Bifröst: A Highly Compatible and Flexible PFC-free RDMA Framework

Paper #515, 12 pages + references + appendix

Abstract

Remote Direct Memory Access (RDMA) has been widely deployed in datacenters to provide low latency and high throughput for applications. There are still many practical problems in production RDMA networks: PFC dependency, interconnectivity of heterogeneous RNICs, and hardware-binded congestion control. Our key observation for the root cause of these problems is that RDMA implementation is highly ingrained into hardware. We present Bifröst, a highly compatible and flexible PFC-free RDMA framework working with legacy RDMA clusters. The key idea of Bifröst is separating the control path and data path of RDMA protocol by a hardware and software co-design. Bifröst completely eliminates PFC risks by working well on lossy Ethernet and provides a non-stop and smooth transition from the lossless RDMA to the PFC-free Bifröst. Our evaluations on the testbed and production clusters show that Bifröst achieves high performance and flexibility in many scenarios, including packet loss, heterogeneous hardware, large-scale incast, cross-pod communication, and distributed systems.

1 Introduction

Remote Direct Memory Access (RDMA) has been widely deployed in datacenters [15,20,34] in the form of *IP ROUTABLE RDMA over Converged Ethernet* (RoCEv2) [5,7] to provide low latency and high throughput for many applications, *e.g.*, RPC systems [25,27,44], key-value store [26,36], distributed transactions [10,50], distributed memory [14,51] and storage systems [15], graph computing [42] and machine-learning systems [33]. The wide coverage of RDMA-based systems manifests the popularity of RDMA in datacenter networks.

However, production deployment of RoCEv2 network still experiences several challenges. First, the PFC storm [20] has the potential risk of turning down the whole network, thus some production deployments limit the PFC to be enabled within a pod [15]. This results in poor performance for the across-pod large-scale applications. Second, different generations of commodity RDMA NICs (RNICs) (*e.g.*, Mellanox ConnectX-4/5/6 [45–47]) and customized RNICs [3, 41] coexist in the same cluster as the datacenters are constantly expanded and newly built. We observed that in a full-mesh traffic pattern with perftest [18], the throughput of dual-port 25Gbps CX-4/5 is 28Gbps/41Gbps respectively. The imbalance of network throughput introduces the computation load imbalance and stragglers in the distributed systems. Third, different applications, *e.g.*, distributed training systems [13,24], storage systems [4,15], disaggregated systems [2,19], hosted in the datacenter show different network traffic patterns, thus requires different parameters or more efficient congestion control algorithms. However, the widely deployed RNICs build the congestion control in the hardware. RDMA clusters are limited to unitary congestion control algorithm (i.e., DCQCN [20]) and loses the flexibility of adopting emerging congestion control schemes, e.g., HPCC [34] and Swift [30].

Our key observation for the essence of the problems is that RDMA implementation is highly ingrained into hardware, which results in the lack of flexibility and customizability. The full set of transport functions of RDMA, *e.g.*, congestion control, reliability, packet processing, are implemented in RNICs. This inspires us to *rethink the labor division of the RoCEv2 transport between hardware and software*, i.e., a subset of functions are onloaded to the software. Due to this *hardware lock-in* property in the commodity RNICs, software developers have little access to the transport layer, which limits the design space for the function onloading. This requires software developers to exhaust all the programmability features of the commodity RNICs to explore the possible onloading functions. In addition, function onloading provides flexibility and programmability but with the cost of CPU and even performance. Thus, the key challenge is to release the *hardware lock-in* while retain a comparable performance as vanilla RDMA within the limited design space.

We present Bifröst, a highly compatible and flexible PFC-free RDMA framework working with legacy RDMA clusters. The key idea of Bifröst is *separating the control path and data path of RDMA transport with a hardware and software co-design* [29,32,39,43,47]. The data path (RDMA transport semantic) still sits in the hardware while the control path

functionalities, *i.e.*, congestion control algorithm, reliability, are onloaded into software. With Bifröst, we also modularize control path functionalities such that different applications have the flexibility of making a different combinations of these modules and customizing them.

Bifröst includes a load-aware dynamic chunking module, an RDMA-semantic-compatible reliability module, and a congestion control module. These software modules can be deployed without hardware modifications and across different generations of RNICs. The load-aware dynamic chunking module achieves a good balance between performance and software control granularity. Bifröst leverages RDMA primitives on Reliable Connection (RC) transport or Unreliable Connection (UC) transport [6] to transfer data and designs a new software reliability mechanism for UC to ensure reliable delivery. The unique *out-of-order delivery* feature of UC provides an opportunity of designing a selective retransmission scheme. Bifröst implements an RTT-based congestion control algorithm similar to Swift [30] and improves RTT measurement accuracy by $\sim 10\times$ than previous work [32] on 99^{th} percentile and 99.9^{th} percentile RTT.

