions of heterogenous RNICs follow the same RoCEv2 specifications, their control path functions are vendor and version specific. To this end, we propose Flor, an open framework that provides a unified hardware data plane atop heterogeneous RNICs and a flexible software control plane running over host CPUs or NPU of RNICs and DPUs. The hardware plane requires no changes to current specifications. The software plane on-loads congestion control and reliability management in the large-scale lossy Ethernet with no PFC dependency. We implemented and evaluated Flor in both testbed and production clusters over Intel E180, Mellanox CX-4 and CX-5 and Broadcom RNICs. Experiments show that Flor achieves comparable performance to vanilla RDMA in many scenarios, including 1/4096 packet loss, 6000:1 incast, and large-scale cross-pod communication. Flor mitigates the performance gap of CX-4 and CX-5 RNICs from 24.3% to 1.3% when they are deployed together.

1 Introduction

Remote Direct Memory Access (RDMA) over Converged Ethernet has been widely deployed in datacenters [3, 5, 11, 14, 30]. It provides low latency and high throughput for many applications, *e.g.*, key-value store [21, 35], distributed transactions [8, 55], distributed memory [9, 56], remote procedure call (RPC) [20, 22, 47], storage systems [11], graph computing [43] and machine-learning systems [29].

With the increasing deployment of RDMA, modern datacenters adopted RDMA-capable NICs (RNICs) of different generations and vendors, *e.g.*, Mellanox ConnectX-(CX-)4/5/6 [49, 50, 52], BlueField [51], Intel E810 [17], and cloud-provider customized RNICs [10, 12, 42]. On the one hand, adopting more than one vendor avoids vendor lock-in, i.e., relying on devices of a particular vendor, which is a serious risk during global supply chain crises such as the COVID-19 pandemic [18, 45]. On the other hand, the disaggregated deployment of storage and computation systems separates the back-end services from the front-end services into different clusters, where each cluster can host different types of RNICs.

The coexistence of heterogeneous network devices in datacenters introduces new challenges [11, 14, 25]. First, devices may adopt different implementations of RDMA engines. It happens among not only different vendors but also different generations of devices of the same vendor. We have investigated the impact of various devices in a large-scale storage system that involves two generations of Mellanox RNICs, which have different variants of DCQCN. In a hybrid deployment of 16 50Gbps CX-4 and CX-5 NICs, we observed a severe bandwidth imbalance, where the average throughput of CX-4 NICs degrades to 28Gbps over a full-mesh traffic pattern. Furthermore, we test the congestion control behaviors of NICs from different vendors. Specifically, Mellanox RNICs set the same congestion control rate for packets with the same destination IP, while Intel E810 RNICs enforce congestion control based on flows with the same five-tuple. In addition, Broadcom RNICs [7] implement DCTCP [2, 6] as the congestion control algorithm, while Intel E810 RNICs implement a window-based DCQCN variant [19]. The different congestion control algorithms can further amplify the bandwidth imbalance.

Second, RDMA requires Priority-based Flow Control (PFC) to maintain a lossless network fabric. Diverse devices increase the risk of generating PFC pause frames, which can propagate to the whole network and cause the network to stop forwarding traffic. In addition, the parameter tuning for the PFC configuration is time-consuming on newly deployed devices [25, 57], which usually takes weeks or months in large-scale networks with multiple vendors. During the long-term operation of production networks, we have observed multiple sources of PFC pause frame generation at both end-hosts and switches. Specifically, we found that implementation bugs of switches and RNICs are one important root cause of PFC storms [11, 14]. In our datacenter, we record that the high loss rates occur due to diverse devices abnormality, system misconfiguration, and congestion of burst traffic in the production system, which has also been reported in prior work [58].

To cope with these issues, we need an open and unified framework to address the growing diversity of datacenter devices and give users the flexibility of RDMA programming to reduce the operational complexity of large-scale datacenter networks. Our key insight is that the RDMA data path, including memory semantics, needs fast and high-performance packet processing. In contrast, the control path including congestion control algorithms and the reliable re-transmission mechanisms, which are RTT-based operations, is relatively slow but needs to guarantee efficiency. This inspires us to rethink the functions division between hardware and software by on-loading congestion control and reliability modules to the software plane while strengthening the data path transport by following the standard RoCEv2 specifications [3–5] in the high-speed hardware.

We present Flor, an *open*, *unified* framework to support applications over heterogeneous networking devices. Flor *separates the data-path and control-path of RDMA transport with a hardware and software co-design* [24, 28, 37, 45, 52]. The data-path functions, *e.g.*, packet processing and bulk memory transfer semantics, remain on the hardware. Flor's data-path follows RDMA primitives without any modifications in hardware to maintain high performance. Flor strengthens Reliable Connection (RC) transport through hardware/software co-design to overcome the low-efficient hardware-based Go-Back-N retransmission [14]. Furthermore, we leverage Unreliable Connection (UC) transport [4] as the first citizen for out-of-order demands in datacenters [42, 46] as it supports the out-of-order delivery of messages between RDMA operations without any requirements on the hardware change. We adopt UD as a key element to enable selective retransmission [36] for RoCEv2 and deliver messages to the applications in an out-of-order manner [42].

The control-path includes a load-aware dynamic chunking module, an RDMA-semantic-compatible reliability module, and a congestion control module. The load-aware dynamic chunking module balances between the performance and the software control granularity. Flor proposes a software selective retransmission scheme by leveraging UC to process *out-of-order delivery*. Flor implements an RTT-based congestion control algorithm similar to Swift [26] but improves the RTT measurement accuracy of previous work [28] by 10× on 99^{th} percentile and 99.9^{th} percentile RTT. By onloading these functions to hosts or programmable devices [40] (*e.g.*, IPU core [18], DPU core [51], CPU or even GPU [1]), the software developers have the flexibility of customizing and generalizing these functions across heterogeneous RNICs. For example, Flor can also adopt emerging congestion control schemes [26, 30] and optimization of transport protocols (*e.g.* Swift, HPCC) instead of waiting for months or years of hardware upgrades.

We evaluate Flor through extensive experiments in an RPC benchmark and real production systems. We compare Flor with a customized RDMA library, XRDMA [32]. XRDMA implements the RPC interfaces with vanilla RDMA primitives. Compared to XRDMA, which suffers significant performance loss with 1/4096 packet loss ratio with lossy RoCE accelerations [53], Flor maintains steadily high throughput. Specifically, by deploying an RTT-based congestion control algorithm, Flor can handle 6000:1 incast with no throughput loss at run time. Flor achieves comparable performance as XRDMA for intra-pod communication and better performance than XRDMA for inter-pod communication. Our evaluations show that Flor reduces the bandwidth gap from 21.8% to 1.3% in the hybrid CX-4 and CX-5 deployment clusters and mitigates the performance gaps by 220% for RNICs of different vendors. For the production systems running big-data applications and cloud storage service, compared with XRDMA, Flor improves the job completion time of a big-data application job by 10% and achieves comparable latency and IOPS on the latency-sensitive cloud storage service. Specifically, the process of upgrading the existing RDMA framework, i.e., XRDMA, to Flor has little performance impact on the running applications. Our practical experience with Flor shows that Flor provides a non-stop and smooth upgrade from lossless RDMA to lossy Flor.

In summary, this paper makes the following contributions:

- The interoperability of devices in RDMA networks needs be better addressed. We study the impact of heterogeneous RNICs in the production network.
- We revitalize the RDMA support by introducing an open unified framework accommodating primary RNICs in the lossy datacenter networks.
- We implement the framework and verify its effectiveness and low software overhead in both testbed and realistic production systems.
- As far as we know, this is the first systematic work considering the operation with heterogeneous devices, which innovates future RDMA system design from the perspective of service providers.

2 Background & Motivation

2.1 RDMA Preliminaries

RDMA is a hardware transport that exposes network operation through verbs API. User-space applications initiate data transmission requests to RNICs by posting Work Queue Elements (WQEs) into queue pairs (QPs). After transmitting the data, the RNICs generate Completion Queue Elements (CQEs) into Completion Queues (CQs) as the transmit completion signals for users. RDMA supports three transport types: Reliable Connection (RC), Unreliable Connection (UC), and Unreliable Datagram (UD) [4]. Correspondingly, it

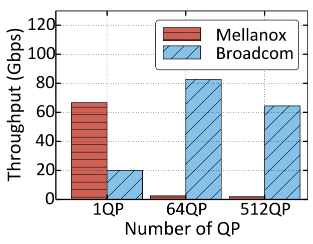


Figure 1: Throughput difference of a Mellanox RNIC and a Broadcom RNIC sending to an Intel RNIC.

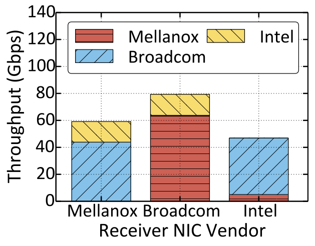


Figure 2: Throughput difference in a Mellanox-Broadcom-Intel hybrid deployment with a full mesh traffic pattern.

provides two well-known primitives: *SEND/RECV* are two-sided operations supported by all transports. *WRITE* is a one-sided operation supported by RC and UC, but *READ* is only supported by RC.

For connection-oriented services (UC and RC), QPs maintain Packet Sequence Number (PSN) and expected Packet Sequence Number (ePSN), respectively, on the sender and the receiver. RC QPs only receive packets with PSN correctly matching ePSN and increase ePSN after successful receiving. For RC, RNIC is responsible for retransmissions, where the receiver RNICs send acknowledge (ACK) packets. Once a packet is dropped, the sender must start retransmitting the lost packet. For UC, the transport does not retry messages with errors, and users must handle the error. Specifically, for UC QPs, when a packet of a verb is lost, the RNICs drop the whole verb, update ePSN, and continue to receive other verbs. Thus, compared with RC, it is more flexible for users to deal with out-of-order messages and potentially provide effective upper-level RPC service through software-defined reliability.

2.2 Production Experience

We present our production experiences demonstrating the difficulties of deploying the RDMA NICs of various generations and vendors in the same datacenter.

Interoperability of heterogeneous RNICs. We first investigate the performance gap between RNICs of different generations belonging to the same vendor. We run IB Perftest[1] in a cluster where 8 servers are equipped with CX-4 and another 8 servers are equipped with CX-5 RNICs. Given a full-mesh traffic pattern, *i.e.*, all servers send requests to each other, the throughput of CX-4 and CX-5 is 28Gbps and 41Gbps respectively. The throughput gap is 13Gbps (46.4%).