We run extensive experiments on an RPC benchmark and real production systems to evaluate Bifröst in different clusters. We mainly compare Bifröst against our customized last generation RDMA library (ARDMA). Our evaluations show that the software congestion control algorithm shrinks the bandwidth gap from 21.8% to 1.3% in the hybrid CX-4 and CX-5 deployment cluster. Compared to vanilla RDMA that suffers significant performance loss with 1/4096 packet loss ratio, Bifröst achieves steadily high throughput with this or even higher packet loss ratio. With a Swift [30]-variant congestion control algorithm, Bifröst can handle 6000:1 incast without throughput loss. Bifröst also shows comparable performance as ARDMA for intra-pod communication and better performance than ARDMA for inter-pod communication.

Our upgrading experience with Bifröst shows that the upgrading is smooth and has little effect to the running applications. Finally, we collect some performance counters from the production systems, i.e., a big-data application and a distributed storage system, running on top of Bifröst. Compared with ARDMA, Bifröst achieves comparable performance with lossy configuration and better performance with lossless configuration.

2 Background & Motivation

2.1 RDMA Preliminaries

RDMA transport services. Table 1 shows the main RDMA transport services and primitives (called *verbs*, *i.e.*, *SEND*/*WRITE*/*READ*, *etc.*). RDMA defines 4 transport services: Reliable Connection (RC), Unreliable Connection (UC), Reliable Datagram (RD) and Unreliable Datagram (UD) [6, 26].

Each transport service (except RD [6]) is implemented on corresponding communication channels (Queue Pairs, QPs). Users initiate data send/receive requests to RNICs by posting Work Queue Elements (WQEs) into QPs. After transmitting the data, the RNICs generate Completion Queue Elements (CQEs) into Completion Queues (CQs) as the transmit completion signals for users.

RDMA Verbs	RC	UC	RD	UD
SEND	✓	✓	< MTU	< MTU
WRITE	✓	✓	×	×
READ	✓	×	×	×

Table 1: RDMA transport services.

RDMA reliability mechanisms. 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. Once a packet is dropped, the sender has to restart the transmission from that packet. 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. For RC, a send CQE means that the packets of the verb has been delivered to the receiver and *acknowledged*, while for UC it means the packets have been *transmitted* to the network.

2.2 Production Experience

2.2.1 The Sword of Damocles: PFC storm

PFC storm is a well-known problem [20, 21, 49, 52] that seriously threats system's availability since it can spread to the whole cluster [20].

Unpredictable PFC storms. In addition to the known causes of PFC storms (*i.e.*, the *slow receiver* reported by [20]) and switch hardware bug [15]), we also found that (1) Machine Check Errors (MCE) caused by memory Error Checking and Correcting (ECC) errors can also lead to PFC storms. When memory ECC errors occur in a server, its RNIC receives data but can not transfer it to the memory of server. Thus, it sends excessive PFC frames to the network. The pause frames propagate to neighbor switches and stop them, which again repeat the spreading chain. As a result, a single point failure tears down the whole network. (2) The lack of memory bandwidth also leads to PFC storms. CPUs and RNICs share the memory bandwidth on a server. When the CPU running applications preempt too much memory bandwidth, the memory bandwidth left for the RNIC is less than network bandwidth. Then the RNICs send PFC frames to prevent packet loss due to the RNIC buffer overflow. In a nutshell, it is very hard to guard against every possibility of PFC storm.

Operation complexity. Production systems [15, 20] build up mechanisms such as monitoring and watchdogs to deal with PFC storms. However, the mass of PFC pause frames also frequently trigger alerts provided by the PFC monitoring system in daily operation, which are annoying and distract operation engineers from real PFC risks.

Cross-pod deployment. To isolate the failure domain, some production practices [15] limit PFC propagation within a small scale, i.e., turn on the PFC within a pod. However, some data center applications, *e.g.*, large-scale machine learning systems and distributed storage systems, ask for network communications across pods. In such case, the system replying on the lossless network experiences low performance due to the packet drops.

Thus, we call for a solution that can efficiently *prevent packet drop* and *recover packets loss* to *disable PFC* in the production system.

2.2.2 Interconnectivity of Heterogeneous RNICs

The interconnectivity problem comes from the coexistence of RNICs of various generations and brands in the same datacenter. In expanding datacenters, servers are purchased and deployed batch by batch. However, the lifetime of servers is about 4-5 years while new RNICs are usually launched every 1-2 years. This time gap means that servers with new generation of RNICs are continuously deployed while the servers with old RNICs are still serving. Besides, all commercial companies are trying to avoid relying on RNICs of a certain vendor. Some companies may design and deploy their own specific RNICs at the same time [43]. This also contributes to the mixture of heterogeneous RNICs in datacenters.

There are many challenges in connecting heterogeneous RNICs when performing RDMA. Different versions of RNICs have different hardware implementations and configurations. For example, Mellanox CX-4 RNICs must run on lossless Ethernet with PFC enabled on both switches and RNICs, while CX-5 and CX-6 RNICs do not have this requirement. There is a dilemma for choosing lossless or lossy configuration when connecting them, especially they are from two existing clusters with different configurations. Choosing backward-compatible lossless configuration brings PFC risks back to CX-5 clusters while choosing lossy configuration disables traffic of CX-4 RNICs. In a cluster with hybrid deployment of CX-4 and CX-5, even if choosing lossless configuration, there is still 21.8% bandwidth gap between them because of these implementation differences (§4.5). The functionality gap also exists among CX-4, CX-5 and CX-6 RNICs. CX-4 and CX-5 RNICs support Go-back-N (GBN) as the reliability in hardware, while CX-6 RNICs support selective retransmission [47] as the reliability scheme.

The distributed systems running on top of the network stack usually take the network performance as one of the factors for loading balancing. The network performance gaps lead to the computing tasks load imbalance on the nodes. Thus, we call for a solution that can have *a uniform and efficient congestion control algorithm and reliability scheme deployed on different versions of RNICs*.

2.2.3 Hardware-binded Congestion Control

Application diversification is a new challenge for datacenter network. For example, the distributed storage systems (*e.g.*, AWS [4] in Amazon and Pangu in Alibaba [15]) serve many applications such as BigData, EBS (Elastic Block Service), Database service and so on. In the face of discrepant performance requirements, diverse hardware deployment, and quite different traffic patterns, a jack-of-all-trades congestion control scheme can not satisfy various data-center applications. For example, big data applications such as Map-Reduce [12] and Hadoop [1] hope to minimize flow completion time in the shuffle pattern. Latency-sensitive applications such as key-value store [26, 36] and coordination service [9, 22] are eager for extremely low latency of each single request. Congestion control schemes are usually designed and optimized for specific scenarios.

However, current RNICs design congestion control algorithm as a fixed function in the hardware. As a result, DCQCN is the only congestion control algorithm supported in RNICs. DCQCN does not work ideally and need refined parameter tuning in many scenarios, *e.g.*, in some asymmetric topology (fail-over case in dual-ToR topology in [15]) and large-scale incast [16]. Designing the congestion control as a fixed hardware function impedes the opportunities of adopting more advanced congestion control algorithms, e.g., HPCC [34] and Swift [30]. Thus, we call for a solution that enables the developers to *customize congestion control algorithms* based on their knowledge on their workloads.

2.3 Rethink RoCEv2 Protocol Offloading

These practical issues prompt us to rethink the labor division of RoCEv2 protocol between CPUs and RNICs. Instead of the offloading the full set of functions, i.e., transport, reliability, congestion control, to the RNICs, a subset of these functions can be onloaded to the software such that the developers can customize these functions. Onloading to software brings flexibility and customizability but incurs performance overhead, i.e., high CPU cost and low throughput. The key challenge for Bifröst is to offer these flexibility while provide the comparable performance as plain RoCEv2 stack. In addition, the *hardware lock-in* constraint of the commodity RNICs that Bifröst targets at limits the design space for onloading. This requires us to exploit the hardware knobs to release the *hardware lock-in* constraint.

Inspired by the recent proposals [29, 35, 43], we realize that RoCEv2 transport can be separated into a data path and a control path. Bifröst implements the data path with RDMA verbs, where packetization, packet processing, direct memory access between RNICs and host memory sit in the hardware. Bifröst extracts the congestion control, reliability and other functions as individual modules and implements them in the software[1]. Datacenter operators can selectively

[1]We are currently working with a NIC vendor on the new type of NICs based on Bifröst's labor division model.

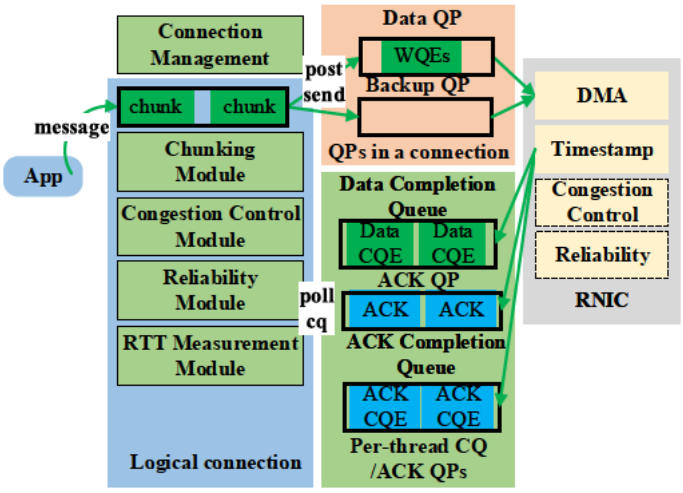


Figure 1: The framework of Bifröst.

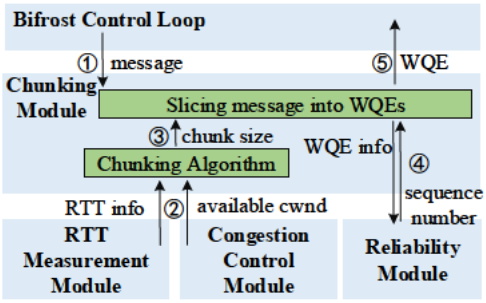


Figure 2: The workflow of Bifröst when transmitting data.

choose a Bifröst module to plug in the application based on the hardware environment. They can even customize their new design of the reliability and congestion control algorithms.