We then investigate the performance gaps among RNICs of different vendors. As shown in Figure 1, when a Mellanox RNIC [51] and a Broadcom RNIC [7] send traffic to an Intel RNIC [17] simultaneously. The configurations of the RNICs are depicted in §7.5. The Mellanox NIC gets 66Gbps, and the Broadcom NIC gets 20Gbps (220% of the performance gap) when each initiates one connection, *i.e.*. 1 QP. The Mellanox NIC gains less bandwidth, *e.g.*, 3Gbps, than the Broadcom NIC when the number of connections increases, *i.e.*, 64 QPs and 512 QPs. This causes unfair bandwidth share between applications hosted atop different RNICs. Figure 2 shows the throughput of each NIC with a full mesh traffic test (*i.e.*, all-to-all traffic) among the three NICs. We stack the throughput of each RNIC to the same receiver RNIC. For example, when a Broadcom NIC and an Intel NIC send traffic to a Mellanox NIC, the Broadcom NIC and the Intel NIC get the throughput of 43Gbps and 15Gbps, respectively (left bar). We can observe a similar performance variation when any two of the RNICs are competing with each other.

The difference in RNICs throughput is significant and leads to the computing tasks load imbalance on the nodes. We find that the root cause is the congestion control implementation difference or the congestion control algorithm difference among these heterogeneous RNICs. After we apply a unified congestion control algorithm, the performance gaps are eliminated (§7.5).

Operational challenges caused by PFC storming. PFC storm is a well-known problem [14, 15, 54, 57] that threats the system's availability if the pause frames are sent to the whole cluster [14]. RDMA systems in production adopt multiple mechanisms to mitigate the impact of these risks, such as PFC monitoring and watchdog, limiting the scale of PFC in a pod.

However, the PFC risk is not thoroughly eliminated and happens repeatedly with new causes. In addition to the known reasons of PFC storms, *e.g.*, the *slow receiver* [14] and switch hardware bug [11], we found that Machine Check Errors (MCE) caused by memory Error Correcting Code (ECC) and the lack of memory bandwidth can lead to the PFC storm when we introduce new RNICs in the datacenter. When MCE occurs on a server, RNIC receives data but can not DMA the data to the server memory. Thus, it sends excessive PFC pause frames to the neighbour switches and then spreads to the network. The occurrence frequency of MEC can be up to 1% [34], which leads to operational difficulty.

The lack of memory bandwidth also leads to PFC storms because CPUs and RNICs share the memory bandwidth on a server. When the CPU running applications preempts too much memory bandwidth, the memory bandwidth left for the RNIC is less than the network bandwidth [41]. Then the RNICs send PFC frames to prevent packet loss due to the RNIC buffer overflow. It is difficult to guard against every possible cause of the PFC storm. We expect to eliminate PFC from our production system while achieving performance compatible with a lossless network.

2.3 Motivation

These practical issues prompt us to rethink the usage of RDMA from the perspective of service providers. We aim to design an open and unified RDMA framework, which meets the following objectives:

- **Compatibility.** The open framework needs to be backward compatible with the legacy devices configured in the cluster.

[1]IB Perftest is a benchmark tool for measuring the throughput and latency of RDMA operations [13].

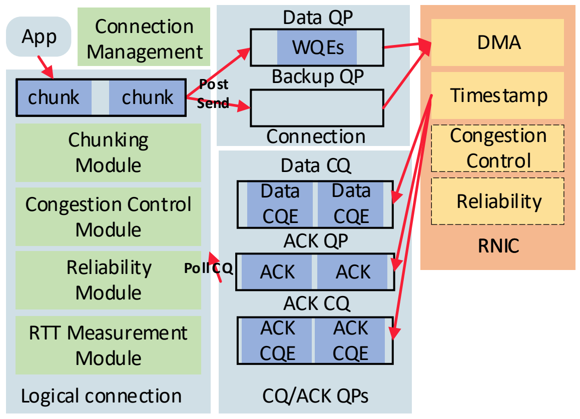


Figure 3: The framework of Flor

To this end, we abstract common features of available RNICs of main vendors and simplify the design by using the minimum set of functionalities in the hardware, *e.g.*, packetization, packet processing, message assembling, etc.

- **Flexibility.** To meet various application requirements and dynamic deployment of disaggregated services, the framework should support high feature velocity. It is programmable to realize efficient user-defined control mechanisms like new congestion control.
- **Availability.** The modern large-scale datacenters are built on Ethernet, which is a lossy network with multiple paths. Our framework is able to mitigate the impact of lossy Ethernet but fully utilizes the available network resources and maintains high performance [34, 42, 46].

3 Flor Design

3.1 Design Rationale

By investigating the RNICs of primary vendors, we notice that they follow the same RoCEv2 specifications in data-path but develop vendor-specific (even version-specific) control-path. Our key insight is that the RDMA data-path should be stable by following the standard specifications. At the same time, control-path needs to guarantee flexibility and availability. We can onload a subset of the transport functions to the software to provide programmability to developers. The design rationale is

Maintaining RDMA data-path specifications in hardware layer. RDMA data-path, including packet processing and memory semantics, is a per-packet-based operation which is fast and high-performance. To maintain low latency and low CPU utilization features of RDMA, Flor places the data-path in hardware, which covers packetization, packet processing, message assembling, and direct memory access between RNICs and host memory. Therefore, Flor is compatible with primary RNICs.

Onloading control-path to software layer. The most flexible features in the control-path are congestion control and transmission reliability with regard to the lossy Ethernet. The corresponding algorithms rely on the signals of packet latency, ECN notification, Inband Network Telemetry (INT), etc., the response interval of which can be several RTTs. Thus, Flor can onload the relatively slow control to the software layer by leveraging its programmability but has little influence on system efficiency. Notably, with the development of programmable devices, we can realize functions of control-path in not only hosts but the NPU [18] of RNICs or DPUs [51].

3.2 Architecture

Flor is an open, unified, high performance RDMA framework over lossy Ethernet in large-scale datacenters. The architecture is shown in Figure 3.

Data path. Since the process details of RDMA operations on QPs are dictated in the RDMA protocol [4], utilizing standard RDMA operations, which are supported by all RNICs, to transfer data among heterogeneous RNICs can achieve comparable performance. Flor takes RDMA **WRITE** and **SEND** as the base WQEs for the data transferring and receiving because they both support RC and UC and maintain high performance of RDMA. Notice that RDMA *READ* has a known performance issue [21] and only supports RC. Flor uses the RDMA *SEND* WQEs to transfer small messages (*e.g.*, $\leq$32KB) and *WRITE* to transmit large messages. For large messages, Flor needs an extra round-trip to exchange remote memory address and buffer size with remote servers. Note that the memory information exchange requires once for each large message. Flor prioritizes the *SEND* WQEs over large messages to avoid that head-of-line blocking to the small messages. On top of the RDMA verbs, Flor provides a *message-based* communication interface to support RPCs favoured by most datacenter applications [20].

Control path. The control-path of Flor consists of five flexible software modules: *Connection Management*, *Chunking*, *Reliability*, *Congestion Control*, and *RTT Measurement*.

- *Connection Management.* This module establishes and releases connections and manages backup QPs. Data are transmitted through QP and backup QPs, which take over the RDMA requests in place of the malfunctioned primary QP. The data CQs provide data completion events. The ACK QPs and CQs are used for sending and receiving software ACKs when using software reliability. Through QP management, Flor can abstract these backup QPs and present them as one QP to upper-level applications.
- *Chunking.* The *Chunking* module splits large messages into small RDMA requests, *i.e.*, *chunk*. Flor takes the *chunk* as the base unit of the selective repeat algorithm and congestion control algorithm, instead of a packet at the traditional transport [26, 57]. A chunk is sent to the network via a WRITE or SEND Work Queue Element (WQE).
- *RTT Measurement.* The *RTT Measurement* module collects the NIC hardware timestamp, synchronizes the hardware and software timestamp, and then updates RTT, which is

used as the signal of network congestion, retransmission and link failure detection.

- *Reliability.* Each WQE is passed to the *Reliability* module, which stores the transport information and maintains the state for packet loss detection and RTT calculation. We develop a software selective retransmission mechanism by tracking every request sent to the network.
- *Congestion Control.* The *Congestion Control* module takes the congestion signals, *e.g.*, RTT, ECN, INT [39], and drop events, to calculate the congestion window or sending rate according to the congestion control algorithms. By default, we apply an RTT-based congestion control algorithm to eliminate the impact of complicated parameter tuning of congestion control on diverse switches [26].

3.3 Optimization and Deployment.

The key challenge for Flor is to offer flexibility while providing comparable performance to the vanilla RoCEv2 stack. We make the following optimizations:

- **Maintaining RDMA performance using the software/hardware co-design.** The RMDA hardware solution provides high throughput, low latency, and low CPU overhead. To maintain these advanced properties, we adopt a dynamic chunking mechanism to tune the size of messages for slow control-path when tracking the software congestion control and loss recovery. The overhead of the control-path functions onloading to the software layer is low because we apply large granularity of messages instead of packet processing (§ 4).
- **Enhanced UC with Selective Retransmission.** The packet loss rate in datacenters is actually low, which will not trigger frequent re-transmission. Designing a correct reliability mechanism while keeping the zero-copy memory semantic is the main challenge that Flor handles. UC has the property that RNICs can deliver the messages to the host without waiting for the previous ones to complete. Flor leverages this property to design a more efficient retransmission scheme, i.e., selective retransmission [36] without any hardware change to speed up the application processing [42] (§ 5).
- **Enhanced RC with Correctness.** RC is one of the data-path transport supported by Flor. Go-back-N, the RC's retransmission mechanism, is known to have low efficiency [36]. Flor enhances the Go-back-N mechanism by adding an additional software retransmission scheme. We address the correctness issue introduced by the software retransmission, where the retransmitted RDMA operators may overwrite the memory region that has been submitted to applications(§ 6).