3 Design

3.1 Overview

Bifröst is a *PFC-free*, *minimally hardware dependent*, *highly flexible*, and *high performance* user-space RDMA library. The key idea of Bifröst is *separating the control and data path of RDMA protocol with a hardware and software co-design*. Figure 1 shows the framework of Bifröst. Bifröst implements the data path with UC and RC transport to achieve RNIC-offloaded RDMA semantic. To ensure reliability on UC-based datapath, Bifröst adopts software reliability mechanism at the granularity of RDMA requests. For RC-based datapath, Bifröst can leverage the RNIC-built-in reliability mechanism to guarantee the reliability.

On the control path, Bifröst builds five customizable modules that responsible for different aspects of transport control: *Connection Management*, *Chunking*, *Reliability*, *Congestion Control*, and *RTT Measurement*. The jobs of *Connection Management* module include establishing and tearing down connections, and backup QP management. The *Chunking* module splits large messages into small RDMA requests, *i.e.*, WQEs, based on a chunking algorithm. The

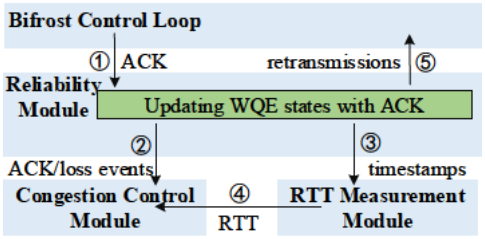


Figure 3: The workflow of processing an ACK in Bifröst.

Reliability module implements a reliability mechanism for UC, including an ACK submodule that sends and processes ACKs, a numbering system that manages sequence numbers for WQEs, and the retransmission mechanism (*e.g.*, Selective Retransmission). The *RTT Measurement* module collects hardware timestamp and updates RTT, which are used as network congestion signal and for setting the timeout. The *Congestion Control* module takes the congestion signals (*e.g.*, RTT, ECN, INT [40], and drop events) to calculate congestion window or sending rate according to the congestion control algorithms.

The flexibility of Bifröst for users is reflected in two aspects: (1) users can customize their own implementation for some modules and (2) users can disable some modules with enabling corresponding hardware functionalities. Table 3 shows the recommended combinations of modules according to different application scenarios (*e.g.*, deployment scale, requirements, and traffic patterns) and deployment restrictions (*e.g.*, host/switch hardware and configurations). For a distributed system running on a small scale with CX-4 RNICs, e.g., rack scale, turning on PFC and relying on hardware reliability and congestion control is a good choice. For a distributed system running across-pod with CX-4 and CX-5 RNICs hybrid deployment, we recommend software reliability and software congestion control to endure the packet loss.

3.2 Bifröst Workflow

Bifröst provides a *message-based* communication interface to support RPC calls, which most datacenter applications favor. Bifröst uses the RDMA **SEND** verb to transfer small messages (*e.g.*, ≤32KB) and **WRITE** to transmit large messages because **WRITE** verbs require remote memory address and message size exchanges before transmitting the data chunk, which incurs extra round trip time for small messages. To avoid the head-of-line blocking, Bifröst prioritizes the small messages (i.e., **SEND** WQEs) to the chunks of large messages (i.e., **WRITE** WQEs).

Here we demonstrate the Bifröst workflow with software reliability and congestion control modules.

Connection setup. When an application setups a connection, Bifröst creates a connection object through the *Connection Management* module and initializes an instance of each module. The *Connection Management* module maintains a logic connection, which includes a primary data QP and some backup data QPs as shown in Figure 1. Besides, each thread

has a UD QP for sending and receiving software ACKs.

Data transmit flow. Figure 2 shows the data transmitting workflow in the Bifröst main control loop. Bifröst passes the message to the *Chunking* module (① in Figure 2). The chunking algorithm in the chunking module takes the available congestion window from *Congestion Control* module and measured RTT from *RTT Measurement* module to calculate a chunk size (②, ③ in Figure 2). Then, the *Chunking* module splits a WQE of the chunk size from the message and passes it to the *Reliability* module. The *Reliability* module stores the WQE information and maintains its state, which is used later for acknowledgment/loss detection and RTT calculation. It then allocates the sequence numbers (§3.4.2), and encodes the numbers into the WQEs (④ in Figure 2), which returned to the Bifröst finally. Bifröst posts the WQE to the RNIC, which converts the WQE to a set of packets and transmits to the network (⑤ in Figure 2). Bifröst loops ②-⑤ until there is not available congestion window and uses ACKs to clock out new data.

Once the working thread is idle, Bifröst starts to poll the completions for the send and receive events for data QPs. If software reliability module is configured, Bifröst also polls to get software ACKs for reliability and congestion control (§3.4.3).

Data transmit completion event. When Bifröst gets a completion event of data transmission, it passes the timestamp of the WQE to the *Reliability* module. In UC transport, this data transmission completion indicates that the packets in the WQE are sent to the network. While it indicates that the WQE is reliably delivered to the receiver and ACKed in the RC transport. The *Reliability* module stores this timestamp for later RTT measurement.