Flor users can select a combination of these software modules in different scenarios. The software congestion control and reliability modules can be bypassed or replaced by the hardware functionalities. Table 1 shows some recommended configuration combinations for different deployment scenarios in our production. For example, in a PFC- and ECN-disabled CX-4 cluster, Flor provides RDMA service by enabling chunking and software congestion control with hardware-based (RC) or software-based (UC) reliable transport. For cross-pod applications, software-based (UC) reliability is recommended to tolerant packet loss.

Scenarios	Chunking	Reliability	CC
Intra-pod, PFC-enabled	No	*RC*	*HW*
Intra-pod, PFC- and ECN-disabled	Yes	*RC* or *UC*	*SW*
Across-pod Applications	Yes	*UC*	*SW*
CX-4/5 Hybrid	Yes	*RC* or *UC*	*SW*

Table 1: Recommended choices of modules in some scenarios. CC represents congestion control. *HW* indicates hardware-offloaded modules, *SW* indicates software-implemented modules.

4 Dynamic Chunking

The chunking algorithm determines the granularity of RDMA requests bursting into the network. A large chunk size may result in a traffic burst that causes congestion and incurs high recovery costs in case of packet loss, while a small one leads to more CPU costs. Therefore, the key design point is dynamically adapting chunking size according to the current network status, which helps achieve both good performance and fine-grained traffic control. More specifically, it tries to adopt large chunk sizes with hardware SEND or WRITE operations to reduce CPU cost when the loss rate is low. Once packet loss occurs, it applies chunk-slicing and retransmission of dropped chunks by software.

Flor uses the estimated RTT as the default feedback signal of network status for the dynamic chunking strategy. The estimated RTT is not only used as the feedback signal of the dynamic chunking algorithm but also used as the congestion control signal, as well as a timeout signal for reliability.

4.1 Accurate RTT Measurement

The accuracy of RTT measurement directly impacts the performance of all these components. Different from the approach [26] where measures RTT based on per-packet timestamp, Flor measures RTT for the chunks with different sizes. RoGUE [28] firstly devises a way of RTT measurement for dynamic verb sizes on RNICs and utilizes the hardware timestamp functionality of RNICs to get an accurate timestamp to calculate RTT. Flor takes the RoGUE's methodology to measure RTT for the RC transport and further improves the RTT measurement accuracy for UC transport.

Figure 4 shows the RTT measurement method in Flor for the UC transport. Note that leveraging hardware timestamp to measure RTT requires synchronizing the software and hardware clock (Appendix A.2.1). On the sender side, the timestamps of the data WQE sending completion (T_1) and the corresponding ACK receiving completion (T_4) can be obtained from the hardware. On the receiver side, the timestamp of the data WQE arrival (T_2) is read from the hardware and the timestamp of the corresponding ACK WQE posting time (T'_3) is accessible through software. (T'_3 - T_2) is computed in the receiver and sent back to the sender by a subsequent ACK packet. Thus RTT can be calculated with:

$$RTT = (T_4 - T_1) - (T'_3 - T_2). \quad (1)$$

However, the measurement is not absolutely accurate as T'_3 is the post time rather than the actual send completion time of the ACK (T_3) due to the queuing of sending requests in NICs. In addition, under a heavy load, the ACK can be delayed up to milliseconds upon receiving by the sender software due to the head-of-line blocking by the data WQEs and the QP scheduling policies. As a result, on the one hand, the measured RTT can be increased by this overhead, which inaccurately reflects the network congestion; On the other hand, this also prolongs the congestion feedback loop such that the sender can not respond to the network congestion in time.

Flor uses two optimization approaches to improve RTT measurement accuracy and shorten the feedback loop delay. First, Flor uses a high-priority UD QP to send and receive ACKs to avoid head-of-line blocking and scheduling delays on RNICs. Second, Flor uses a separate completion queue for ACKs and polls it prior to the data completion queue in each batch. Our measurement shows that the measured tail RTT without optimization can be up to several milliseconds due to QP scheduling (Figure 16 in Appendix A.2.2). The overestimated RTT can cause an unnecessary window decrease. Using a separate QP and completion queue for ACKs improves the RTT measurement accuracy by 10× on tail RTTs.

4.2 Chunking Strategy

Flor extracts the chunking algorithm as a module such that users can specify their own chunking algorithms. Here we present the default algorithm used by Flor. The key idea is to dynamically reduce the chunk size when the RTT of network gets worse and increase the chunk size when its status gets better. We initialize the chunk size by the minimum value of the available congestion window (acwnd) or bandwidth-delay product (BDP) ($chunk\_size \leftarrow \min\{acwnd, BDP\}$). Then we update the chunk size for each RTT.

As shown in Algorithm 1, we use estimated RTT, which reflects network queuing, as the feedback signal of chunking in Flor. We maintain two smoothed RTTs (rtt_s and rtt_l) with Exponentially Weighted Moving Average (EWMA) [31] using different indexes α (the parameter that controls the weight of new feedback). The short-term RTT rtt_s demonstrates the up-to-date congestion status, while the long-term RTT rtt_l indicates the common status of the connection.

Generally, we define a span of expected RTT denoted by ($\beta_{max} * rtt_l - \beta_{min} * rtt_l$). Here β_{max} and β_{min} are configurable parameters to identify the upper and lower bounds. ($\beta_{max} * rtt_l - rtt_s$) indicates the position of short-term RTT in the RTT span. The Algorithm divides the RTT span linearly, as shown in line 4 of Algorithm 1. $size_{max}$ is the maximal chunk size [2] For example, if the rtt_s reduces to the lower bound of RTT span ($\beta_{min} * rtt_l$), it indicates that the status of network is good and the chunk size increases to the $size_{max}$. On the contrary, if the rtt_s increases to the upper bound of RTT span ($\beta_{max} * rtt_l$), it indicates that the load of the network is high and we should reduce the chunk size to the $UNIT\_SIZE$.

Algorithm 1 An RTT-based Chunking Algorithm

```
Input: RTT $rtt$, available congestion window $acwnd$
Output: $chunk\_size$
1: update short-term RTT $rtt_s$ with $rtt$, $\alpha_s$ via EWMA
2: update long-term RTT $rtt_l$ with $rtt$, $\alpha_l$ via EWMA
3: $size_t \leftarrow size_{max} * (\frac{\beta_{max}*rtt_l - rtt_s}{\beta_{max}*rtt_l - \beta_{min}*rtt_l})$
4: $chunk\_size \leftarrow \min\{size_t, acwnd\}$
5: return $chunk\_size$
```

Note that the chunking module still applies to have a fine-grained traffic control if Flor uses a rate-base congestion control, *e.g.*, DCQCN. To support the reliability design (§3.2), Flor aligns the $chunk\_size$ to $UNIT\_SIZE$ (the minimal $chunk\_size$), *i.e.*, $chunk\_size$ is exactly multiple times of $UNIT\_SIZE$. Note that to prevent deadlock of the congestion window, the $chunk\_size$ is set to $UNIT\_SIZE$ when the available congestion window is smaller than $UNIT\_SIZE$. The $UNIT\_SIZE$ can be equal to the value of the current MTU. To mitigate the impact of packet loss, Flor adopts different chunking mechanisms for RC or UC transport, respectively. For RC transport, Flor directly reduces the large chunk sizes to the minimum chunk size of one $UNIT\_SIZE$ to reduce retransmission overhead and avoid the live lock of retransmission. For UC transport, Flor relies on congestion window, which will cut its size by a half upon a packet loss. And finally, the chunk size becomes $UNIT\_SIZE$ when continuous packet loss happens. In particular, Flor can adopt Selective Retransmission with UC in Section 5 to reduce chunk retransmission under high packet loss and achieve better transmission efficiency than Go-Back-N.

5 Selective Retransmission with UC

UC transport only drops the verbs that have packets dropped but does not drop the subsequent successfully-delivered verbs. Thus, based on the chunking mechanism that splits RDMA messages into varied sizes of WQEs, Flor is able to design reliability mechanisms (sequence number, acknowledgement, and retransmission) at the granularity of chunking WQEs. The transmission of RDMA $WRITE$ operations does not need any reordering buffer because $WRITE$s are directly DMA-ed into the host buffer. However, there are still some challenges to implement reliability in software for one-sided RMDA $WRITE$ operations:

- **Challenge #1: Additional data copy.** Since RDMA $WRITE$ writes an array of memory pieces to one continuous

[2] By default, $size_{max}$ is 64KB, which is a trade-off between the CPU efficiency to transmit large chunks and the retransmission overhead of packet loss to transmit smaller ones.

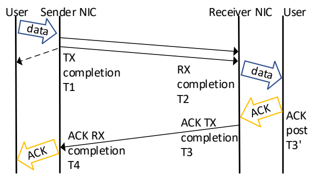


Figure 4: RTT measurement for UC in Flor. $T1$–$T4$ are timestamps of events.

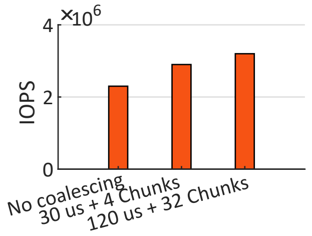


Figure 5: Performance impact of ACK coalescing.

remote address, Any extra content added to the chunks of a large message, including information needed by reliability mechanisms such as sequence numbers, can break the original message, which then incurs a memory copy at software to assemble the scattered memory into the original message.

- **Challenge #2: Software ACK.** *WRITE* is a one-sided RDMA operation that bypasses the CPU of the receiver. The receiver can not know whether a *WRITE* succeeds or fails, which is one of the main issues when the one-sided operator is used in practice. Flor uses WQEs to encode software ACK messages as UC does not generate hardware ACK packets. Without careful design of the ACK message, the high frequency of the software ACK will significantly degrade the performance.
- **Challenge #3: Repetitive memory access.** The retransmission of RDMA *WRITE* operation may cause a data integrity issue as RNICs can write to a piece of memory already submitted to the application.

To solve these problems, some novel designs, including sequence number space for *WRITE_WITH_IMM*, software ACKs, and two-sided retransmission, are adopted in Flor's reliability mechanism.

Sequence number space for *WRITE_WITH_IMM*. To assemble the chunks into the original message on the receiver without the extra data copy (**Challenge #1**), Flor uses *WRITE_WITH_IMM* to generate signals to software for the arrival of chunks. One feature of *WRITE_WITH_IMM* is that it can carry an extra 32-bit *imm_data* set by the sender. With *WRITE_WITH_IMM*, the receiver can detect the arrival of RDMA verbs and receive the chunk number without polluting the application memory. For *SEND*-transported small requests, Flor adds an additional header in the payload to carry additional information, including the sequence numbers.

However, to reassemble the initial messages from the chunks, we need another sequence number for the chunks. Note that chunks of a large message are also not continuous in the sending queue since some *SEND*s operations (*e.g.*, retransmissions and the address-exchanging messages of *WRITE*) are prioritized in Flor.

Similar to QUIC [27], Flor encodes two sets of sequence numbers into each RDMA WQE: *global sequence number* and *reliability sequence number*. The global sequence number is a 64-bit value, and all the RDMA WQEs have a unique global sequence number to identify the sequence within a QP. The reliable sequence number is used to identify the sequence within the same type of WQEs, i.e., WRITE WQEs and SEND WQEs have separate reliable sequence number space. These two sequence number spaces also allow Flor to identify the original WQEs and the retransmitted WQEs by different global numbers such that ACK information and the timestamp carried in ACKs are clear. The retransmission WQEs share the same reliable sequence number with the original WQE. More details can be found in Appendix A.1.1.

Software ACK. Flor acknowledges every WQE, but generating an ACK for each WQE can cause high overhead for small-message traffic (**Challenge #2**). Flor puts multiple sequence numbers into one software ACK to reduce the CPU overhead. The detailed ACK format is explained in Appendix A.1.2. However, ACKs should be sent timely since ACKs carry RTT and WQE numbers used in the congestion control and reliability mechanism. Thus, Flor sets these trigger rules for receivers to send an ACK immediately: (1) the cumulative number of WQEs; (2) the cumulative size of WQEs; (3) the last WQE in the congestion window signed by a hint bit in the header or IMM_DATA field; and (4) an out-of-order reliable sequence number, whichever reaches first. Flor also sets a timer as a trigger because the tail of a flow or small bursts may not have enough data to trigger the cumulative counters.

To show the impact of different ACK triggering frequencies, we set up an experiment with a client and a server, and each is equipped with a CX-4 dual-port NIC. There are 8 threads and 64 QPs on each node. Figure 5 shows the IOPS in a 128-byte request and response RPC benchmark with different ACK triggering mechanisms. The ACK coalescing mechanism improves ∼40% IOPS compared to a per-WQE ACK mechanism (*No coalescing* in Figure 5). In addition, among different coalescing WQE sizes and timers, the setting of 120μs timer and cumulative counter of 32 WQEs achieves a satisfying IOPS, which is the default configuration of Flor.

Two-sided retransmission. When the sender detects an out-of-order delivery from an ACK or a timeout event, it retransmits the WQEs. Flor handles the retransmissions by *SEND*, since spurious retransmissions of *WRITE* may cause a data integrity issue (**Challenge #3**). The data integrity issue can happen when WQEs are queued in the network for a long time, which incurs timeout retransmission at the sender. In this case, the original WQE is received by the receiver, and the whole message is submitted to the upper-layer application. The application can write the content of the message as it needs. However, the subsequent retransmitted *WRITE* WQE arrives at the receiver and overwrites the memory region that the application has already changed without informing the CPU of the receiver. Flor retransmits WQEs by *SEND* through the same QP for the lost one to avoid uncontrolled memory access from RNICs. In a *SEND* operation, the data is written into a piece of pre-posted memory. Flor then copies the data into its desired location if the message has not been submitted to the application. If the retransmission *SEND* arrives earlier on the receiver, the following original *WRITE_WITH_IMM* will be dropped by the receiver RNIC because its hardware packet sequence number is smaller than the expected hardware packet sequence number, which is updated because of the arrival of the *SEND*.

6 Enhance Hardware Retransmission

Flor is compatible with hardware reliability by using RC to reduce software costs and achieve better performance. Note that Flor supports both one-sided and two-sided RDMA operations on RC QPs. The hardware retransmission of RDMA operations is out of the control of software congestion control. This has potential risks of incurring network congestion since the size of inflight data can be much larger than the software congestion window. Flor improves the hardware reliability by adding a software retransmission scheme. Flor sets short retransmission retry times in RC QPs to limit the size of data exceeding the software congestion control window.

If the retransmission fails for the retry times, the QP is turned into error states by the RNIC hardware. Turning the error states of the QPs into the working states takes a long time, e.g., 5ms. Instead, Flor resubmits the uncompleted inflight WQEs to another pre-connected QPs, called backup QPs [28], and flips the backup QP to be the primary QP to continue data transmission. At the same time, Flor re-connects the original QP in the background. The inactive backup QPs do not consume additional cache resources on RNICs and thus have no side effect on performance [28]. Our practical experience shows that using one retry time incurs too many QP switches in some extreme cases *e.g.*, large-scale incast. By default, Flor uses two retry times and two backup QPs for each connection. Compared to QP reconnecting, switching to backup QPs costs less time, *i.e.*, ~60μs.

A racing issue occurs when QPs turn into the error state occasionally: the sender may return a failure of a WQE while the receiver successfully receives it. This case happens when the QP at the sender turns into the error state with successful inflight operations. In such a case, the sender posts a duplicated WQE mistakenly on the backup QP of the logical connection. Another corner case that causes the same problem is that the sender times out when the hardware ACK is on the flight. Flor retransmits the WQEs with RDMA *SEND* and checks if the message has been submitted to the application before the data are copied into the destination application's memory.

7 Evaluation

We evaluate Flor by answering the following questions:

1. The software overhead of Flor in the 100Gbps network (§7.2) and its robustness against packet loss (§7.3)?
2. The behaviour of Flor in both intra-pod and inter-pod communications when PFC is disabled (§7.4)?
3. The effectiveness of Flor in a hybrid deployment with heterogeneous RNICs (§7.5)?
4. The performance of Flor's default *Congestion Control* in large-scale incast scenario (§7.6)?
5. The impact on services when upgrading from the current network to Flor in production systems (§7.7)?

Cluster	Nodes	RNICs
A	100	CX-4 Lx 25Gbps dual-port
B	16	8 × CX-5 25Gbps dual-port 8 × CX-4 Lx 25Gbps dual-port
C	48	CX-5 100Gbps dual-port
D	3	Intel E810 100Gbps dual-port Broadcom P2100G 100Gbps dual-port Mellanox BlueField-2 100Gbps dual-port

Table 2: Clusters setups used in our evaluation.

7.1 Experiment Setup and Benchmarks

Cluster setup. Table 2 lists four clusters used in the evaluation. The default RDMA configurations of our clusters are that: (1) for CX-4 lossless RNICs, PFC is enabled on ToR and Leaf switches but disabled on Spine switches; (2) for CX-5 lossy RNICs, PFC is disabled on all switches, and the *lossy RoCE acceleration features* [53] are enabled.

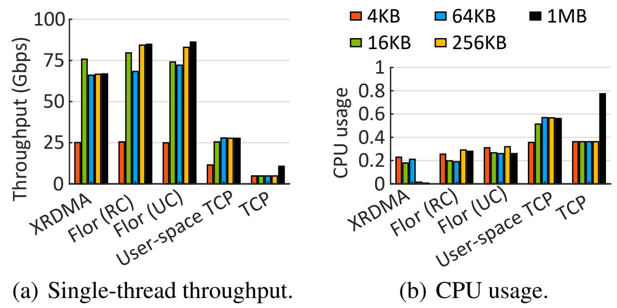
Baseline and workload. Our RPC system used in evaluation supports XRDMA, Flor, user-space TCP, and kernel TCP. XRDMA is a vanilla RoCEv2-based RPC library, which is deployed in the clusters listed in Table 2 before upgrading to Flor. The user-space TCP is based on DPDK. Two applications run on top of our RPC framework: a Map-Reduce-like application and a distributed block storage service.

Configuration for Flor. In the clusters, we configure one specific priority queue on both switches and RNICs. The PFC and ECN are enabled on this priority queue (lossless queue). Besides, we reserve another priority queue (lossy queue) for Flor in which PFC and DCQCN are disabled. The coalescing ACK parameters are 120μs timer, 32 WQEs and 32KB data at most. The base RTT used in the software congestion control is 50μs. The minimal and maximal chunk size is 1KB and 64KB, respectively.

7.2 Software Overhead.

Flor introduces additional CPU cost due to the software implementation of reliability mechanism and chunking. To evaluate its impact, we compare the single-core performance of different network protocol stacks with our RPC benchmark, including XRDMA, Flor (RC), Flor (UC), user-space TCP, and kernel TCP. The I/O depth (*i.e.*, the maximal number of inflight RPC requests) is 8. We vary the RPC request size from 4KB to 1MB and fix the response size at 128 bytes. The servers are equipped with Intel CPUs (2.9GHz) which have 96 logic cores, and CX-6DX 100Gbps dual-port RNICs (not listed in Table 2).

Figure 6(a) demonstrates the throughput of all the stacks with different RPC sizes. The single-thread throughput of Flor



(a) Single-thread throughput. (b) CPU usage.

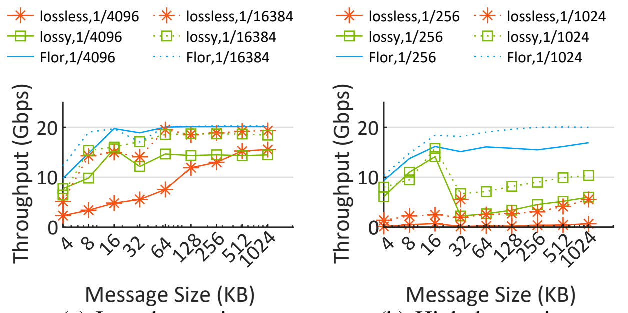
Figure 6: Single-thread throughput and CPU usage of Flor and other different network stacks.

and XRDMA can reach to ~88Gbps while user-space TCP and kernel TCP achieve up to ~30Gbps even with large RPC size, i.e., 1MB. Notice that due to the hardware and RoCEv2 protocol overhead, such as headers of Ethernet, IP, UDP, and IB with 1024B MTU, the standard RDMA benchmark, i.e., Perftest [13] achieves a maximum bandwidth of 88Gbps. This shows that the *Chunking* mechanism does not hurt the single-thread throughput of Flor. We observed that the performance degradation of Flor (RC/UC) happens at 64KB due to extra memory information exchanges for using RDMA WRITE to transmit each large message and dismisses as message sizes increase. The throughput of large requests in XRDMA is lower than Flor because XRDMA uses RDMA *READ* operation to transfer the large requests, and RDMA *READ* operation has inflight bound on RNICs along with some well-known performance issues [16, 23]. When using a chunk of 4KB, Flor can handle over 770K chunks per second with a single thread. Flor is able to maintain high throughput as the chunk size is usually larger than 4KB, and applications often adopt multiple threads.