Data receive flow. When Bifröst polls a data receive completion event, it passes the control information in the completion event (hardware timestamp and sequence numbers) to the *Reliability* module. The *Reliability* module detects packet loss and sends the ACKs if necessary. If the *Reliability* module verifies that all chunks of a message is received, it informs the Bifröst main control loop to submit the message to applications.

ACK process flow. When Bifröst gets a software ACK, it passes the ACK to the *Reliability* module (① in Figure 3). The *Reliability* module updates the states of inflight WQEs (ACKed or lossed) with received acknowledged chunk sequence number and timestamps. Acknowledged or loss events are passed to *Congestion Control* module to update the congestion window (② in Figure 3). Timestamps of ACKed WQEs are passed to *Reliability* module for RTT calculation (③ in Figure 3). The *RTT* measurement then updates the new RTT to the *Congestion Control* module (④ in Figure 3). Retransmissions of lost WQEs are passed into the retransmission queue of Bifröst main control loop (⑤ in Figure 3). The retransmission queue is always checked and transmitted before sending new data in each control loop.

Network Protocol	MTU (Bytes)	Throughput (Gbps)
Linux TCP	1500	22
SNAP	1500	39.1
SNAP (+I/OAT)	5000	67.5 (82.2)
eRPC on IB	3840	73
ARDMA	1024	88
Bifröst	1024	84

Table 2: The single-core bandwidth of different network transport stack in 100Gbps network.

Scenarios	Chunking	Reliability	C.C.
Intra-pod, PFC-enabled	N	HW	HW
Intra-pod, PFC/ECN-disabled	Y	HW/SW	SW
Cross-pod Apps	Y	SW	SW
CX-4/5 Hybrid	Y	HW/SW	SW

Table 3: Recommended choices of modules in some scenarios.

RC process flow. The RC process flow is a bit different from software reliability procedure since the hardware has guaranteed the reliability. Thus the ②-④ in Figure 3 are moved into the data transmit completion processing flow.

3.3 Data Path

The key feature of the I/O path in RDMA is the hardware-offloaded network transport stack and directly accessing remote memory (*i.e.*, **READ** and **WRITE**) on remote hosts through local and remote RNICs. To obtain high performance and compatibility, Bifröst leverages the RDMA semantics with RC and UC transport.

Compared to packet-IO transport (*e.g.*, TCP, UDP, DPDK and RDMA UD), RDMA semantics brings two major performance benefits. First, RDMA semantics minimizes CPU consumption by bypassing kernel, eliminating data copy, and offloading the network transport stack to RNICs. It keeps CPU free from protocol-related packing/unpacking processing and data movement. Secondly, RDMA semantics also reduces the memory bandwidth consumption for network I/O by eliminating data copy. The memory bandwidth is a bottleneck in some data-intensive systems with high-performance IO device [15].

To demonstrate the performance strength of RDMA semantic, we show the single-thread throughput of different network protocols under 100Gbps network in Table 2. The bandwidth number of SNAP [35] and eRPC [25] are read from the published papers and we measured throughput for Linux TCP with an RPC benchmark. Table 2 shows that network protocols with RDMA semantics (i.e., ARDMA and Bifröst) achieve higher throughput than other network stacks. Though the throughput is increased for eRPC and SNAP by increasing the MUT size, using large MTU sizes (*e.g.*, 4KB) requires a unified and standard configuration, which is difficult in a complicated production environment.

RDMA UC transport offloads the I/O segmentation to the RNICs but does not provide *reliability*. We observed that RNICs drop the whole WQE when one packet within the WQE is dropped, but the WQEs arriving later without any drops are delivered to the remote memory. Bifröst utilizes this feature to implement a selective retransmission for the

UC transport. Note that an RC QP drops all the later arriving packets when it observed a packet drop, which limits the design space for reliability mechanism.

3.4 Control Path

In this subsection, we will show how we solve the technical challenges through *hardware & software codesign* in each module, including how to balance the trade-off between performance and software control granularity (§3.4.1), how to design reliability mechanism with RDMA semantic (§3.4.2), how to achieve precise RTT measurement and effective congestion control algorithm (§3.4.3).

3.4.1 Chunking Algorithm

The chunking algorithm decides the granularity of each RDMA request that bursts to the network. A larger chunk size can bring traffic burst that may contribute to congestion and incurs high recover cost in case of packet loss, while a smaller one brings more CPU cost. The key design point is dynamically adapting chunking size according to current network status, which helps us achieve both good performance and fine-grained traffic control.

Bifröst users can customize their own chunking strategies according to the traffic pattern and application requirements. Here we present our default chunking strategy. We first cap the chunk size by the available congestion window or bandwidth delay product, whichever goes smaller. Note that the chunking module is an independent module from the congestion control. The chunking module still applies to have a fine-grained traffic control if Bifröst uses a rate-base congestion control, e.g., DCQCN, as the default. Given Bifröst always measures RTT to setup timeout value, we use estimated RTT, which reflects network queuing, as the feedback signal of chunking in Bifröst by default. The queuing delay (measured by $RTT - base\_RTT$) can reflects the buffer usage of in-path switches. A smaller *chunk_size* is adopted the when queuing delay grows larger.