We then estimate the corresponding CPU usage. The CPU usage is obtained from *perf* [38] tool since we use polling mode for RDMA. Notice that in our production storage system, it adopts a run-to-completion model based on a co-routine IO framework [11], where the network polling for disk read or write uses the same core with storage protocol processing in concurrent execution. As shown in Figure 6(b), Flor takes less than 0.3 CPU core for large data size of 1MB message in 100 Gbps network. Most modern servers usually have large numbers of cores (>96) and 0.3 CPU cores usage (<0.4% usage) has little impact on the production system.

Flor maintains low CPU cost because it leverages zero-copy features of RDMA and mainly deals with lightweight control events for congestion control and reliability. In addition, Flor is compatible with different platforms, which can further reduce the host CPU cost by offloading Flor to SmartNICs, and FPGA in computation-intensive hosts.

We also measure the single-core throughput of Flor at 200Gbps RNICs and show that Flor can achieve comparable throughput as vanilla RDMA. Besides, we show that Flor outperforms the other network stacks, *i.e.*, SNAP [33], eRPC [20]



(a) Low drop ratio. (b) High drop ratio.

Figure 7: The throughput of lossless, lossy XRDMA and Flor under different packet drop ratio.

(as shown in Table 4).

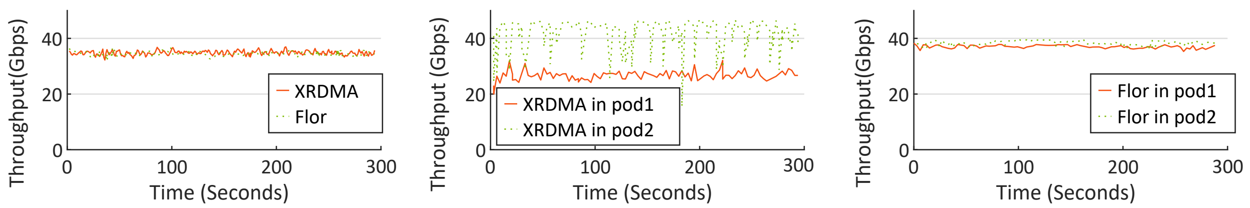
7.3 Performance with Packet Loss.

To validate the effectiveness of the software *Reliability* mechanism of Flor, we take two CX-5 RNICs from cluster *B* in Table 2 and disable one port on each RNIC. We manually configure packet drop ratios on the RNIC of the receiver. We compare Flor against XRDMA using the lossless and lossy configuration of CX-5. The congestion control is disabled to avoid transmission rate back-off due to packet loss.

We use various request sizes (4KB~1MB) under four packet drop ratios (two high ratios of 1/256 and 1/1024, and two low ratios of 1/4096 and 1/16384). As shown in Figure 7, Flor outperforms the lossy and lossless setup across all the drop ratios. Due to software selective retransmission, Flor is able to maintain a performance close to that of zero packet drop at the low packet drop ratios. Its performance decreases slower than the lossy and lossless setup at the high drop ratios. We observe that the lossy acceleration feature achieves higher throughput with the occurrence of packet loss compared to the lossless setup, i.e., the lossy acceleration features of CX-5 does improve the packet loss recovery performance. However, the throughput of lossless and lossy RDMA both decrease dramatically when the drop ratio is larger than 1/4096. It indicated that the lossy acceleration features of current hardwares still cannot maintain good performance under high packet loss. Flor achieves similar results by performing the same experiments through manually configuring the random packet drop ratio on the port of the ToR switch connecting to the live port of the RNIC at the host.

7.4 Intra- and Inter-Pod Traffic

Flor uses an advanced congestion control to enable lossy RDMA support, eliminating the PFC dependency while maintaining high performance. We validate the performance gain of Flor in large-scale intra- and inter- pod transmission through cluster A (pod1) and B (pod2) (Table 2). We use the default configuration in our clusters in the tests of XRDMA: PFC for RoCE traffic is only enabled on ToR and Leaf switches (i.e., intra-pod) and disabled on Spine switches



(a) Intra-pod full-mesh test. (b) XRDMA in inter-pod full-mesh test. (c) Flor in inter-pod full-mesh test.

Figure 8: The throughput of XRDMA and Flor in cross-pod scenarios.

(i.e.,inter-pod). Similarly, we configure the intra-pod traffic of Flor (UC) on the lossless queue and the inter-pod traffic of Flor (UC) on the lossy queue. For each node, a client sends large RPC requests (512KB) to each server on other nodes with IO depth of 8. The size of the RPC response is 128 bytes. Each client and server uses 4 threads in the tests.

As shown in Figure 8(a), for intra-pod traffic, the average throughput of Flor (UC) is comparable to XRDMA which is stable at ∼35Gbps. For inter-pod traffic test, as shown in Figure 8(b) and 8(c), the throughput of XRDMA in one pod shakes fiercely between 35Gbps and 45Gbps, and the throughput in the other pod oscillates between 35Gbps and 20Gbps. The unstable and unbalanced throughput of XRDMA is caused by packet loss as PFC is disabled in inter-pod switches, and DCQCN can not prevent packet loss. In contrast, the throughput of Flor (UC) is stable and balanced between the two pods. In summary, Flor can achieve higher and more stable throughput than XRDMA for lossy inter-pod tests. The clients in pod1 suffer more from throughput loss because they send more cross-pod traffic and experience more packet loss in this full-mesh traffic pattern.

7.5 Heterogeneous RNICs

To evaluate the effectiveness of Flor over heterogeneous RNICs, we test with 8 CX-4 and 8 CX-5 RNICs in cluster B (Table 2) with PFC and DCQCN enabled. We run the RPC benchmark with a full-mesh traffic pattern atop of Flor and XRDMA. When using XRDMA, the average throughput of CX-4 and CX-5 RNICs is 33.2Gbps and 41.3Gbps, respectively. The throughput gap between CX-4 and CX-5 is 8.1Gbps (24.3%). When using Flor, the throughput of CX-4 and CX-5 RNICs with Flor is 37.1Gbps and 36.6bps, and the throughput gap is 0.5Gbps (1.3%). This indicates that Flor eliminates the throughput gap between CX-4 and CX-5 RNICs by replacing the hardware congestion control with a unified software congestion control, which minimizes the performance difference introduced by the control path of heterogeneous hardware.

To verify the effectiveness of Flor over RNICs from different vendors, we run perftest among 100Gbps RNICs, including Mellanox BlueField-2 [51], Intel E810-C RNIC [17] and Broadcom NetXtreme P2100G RNICs [7]. For congestion control, BlueField-2 only supports DCQCN, P2100 only supports DCTCP, and E810-C supports DCQCN, DCTCP and Timely. To clarify the unmatched performance introduced by different congestion control algorithms, we choose to set DCQCN for BlueField-2 and E810-C, and DCTCP for P2100G with PFC and ECN enabled on RNICs and the connected switches. Each sender issues 512 QPs and sends traffic with 64KB data blocks. Four Flor configurations, i.e., *PFC with hardware congestion control (CC) and Flor*, *hardware CC and Flor*, *Flor*, and *Flor with fixed cwnd* are tested. *Flor with fixed cwnd* means the software congestion control algorithm has a fixed congestion window and does not change throughout the whole experiments. We set the fixed congestion window size as one bandwidth-delay product in this experiment.

Figure 9 shows the throughput of each NIC under the full-mesh traffic pattern. BlueField-2 and P2100G get the same bandwidth when they are competing with each other to send traffic to E810-C. Compare with Figure 2, with the configurations of PFC + hardware CC + Flor (1), hardware CC + Flor (2) and Flor (3), and we see that Intel NIC gets the same throughput, i.e., 18Gbps, when competing with Mellanox NIC and Broadcom NIC. We see the performance gap between Intel NIC and Mellanox NIC when they send traffic to the Broadcom NIC, even if we only apply the Flor's congestion control (3), where the Intel NIC gets 19Gbps, and the Mellanox NIC gets 56Gbps, 194% of the performance gap.

The reason is that Flor's congestion control takes the RTT as the congestion signal. Intel NIC has a higher packet processing time, which causes the estimated RTT between the Intel NIC and the Broadcom NIC to be higher than the one between the Mellanox NIC and the Broadcom NIC. Thus, the RTT-based congestion control algorithm reduces window size in Intel NIC and results in lower throughput. This indicates Flor's congestion control needs to be further improved by taking the NIC processing delay into account. If we fixed the congestion window, i.e., Flor with fixed cwnd (4), the bandwidth is less-skewed shared between the Mellanox NIC and the Intel NIC.

7.6 Large-scale Incast

We build a group of incast tests in cluster A (Table 2) to evaluate the performance of Flor's congestion control. We measure the throughput and Out-of-Sequence (OOS) counter. We configure Flor running on the lossy queue, i.e., disabling

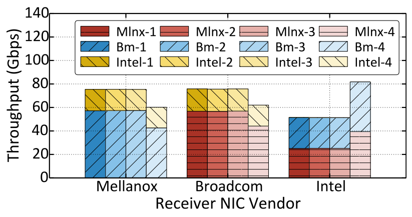


Figure 9: Bandwidth in a Mlnx-Broadcom-Intel hybrid deployment with Flor. Configurations: 1 = PFC+hardware CC+Flor, 2 = hardware CC+Flor, 3 = Flor, 4 = Flor +fixed cwnd.

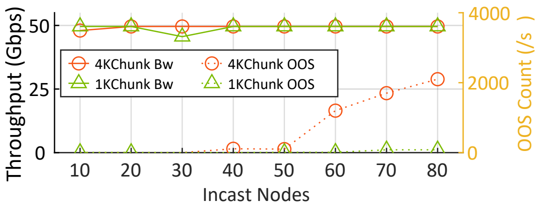


Figure 10: Flor performance in large-scale incast test.