To support the reliability design (see §3.4.2), Bifröst 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 congestion window, the *chunk_size* is set to $UNIT\_SIZE$ when available congestion window is smaller than $UNIT\_SIZE$. A message can always be chunked and sent when available congestion window is not zero.

3.4.2 Reliability

The challenges of designing *Reliability* module are software overhead and limited design space in RDMA operation.

A. Selective Retransmission for UC

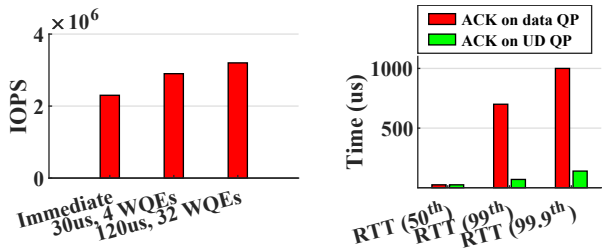
As we referred, UC drops the verb solely if one of the packets within it is lost. The later arriving verbs are still get delivered to the receiver, i.e., UC supports out-of-order delivery. This feature provides the possibility to design an effective selective retransmission reliability scheme. Similar to the TCP and RDMA RC's reliability mechanism, we use the chunk sequence number instead of packet sequence number to detect the packet loss and software ACKs to ensure the reliable delivery of the RDMA requests.

Chunk sequence number. RDMA **WRITE** is an one-sided operation that bypasses the CPU of the receiver. The receiver can not know whether a **WRITE** succeeds or fails in the UC transport. Besides, since **WRITE** writes a group of memory pieces to *one* continuous remote address, if we carry extra reliability headers in each data chunk, the data of an RDMA operation will not be continuous in the memory of receiver side. This causes additional overhead for data copy. Thus, to assemble the chunks into the original message on the receiver without inserting any headers, Bifröst uses **WRITE_WITH_IMM** to generate signals to software for the arrival of chunks. Another 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** requests, Bifröst uses an additional header as payload to carry additional information, including the sequence numbers.

A single sequence number space for all RDMA verbs can ensure in-order delivery, but the receiver can not validate the integrity of a large message by checking the sequence numbers of received chunks. The chunks of a large message are even not continuous in the sending queue since the **SEND**s are prioritized (§3.2). For example, a large message of $40KB$ is sliced into two chunks with size of $8KB$ and $32KB$. There are two small messages of $1KB$ and $4KB$ size submitting to Bifröst before all the **WRITE** chunks are sent to the network. The sequence number of chunk $8KB$, $32KB$, $1KB$ and $4KB$ can be $1, 4, 2, 3$. Thus the receiver can not validate the integrity of a large message by checking the chunk sequence numbers. Similar as QUIC [31], Bifröst 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. Note that Bifröst 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. **WRITE** WQEs and **SEND** WQEs have separate reliable sequence number space. 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. Bifröst acknowledges every WQE, but generating an ACK for each WQE can cause high overhead for small-message traffic. Bifröst puts multiple sequence numbers into one software ACK to reduce the network traffic and CPU cost of generating the software ACKs. 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. Bifröst sets triggers at receivers to send an ACK immediately: the cumulative number and size of WQEs (*e.g.*,



(a) Performance impact of ACK coalescing. (b) RTT accuracy of hyrid/isolated QPs.

Figure 4: Evaluation of optimizations on ACK designs.

32 WQEs, 32KB), the last WQE in congestion window (with a hint bit in the header or IMM_DATA field), and out of order of the reliable sequence number. Bifröst also sets a timer as a trigger because the tail (or small bursts) of a flow may not have enough data to trigger the cumulative counters.

To show the impact of different ACK triggering frequencies, we setup 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 4(a) shows the IOPS in 128-byte request/response RPC benchmark with different ACK triggers. The ACK coalescing mechanism improves over 40% IOPS compared to a per-WQE ACK mechanism (*Immediate* shown in Figure 4). In addition, among different coalescing WQE sizes and timer, the setting of 120μs timer and cumulative counter of 32 WQEs achieves a satisfying IOPS, which becomes the default configuration for Bifröst.

Retransmission. When the sender detects an out-of-order delivery from an ACK or a timeout event, it retransmits the WQEs. Bifröst handles the retransmissions by **SEND** whether the original WQE is **WRITE** or **SEND** since spurious retransmissions of **WRITE** may cause a data integrity issue. The data integrity issue can happen when WQEs are queued in the network for a long time, which incurs timeout retransission at the sender. In this case, the original WQE is received by the receiver and the whole message is submitted to 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 application have already changed without informing the CPU of the receiver. This phenomenon has been observed in our experiments. Bifröst retransmits WQEs by **SEND** of minimal chunk size (*UNIT_SIZE*) to avoid uncontrolled memory access from RNICs. In a *SEND* operation, the data is written into an piece of pre-posted memory. Bifröst then copies the data into its desired location if the message has not been submitted to the application.

B. Hardware reliability for RC

Bifröst is compatible with hardware reliability by using RC to reduce software cost and achieve better performance. 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 more larger than the software congestion window. Bifröst improves the hardware reliability by adding an additional software retransmission scheme. Bifröst sets a small retransmission retry times in RC QPs. This eliminates the unnecessary retransmission when the network is highly congested, i.e., the sender should slow down the retransmission. If the retry times of the retransimission fail, 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, Bifröst resubmits the uncompleted inflight WQEs to another pre-connected QPs (called backup QPs [32]) and flips the backup QP to be the primiary QP to continue transmission. At the same time, Bifröst re-connects the original QP in the background. When recovering the QP, we also consider bypassing the congested link or failure link (§A.3). The inactive backup QPs do not consume additional cache resources on RNICs thus have no side effect on performance [32].

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, Bifröst uses two retry times and two backup QPs for each connection. Compared to QP reconnecting, switching to backup QPs requires less time, *i.e.* $\sim$60μs. We also resolved a racing issue caused by the QP switch (§A.2.1).

3.4.3 Congestion Control

The decisions on congestion control. To break the inflexibility of hardware congestion control and eliminate hardware dependency, Bifröst extracts the congestion control module from data path. The congestion control module accepts the congestion signals, e.g., RTT, ECN, INT, and drop events, *etc.*, and calculates the new congestion window or new transmitting rates to avoid packets drop. Bifröst's default congestion control algorithm takes the RTT as the congestion signal. RTT is a common, precise, and switch-independent congestion control signal [30,37]. Though experienced drastic discussions [37,52,53], the effectiveness of RTT is proved by large scale deployment in Google (*i.e.*, Swift [30]).

Bifröst uses an RTT-based Additive Increase Multiplicative Decrease (AIMD) congestion control algorithm. After the publishing of Swift [30], we referred to it and refined our congestion control algorithm. Since Bifröst is also deployed in lossy fabrics, we also take packet drop events into consideration, the congestion window is set to the minimal value when packet drop occurs.

Accurate RTT measurement. As the congestion control signal, the accuracy of RTT measurement directly impacts the performance of congestion control algorithm. RoGUE [32] firstly mentioned common ways for RTT measurement for dynamic verb sizes on RNICs: utilizes the hardware timestamp functionality of RNICs to get accurate timestamp to calculate RTT. Note that leveraging hardware timestamp to measurement RTT requires to synchronize the software

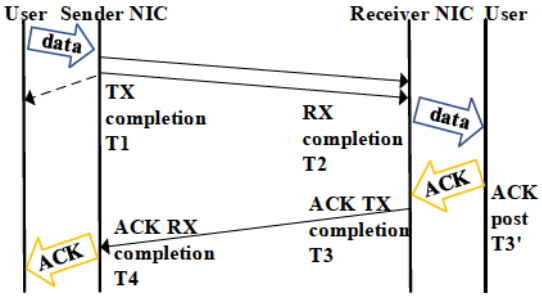


Figure 5: RTT measurement for UC in Bifröst.

and hardware clock (§A.5). Bifröst takes the RoGUE's methodology to measure RTT for the RC transport. Similarly, Figure 5 shows the RTT measurement method in Bifröst for the UC transport. On the sender side, the timestamp of the data WQE sends completion (T_1) and corresponding ACK receive completion (T_4) can be obtained from the hardware. On the receiver side, the timestamp of the arrival of the data WQE (T_2) and the corresponding ACK WQE post time (T_3') is accessible. Thus RTT can be calculated with:

$$RTT = (T_4 - T_1) - (T_3' - T_2) \quad (1)$$

However, this method is not absolutely accurate as T_3' is the post time rather than the actual send completion time of the ACK (T_3). In addition, in the case of 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 prolongs the congestion feedback loop such that the sender can not respond the network congestion timely.

Bifröst uses two optimizations to improve RTT measurement accuracy and shorten the feedback loop delay. First, Bifröst uses a high priority UD QP to send and receive ACKs to avoid head-of-line blocking and scheduling delay on RNICs. Second, Bifröst uses a separate completion queue for ACKs and polls it prior to the data completion queue in each batch. Figure 4(b) shows the improvement on RTT measurement accuracy with the same experiment setup (except a larger RPC request size, 1MB) as Figure 4(a) by comparing the measured RTT values with and without Bifröst optimization (legend **ACK on UDP QP** and **ACK on Data QP** in Figure 4(b)). As we can see, the measured tail RTT without optimization can be up to several milliseconds due to QP scheduling. The overestimated RTT can cause unnecessary window decrease. Using a separate QP and completion queue for ACKs with high priority provides 10× accuracy.

4 Evaluation

We evaluate Bifröst with a series of experiments in different dimensions:

1. How much software overhead does Bifröst bring in 100Gbps network (§4.2.1)?
2. How robust is Bifröst against packet loss (§4.2.2)? Is Bifröst competent for intra-/inter-pod traffic without PFC (§4.3)?
3. How well does the Bifröst's default *Congestion Control* work in large-scale incast scenario (§4.4)?
4. How well does Bifröst work with a hybrid deployment of heterogeneous RNICs (§4.5)?
5. How long does the service get disrupted when upgrading from current network to Bifröst in production systems (§4.6.1)?
6. How well does Bifröst work when supporting real production applications with different deployment (§4.6.2, §4.6.3)?