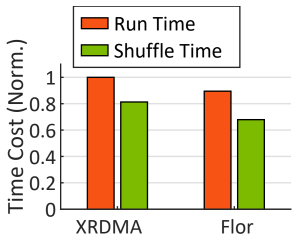


Figure 11: Run time in big data application.

PFC and DCQCN. For incast traffic, we install RPC clients on N machines and RPC server on one remote server. Each client has 8 threads connecting to the server, and each thread issues 32KB RPC requests with IO depth of 8. Thus, the number of QPs on each client is $8 \times 8 = 64$, and the number of QPs on the server is N $\times$ 64. Thus, the maximum incast degree is ~6000 when the node number is 90 in this test.

Figure 10 shows the throughput and OOS counters with different incast nodes of N and the minimal chunk sizes. Given the minimal chunk size of 4KB or 1KB, the throughput is consistent of 50Gbps with the increasing scale of incast. However, given the minimal chunk size of 4KB, the number of OOS increases when the incast nodes are larger than 50 (about 4000 : 1 incast). This is because the minimum chunk size of 4KB is the minimal congestion window size for each QP, and the volume of burst traffic is too large in such a large-scale incast. Therefore, we apply the minimal chunk size of 1KB, which avoids the generation of OOS in the large-scale incast. Notably, we set the minimal chunk size of 4KB in the storage production network because we have optimized the storage application to balance the load across nodes, which avoids such large incast events in practice [11].

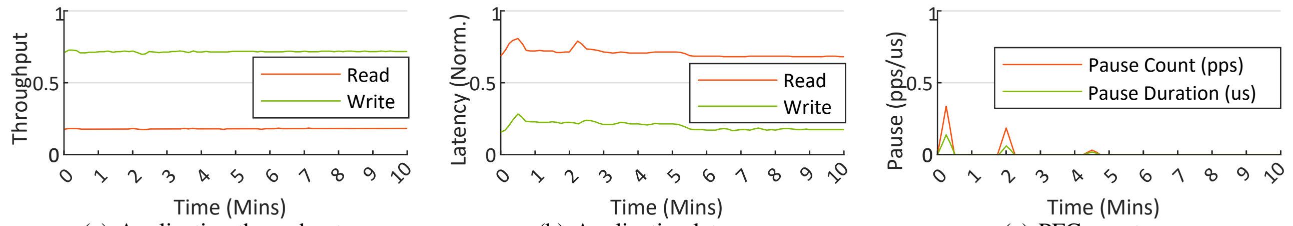
7.7 Evaluation in Production Network

Big-data applications. ServiceX is a Map-Reduce-like big-data application that runs on top of the distributed storage service. The completion time of the shuffle processing influences the performance of the whole task, and it desires fast and stable network transmission. To estimate the impact of Flor in the production system, we run ServiceX atop of Flor and XRDMA in cluster C (Table 2) with a lossless network. We conduct the task of sorting 1TB of data. The number of mapper tasks and reducer tasks are both 1K, and each mapper processes data of 1GB. We apply two key metrics: the average running time of a mapper and the average shuffle time, *i.e.*, the time of transferring data in the network. As shown in Figure 11, in comparison with XRDMA, Flor reduces the average running time by 10% and accelerates the average shuffle time by 16% due to an efficient congestion control strategy.

Non-stop upgrade. Flor allows to upgrade online with negligible down time of service, which is crucial to meet service level requirements in modern datacenter. We measure the impact of upgrading from XRDMA to Flor on applications such as Pangu [11] through the measurements of normalized throughput and latency and PFC counters. In the experiment, the software upgrading takes place at the 0.5^{th} minute. As shown in Figure 12(a), the read and write throughput of the application have a slight jitter (<2%) when switching to Flor. Figure 12(b) shows that the latency increases by 10% at 0.5 minute, lasting less than 30s. Figure 12(c) shows the generation of PFC pauses (packets per second, pps) and its duration time (μs), which appears in a very short time period. At the 5.5^{th} minute, we then disable DCQCN and PFC. The throughput is unaffected, and the latency decreases slightly (~3%). This latency might be caused by the interference between DCQCN and the software congestion control. When running Flor with and without PFC and DCQCN, all the metrics are healthy.

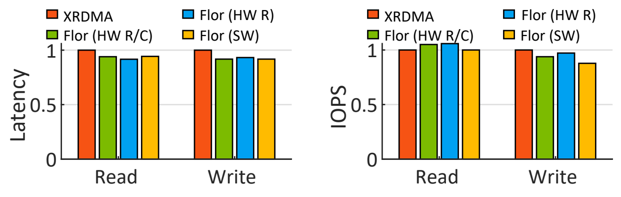
High-performance block storage service. Flor is applied for latency-sensitive applications such as the Enhanced SSD (ESSD) product of Elastic Block Storage (EBS). ESSD provides block storage service (virtual disks) as local devices through high performance network. We compare the latency and requests per second (IOPS) of an EBS application running with XRDMA or Flor in cluster C (Table 2). We adopt the workload of a real ESSD storage application with an I/O size of 4KB. XRDMA is tested with RoCE lossy accelerations enabled. Flor is tested with these features disabled. There are three kinds of configurations for Flor: (i) Flor with hardware reliability and hardware congestion control (Flor HW R/C); (ii) Flor with hardware reliability and software congestion control (Flor HW R); and (iii) Flor with software reliability and congestion control (Flor SW).

Figure 13 (a) shows the normalized single-operation latency of XRDMA and Flor. Flor demonstrates comparable average latency performance with XRDMA among all the operation types. Although software-based modules are involved, the latency of Flor is still slightly lower (1%–8%) than XRDMA due to the optimized software stack of Flor. Flor (HW R/C) has the lowest average latency through hardware-based implementation. Flor (SW) and Flor (HW R) have slightly higher latency (< 3%) due to the overhead of software stack. Figure 13 (b) shows that Flor achieves the comparable normalized IOPS as XRDMA for 4KB *Read* and *Write*. In conclusion, in supporting block storage service, Flor has comparable latency and IOPS with XRDMA.



(a) Application throughput. (b) Application latency. (c) PFC counters.

Figure 12: The system performance of upgrading from XRDMA to Flor.



(a) Average latency of 4KB. (b) IOPS of 4KB operation.

Figure 13: Performance of Block Storage Service.

Network Protocol	MTU (B)	Throughput 100/200 (Gbps)
Linux TCP	1500	22
SNAP	1500	39.1
SNAP (+I/OAT)	5000	67.5 (82.2)
eRPC on IB	3840	73
Perftest	1024	88/193
XRDMA	1024	88/174
Flor	1024	84/173

Table 3: The single-core bandwidth of different network transport stacks in 100 and 200Gbps networks.

8 Discussion

Hybrid RDMA NICs' deployment in datacenters. In the production network with over 100K servers, new servers and new NICs are incrementally deployed, which results in the mix of the latest generation and earlier generations (or different vendors) of RDMA NICs co-existing in the same datacenter network. In addition, the malfunctioning servers in the built-up clusters can be replaced by servers configured with NICs of different generations or vendors. The new generation of RNICs is released by the vendors every 2 or 3 years. From our experience in Block Storage Service, we first deployed storage servers with CX-4 RDMA NICs in 2016. In 2019, we started to deploy new servers of CX-5 RDMA NICs because of higher performance and price ratio. As a result, there exists the hybrid deployment of both CX-4 and CX-5 RDMA NICs. The same deployment choice was made when we introduced CX-6DX RDMA NICs in 2021.

Programmable control plane. Flor provides a new perspective of layered architecture to achieve high velocity and performance with diverse hardware. Flor envisions the "programmability" of reliability and CC modules of RDMA in the control plane, which can be implemented in programmable hardware, such as smartNICs with embedded CPU or NPU, e.g., ConnectX-6, Bluefield, Intel IPU. In this way, we can make use of the programmable hardware capability such as hardware timestamp, rate-limiting, and packet drop detection to further improve the efficiency of CC and reliability while keeping the unified control policy among heterogeneous RNICs with the same Flor architecture.

Concerns of implementation. Currently, not all NICs support UC. First, this research work has demonstrated, for the first time, the effectiveness of using UC for high-performance out-of-order transmission in product networks. Second, UC is a standard transport defined in the RoCE specification, and its logic is simple and easy to implement by hardware. Therefore, this work provides a new choice for the community, and we expect more vendors to support UC. In addition, when migrating Flor to SmartNICs, e.g., Bluefield 2, which has multiple ARM cores and sufficient DRAM memory, thanks to Flor 's architecture that clearly separates each component, each individual component is easy to move to the BlueField's ARM cores. The main differences from the host CPU implementation are that the ARM core is less performant than the host CPU core, and the L3 cache size of Bluefield 2 is limited, requiring the developers to optimize the system carefully. With the capability of directly operating data in the host memory introduced by Bluefield 3, the limited L3 cache size caused performance degradation can be resolved.

9 Related Work

Software solutions. We compare Flor with different network protocols. Table 3 shows the single-thread throughput between two nodes in the 100Gbps and 200Gbps network. The throughput of SNAP [33] and eRPC [20] are from the published papers. We can see that network protocols with hardware-offloaded RDMA semantics, *i.e.*, Perfest [13], XRDMA and Flor, achieve higher throughput than other network stacks. In addition, the throughput of XRDMA and Flor are comparable with Perfest, which plays the raw IB verbs without any overhead of RPC. The RDMA-based protocols can also maintain high bandwidth utilization in a 200Gbps network, where the throughput of Flor is the same with XRDMA. Though the throughput of eRPC and SNAP increases as the MTU size increases, using large MTU sizes requires a unified and standard configuration, which adds operation complexity in a complicated production environment. Besides, software solutions suffer from higher latency [33] and CPU overhead [20] without hardware

Functionalities	**Lossless RDMA**	**Lossy RDMA**	**eRPC**	**1RMA**	**Flor**
Transport granularity	Verbs	Verbs	MTU	Op (4KB)	Variable chunk (*e.g.*, 4KB-64KB)
CPU Involvement	One/Two-sided	One/Two-sided	Two-sided	One-sided	One/Two-sided
Congestion control & Signal & Type	HW, ECN, rate	programmable HW, ECN+RTT, rate	SW, credit, window	SW, RTT, window	SW/HW, RTT/ECN, window/rate
Reliability & Retransmission	HW, GBN	HW, SR for RDMA Operation	SW, GBN	SW, unknown	SW/HW, intra-chunk GB0 & inter-chunk SR/GBN
PFC dependency & Loss tolerance	Yes, poor	No, high	No, poor	No, unknown	No, high
HW dependency	All RNICs	CX6-dx	DPDK/all RNICs	Customized NIC	All RNICs
Datapath zero copy	Yes	Yes	No	Yes	Yes

Table 4: Comparison on transport features of Flor and other network solutions.

acceleration.

Hardware solutions. To get rid of PFC, Mellanox brings up Resilient RoCE [48] and Lossy RoCE Accelerations [53] on lossless RNICs, *i.e.*, Go-Back-N-based RNICs. Resilient RoCE utilizes congestion control, *i.e.*, DCQCN, to deal with network congestion and avoid packet loss. A recent study [44] shows that the Resilient RoCE can prevent packet loss in some specific scales but still suffers unfairness from packet loss in large-degree incast events. Hardware-based lossy RDMA solutions such as Mellanox CX-5/6 [50,52] and IRN [36] rely on strengthened hardware to run on a lossy network. They can not be deployed with CX-4 RNICs, and also lack the flexibility for users to customize each function as the implementation is highly ingrained into the hardware.

Hardware & software co-design solutions. RoGUE [28] designs a software congestion control for RDMA but relies on hardware reliability mechanism to recover from packet loss. It uses a large static chunk size, *i.e.*, 64KB, which needs to be revised to deal with the large-scale incast scenario. 1RMA [45] is a high-performance network system that provides congestion control and reliability in software. 1RMA also enables one-sided RDMA operations based on novel hardware with RDMA READ-like operation. However, it can not work on commodity RNICs, so it has little help for existing RDMA systems. Table 4 shows the clear difference between Flor and other network frameworks.

10 Conclusion

We present Flor, a flexible lossy RDMA framework for heterogeneous RNICs that solves a set of problems raised in production RoCEv2 clusters. These problems include PFC dependency, the interconnectivity of heterogeneous RNICs and hardware-bonded congestion control schemes. Flor onloads the reliability and congestion control function from RNICs to the software. Flor proposes a software selective retransmission for the first time at the RoCEv2 network and uses a software RTT-based congestion control to deal with the performance gap among the heterogeneous RNICs. Our evaluation of the testbed and production clusters shows that Flor achieves high performance and flexibility in many scenarios, including packet loss, heterogeneous hardware, large-scale incast, and distributed systems. Flor also shows that the process of upgrading the existing RDMA framework to Flor has little performance impact on the running applications.

Acknowledgments. We are extremely grateful for our shepherd, Costin Raiciu, and the anonymous OSDI'23 reviewers for their wonderful feedback. Xiaoliang Wang is supported by NSFC No.62172204. Qiao Xiang is supported in part by the National Key R&D Program of China 2022YFB2901502, Alibaba Innovative Research Award, NSFC No.62172345, Open Research Projects of Zhejiang Lab 2022QA0AB05, MOE China Award 2021FNA02008, and NSF-Fujian-China 2022J01004. And we also appreciate for the help from Lei Yan and Shanghai Yunsilicon Technology Co., Ltd.

References

[1] Introducing the gaudi2 processor for training deep learning workloads. https://habana.ai/training/gaudi2/, 2022.

[2] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). In *SIGCOMM*. ACM, 2011.

[3] InfiniBand Trade Association. Infiniband architecture specification release 1.2.1 annex a16: RoCE, 2010.

[4] InfiniBand Trade Association. Infiniband architecture specification release 1.2.1, 2014.

[5] InfiniBand Trade Association. Infiniband architecture specification release 1.2.1 annex a17: Rocev2, 2014.

[6] Broadcom. Changing congestion control mode settings. https://techdocs.broadcom.com/us/en/storage-and-ethernet-connectivity/ethernet-nic-controllers/bcm957xxx/adapters/Configuration-adapter/RoCE/advanced-network-configuration/changing-congestion-control-mode-settings.html, 2022.

[7] Broadcom. Netxtreme®-e series. https://www.broadcom.com/products/ethernet-connectivity/network-adapters/p2100g, 2022.

[8] Yanzhe Chen, Xingda Wei, Jiaxin Shi, Rong Chen, and Haibo Chen. Fast and general distributed transactions using RDMA and htm. In *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016.

[9] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. FaRM: Fast remote memory. In *NSDI*, 2014.

[10] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: SmartNICs in the public cloud. In *NSDI*, 2018.

[11] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When cloud storage meets RDMA. In *NSDI*, 2021.

[12] Dan Gibson, Hema Hariharan, Eric Lance, Moray McLaren, Behnam Montazeri, Arjun Singh, Stephen Wang, Hassan MG Wassel, Zhehua Wu, Sunghwan Yoo, et al. Aquila: A unified, low-latency fabric for datacenter networks. In *NSDI*, 2022.

[13] Github. Perftest. https://github.com/linux-rdma/perftest, 2021.

[14] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over commodity Ethernet at scale. In *SIGCOMM*. ACM, 2016.

[15] 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 *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, 2017.

[16] Alibaba Inc. Pangu, the high performance distributed file system by Alibaba cloud. https://www.alibabacloud.com/blog/pangu-the-high-performance-distributed-file-system-by-alibaba-cloud_594059, 2018.

[17] Intel. Production brief for Intel® Ethernet Controller E810-CAM2/CAM1/XXVAM2. https://cdrdv2.intel.com/v1/dl/getContent/615503, 2020.

[18] Intel. Intel infrastructure processing unit (IPU). https://www.intel.cn/content/www/cn/zh/products/network-io/smartnic.html, 2021.

[19] Intel. Irdma readme. https://downloadmirror.intel.com/738730/README_irdma.txt, 2022.

[20] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter RPCs can be general and fast. In *USENIX NSDI*, 2019.

[21] Anuj Kalia, Michael Kaminsky, and David G Andersen. Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 295–306. ACM, 2014.

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

[23] Anuj Kalia Michael Kaminsky and David G Andersen. Design guidelines for high performance RDMA systems. In *USENIX Annual Technical Conference (ATC)*, 2016.

[24] Antoine Kaufmann, Tim Stamler, Simon Peter, Naveen Kr Sharma, Arvind Krishnamurthy, and Thomas Anderson. TAS: TCP acceleration as an OS service. In *Proceedings of the Fourteenth EuroSys Conference*, 2019.

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

[26] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. Swift: Delay is simple and effective for congestion control in the datacenter. SIGCOMM, 2020.

[27] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC transport protocol: Design and internet-scale deployment. In *SIGCOMM*, ACM, 2017.

[28] Yanfang Le, Brent Stephens, Arjun Singhvi, Aditya Akella, and Michael M Swift. Rogue: RDMA over generic unconverged Ethernet. In *SoCC*, 2018.

[29] Hao Li, Asim Kadav, Erik Kruus, and Cristian Ungureanu. Malt: distributed data-parallelism for existing ml applications. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015.

[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 *SIGCOMM*. ACM, 2019.

[31] James M. Lucas and Michael S. Saccucci. Exponentially weighted moving average control schemes: Properties and enhancements. *Technometrics*, 32(1):1–12, 1990.

[32] Teng Ma, Tao Ma, Zhuo Song, Jingxuan Li, Huaixin Chang, Kang Chen, Hai Jiang, and Yongwei Wu. X-RDMA: Effective RDMA middleware in large-scale production environments. In *IEEE International Conference on Cluster Computing (CLUSTER)*, 2019.

[33] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. Snap: A microkernel approach to host networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP, 2019.

[34] Rui Miao, Lingjun Zhu, Shu Ma, Kun Qian, Shujun Zhuang, Bo Li, Shuguang Cheng, Jiaqi Gao, Yan Zhuang, Pengcheng Zhang, Rong Liu, Chao Shi, Binzhang Fu, Jiaji Zhu, Jiesheng Wu, Dennis Cai, and Hongqiang Harry Liu. From Luna to Solar: The evolutions of the compute-to-storage networks in Alibaba cloud. In *SIGCOMM*, New York, NY, USA, 2022. Association for Computing Machinery.

[35] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store. In *USENIX Annual Technical Conference (ATC)*, 2013.

[36] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for RDMA. In *Proceedings of ACM Special Interest Group on Data Communication*. ACM, 2018.

[37] Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Prateesh Goyal, Srinivas Narayana, Radhika Mittal, Mohammad Alizadeh, and Hari Balakrishnan. Restructuring endpoint congestion control. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018.

[38] Wiki of Linux perf command manpage. Perf wiki. `https://perf.wiki.kernel.org/index.php/Main_Page`, 2021.

[39] OpenCompute. In-band network telemetry in Barefoot Tofino. `https://www.opencompute.org/files/INT-In-Band-Network-Telemetry-A-PowerfulAnalytics-Framework-for-your-Data-Center-OCP-Final3.pdf`, 2019.

[40] P4. P4. `https://p4.org/`, 2021.

[41] Behnam Montazeri Masoud Moshref Khaled Elmeleegy Luigi Rizzo Marc de Kruijf Gautam Kumar Sylvia Ratnasamy David Culler Amin Vahdat Saksham Agarwal, Rachit Agarwal. Understanding host interconnect congestion. In *in ACM HotNets*. ACM, 2022.

[42] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. Supercomputing on Nitro in AWS cloud. *IEEE Micro*, 2020.

[43] Jiaxin Shi, Youyang Yao, Rong Chen, Haibo Chen, and Feifei Li. Fast and concurrent rdf queries with RDMA-based distributed graph exploration. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2016.

[44] Alexander Shpiner, Eitan Zahavi, Omar Dahley, Aviv Barnea, Rotem Damsker, Gennady Yekelis, Michael Zus, Eitan Kuta, and Dean Baram. RoCE rocks without PFC: Detailed evaluation. KBNets, 2017.

[45] 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 *SIGCOMM*, 2020.

[46] Stanford. Homasimulation. `https://github.com/PlatformLab/HomaSimulation`, 2018.

[47] Maomeng Su, Mingxing Zhang, Kang Chen, Zhenyu Guo, and Yongwei Wu. RFP: When RPC is faster than server-bypass with RDMA. In *Proceedings of the Twelfth European Conference on Computer Systems*, 2017.

[48] NVIDIA Networking (Mellanox Technologies). Resilient roce. `https://community.mellanox.com/s/article/introduction-to-resilient-roce---faq`, 2018.

[49] NVIDIA Networking (Mellanox Technologies). Connectx-4 lx en en card. `https://www.mellanox.com/files/doc-2020/pb-connectx-4-lx-en-card.pdf`, 2020.

[50] NVIDIA Networking (Mellanox Technologies). Connectx-5 en en card. `https://www.mellanox.com/files/doc-2020/pb-connectx-5-en-card.pdf`, 2020.

[51] NVIDIA Networking (Mellanox Technologies). Accelerating data center security with bluefield-2 dpu. https://developer.nvidia.com/blog/accelerating-data-center-security-with-bluefield-2-dpu, 2021.

[52] NVIDIA Networking (Mellanox Technologies). ConnectX-6 Dx Ethernet SmartNIC. https://nvdam.widen.net/s/qpszhmhpzt/networking-overal-dpu-datasheet-connectx-6-dx-smartnic-1991450, 2021.

[53] NVIDIA Networking (Mellanox Technologies). Mellanox lossy RoCE accelerations. https://community.mellanox.com/s/article/How-to-Enable-Disable-Lossy-RoCE-Accelerations, 2021.

[54] C. Tian, B. Li, L. Qin, J. Zheng, J. Yang, W. Wang, G. Chen, and W. Dou. P-PFC: Reducing tail latency with predictive PFC in lossless data center networks. *IEEE Transactions on Parallel and Distributed Systems*, 31(6):1447–1459, 2020.

[55] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. Fast in-memory transaction processing using RDMA and htm. In *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015.

[56] Jian Yang, Joseph Izraelevitz, and Steven Swanson. Orion: A distributed file system for non-volatile main memories and RDMA-capable networks. In *17th USENIX Conference on File and Storage Technologies (FAST)*. USENIX Association, 2019.

[57] 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 *SIGCOMM*, 2015.

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

APPENDIX

A Design Details

A.1 Software Reliability

A.1.1 Chunk sequence number

Flor has two sequence number spaces, *i.e.* global sequence number (GN in Figure 14) and reliable sequence number (RN in Figure 14). The global sequence number is a 64-bit value and all the RDMA WQEs have a unique global sequence number to identify the sequence within a QP. The reliable

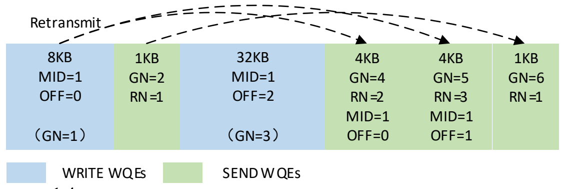


Figure 14: An example of the numbering system of Flor. An $40KB$ of *WRITE* WQE is splited into a $8KB$ and a $32KB$ WRITE WQE. The $8KB$ *WRITE* WQE and $1KB$ *SEND* WQE are lost and get retransmitted. The retransmissions of $8KB$ *WRITE* WQE are transmitted via two $4KB$ *SEND*s.

sequence number is used to identify the sequence within the same type of WQEs, i.e., *WRITE* WQEs and *SEND* WQEs have separate reliable sequence number space. Flor identifies the original WQEs and the retransmitted WQEs with different global numbers such that ACK information (and timestamp information carried in ACKs) is not ambiguous. The global sequence numbers for *WRITE* WQEs are maintained only at the senders and not transmitted to the receiver. The *SEND* WQEs carry both the reliable numbers and the global sequence numbers to the receiver. The retransmission WQEs share the same reliable sequence number with the original WQE. Note that Flor uses *SEND* WQE to retransmit *WRITE* WQE. This *SEND* WQE that is used for *WRITE* retransmission carries the original *WRITE* reliable sequence number, a new reliable number and a new global sequence number to the receiver such that this retransmission is able to be identified both by the sender and receiver.

The reliable sequence number of *WRITE* WQEs consists of 1-bit hint, 21-bit message id (MID in Figure 14) and 10-bit $chunk\_offset$ (OFF in Figure 14). The hint bit is set when the WQE is the last WQE in the congestion window. We align the $chunk\_size$ to the chunk unit size (*e.g.*, UNIT_SIZE). The $chunk\_offset$ represents the offset of the memory address of a chunk ($addr_c$) from the staring memory address of the same message ($addr_m$), *i.e.*,

$$chunk\_offset = (addr_c - addr_m)/UNIT\_SIZE$$

The receiver can validate the integrity of the message by checking whether it has received data of all chunk offsets covered the message size and assemble the messages to notify the application. If using $UNIT\_SIZE = 4KB$, then 10-bit chunk offset supports up to $4KB \times 2^{10} = 4MB$ message. To support larger message in applications, users can allocate more bits for chunk offset field.

Figure 14 shows an example how this numbering system works. Here a large message of 40KB is split into 2 *WRITE* WQEs, i.e., 8KB and 32KB and a *SEND* message is sent between these two *WRITE* WQEs. Each WQE has a unique global sequence number (GN) and the *WRITE* WQEs do not carry the GN to the receiver while the **SEND** does.

In the case that 8KB *WRITE* WQE and the 1KB *SEND* WQE are retransmitted. The 8KB **WRITE** WQE is split into two 4KB *SEND* WQEs, where the minimal $chunk\_size$ is

4KB. Each *SEND* WQE for *WRITE* retransmission carries the original reliable sequence number of the *WRITE* WQE and has a new *SEND* reliable sequence number and a new global sequence number. Each *SEND* WQE for *SEND* retransmission carries the original *SEND* reliable sequence number and a new global sequence number to the receiver.

A.1.2 ACK Format and Compression

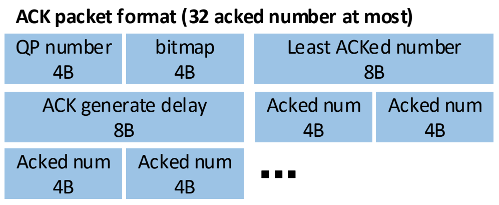


Figure 15: Software ACK format.

Figure 15 shows the software ACK format. A 32-bit bitmap (4B) in ACK packet indicates that each ACK packet signals at most 32 WQEs. A i^{th} bit set in the bitmap indicates that the i^{th} ACKed number is a global sequence number for a **SEND** WQE, otherwise, the i^{th} ACKed number is the reliable sequence number for a **WRITE** WQE. It is possible that the ACK packet contains the number of ACKed number is less than 32. As Figure 15 shows that the first ACKed number starts from 24^{th}B and each ACKed number is 4B. The number of ACKs carried in one packet is calculated as follows:

$$(ack\_length - 24B)/4B,$$

where *ack_length* is the packet length of the ACK packet. For example, an ACK of packet length 40B carries $(40B - 24B)/4B = 4$ acked numbers. If the bitmap is $0XC0000000$, then the first 2 ACKed numbers acknowledge **WRITE** WQEs and the 3^{th} and 4^{th} ACKed numbers acknowledge **SEND** WQEs.

We limits the acked number to be 32-bit to shorten the length of the ACK packets. The reliable sequence number of a **WRITE** WQE and the global sequence number of a **SEND** WQE will be ACKed back to the sender. Recall that the reliable sequence number is 32-bit and the global sequence number is 64-bit. Thus, we compress the 64-bit global sequence number to a 32-bit ACKed number as follows:

$$acked\_num = global\_num - least\_acked\_global\_num,$$

where the *least_acked_global_num* is the smallest global sequence number in a ACK packet. The WQEs with global sequence number larger than

$$least\_acked\_global\_num + 2^{32} - 1$$

are dropped by Flor if received. This window size is large enough in practice. The silently dropped WQEs, if there are, are detected by timeout.

A.2 RTT measurement

A.2.1 HW/SW Clock Synchronization

The timestamps are generated by the NIC hardware clock. Except from obtaining timestamps from completions events to calculate, Flor may also need time for other usages, *e.g.*, setting retransmission timers. However, querying current time from RNICs is a time-consuming operation (*e.g.*, costs 1μs in CX-4). Thus Flor maintains a software clock based on *rdtsc()* and synchronizes the clock with hardware clock. When Flor sends or receives an operation and the clock is not corrected for 100μs, then Flor queries a timestamp from RNIC and update the offset when the error exceeds threshold 10μs. According to our observation, the successfully correct ratio (*i.e.*, the ratio that the error exceeds 10μs) is less than 1%.

A.2.2 Improve RTT Measurement Accuracy

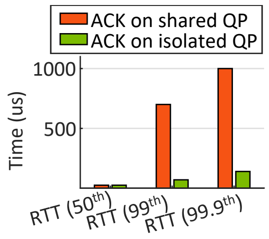


Figure 16: RTT accuracy with shared and isolated QP.

Figure 16 shows the measured RTT values with and without Flor optimization, *i.e.*, *ACK on isolated QP* and *ACK on shared QP*, respectively. The experiment setup is the same as Figure 5 except using a larger RPC request size, *i.e.*, 1MB.

A.2.3 RTT Measurement for UC

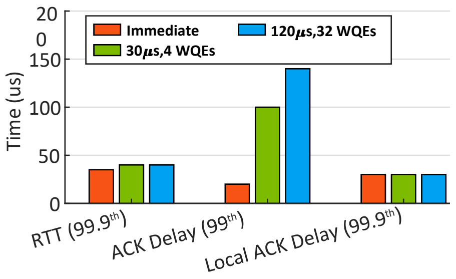


Figure 17: Evaluation of optimizations on ACK designs.

Note that Flor ACKs coalescing mechanism can also cause the ACKs being delayed. Thus, we measured the ACK delay (*i.e.*, $T'_3 - T_2$) and local delay duration (the time between T_4 and ACK processing time) with different acknowledgement frequencies. Figure 17 reports the 99^{th} percentile of these delays as the acknowledgement frequencies changes. As expected, the ACK delay increases as the number of WQEs' ACKs coalescing increases, this is because the receiver needs to wait more WQEs to finish or a timer to generate an ACK. The local delay stays the same because Flor prioritizes to poll the ACK completion queue and Flor processes one

ACK regardless the number of the number of WQEs' ACKs coalescing. Finally, the RTT measurement results show that RTT measurement accuracy does not impact by the ACK frequencies with this improvement.