Name	Nodes	NIC
A	10	CX-4 Lx 25Gbps dual-port
B	90	CX-4 Lx 25Gbps dual-port
C	16	8 * CX-4 Lx 25Gbps dual-port 8 * CX-5 25Gbps dual-port
D	100	CX-4 Lx 25Gbps dual-port
E	48	CX-5 100Gbps dual-port

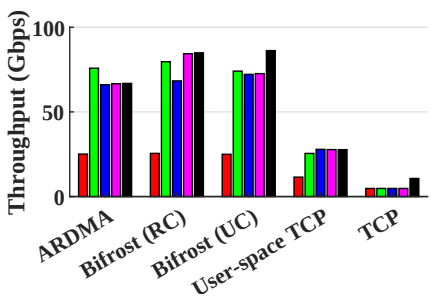
Table 4: Clusters setups used in evaluation.

4.1 Experiment Setup and Benchmarks

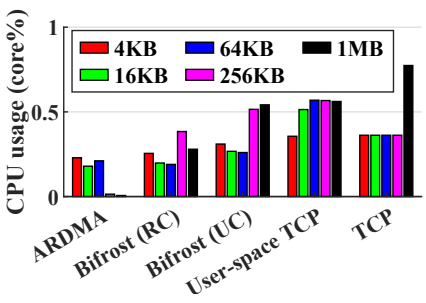
Cluster setup. Table 4 lists all the clusters used in our evaluation. The default RDMA configurations of our clusters are that: (1) for CX-4 lossless RNICs, PFC is enabled on ToR and Leaf switches but disabled on Spine switches; (2) for CX-5 lossy RNICs, PFC is disabled on all switches and the *lossy RoCE acceleration features* [48] are enabled. The lossy RoCE accelerations features include: (1) a fixed-size window of inflight PSN; (2) slow (non-line-rate) start and flow rate decrease on packet loss; and (3) adaptive retransmission timeout value. Note that these features are only available on CX-5 RNICs.

Baseline and workload. We have two types of benchmark framework: (1) a lightweight client/server RPC benchmark and (2) a distributed storage system running on top of an RPC framework. The RPC system supports ARDMA, Bifröst, user-space TCP, and kernel TCP. ARDMA is a vanilla RoCE-based RDMA library, which is deployed at the clusters in Table 4 before upgrading to Bifröst. The user-space TCP is a DPDK-based TCP communication library. We take ARDMA as the main baseline for comparison as ARDMA matches the vanilla RDMA performance. We run RPC benchmarks on testbed clusters of different scales and deployment (cluster A, B and C in Table 4). A Map-Reduce-like application running on top of our RPC framework is deployed in cluster D. A block storage service based on our distributed storage system runs in cluster E.

Configuration for Bifröst. In our clusters, RDMA traffic is transported on a specific priority queue both on switches and RNICs. The PFC and ECN are configured on this priority queue (lossless queue). Bifröst can run on the RDMA priority queue to be compatible with the lossless configuration. We put Bifröst on another reserved priority queue (lossy queue)



(a) Single-thread throughput.



(b) CPU usage.

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

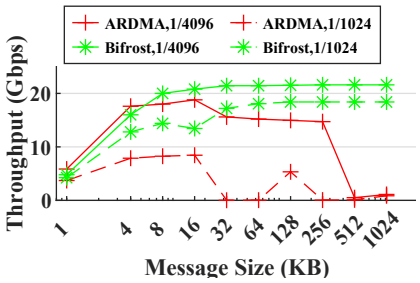


Figure 7: Performance of Bifröst with packet loss.

to disable PFC and DCQCN. Here are default parameters of Bifröst that we used in these evaluations. The coalescing ACK parameters are 120μs timer, 32 WQEs and 32KB data at most. The minimal and maximal chunk size is 4KB and 64KB respectively. The base RTT used in the software congestion control is 50μs.

4.2 Microbenchmarks

4.2.1 Software Overhead

To evaluate the software cost of Bifröst, we compare the single-core performance of our RPC benchmark with different network protocal stacks, including ARDMA, Bifröst (RC), Bifröst (UC), user-space TCP, and kernel TCP, on two CX-6Dx 100Gbps dual-port RNICs (not listed in Table 4). The CPUs used in this test are Intel series (2.9GHz). Both the RPC client and server run one connection on one port of the RNIC with a single thread. 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.

Figure 6(a) shows the throughput of all the stacks with different RPC sizes. As we can see, the single-thread throughput of Bifröst and ARDMA can reach to ~80Gbps while user-space TCP and kernel TCP achieve up to ~30Gbps even with large RPC size, i.e., 1MB. This shows that the *Chunking* mechanism does not hurt the throughput of RDMA. The throughput of large requests in ARDMA is lower than Bifröst because ARDMA uses RDMA **READ** operation to transfer the large requests, and RDMA **READ** operation has inflight bound on RNICs along with some well-known performance issue [23,28]. In addition, the single-core bandwidth of packet-IO solutions, e.g., Kernel TCP and DPDK-based TCP stack is much smaller which matches the results shown in Table 2.

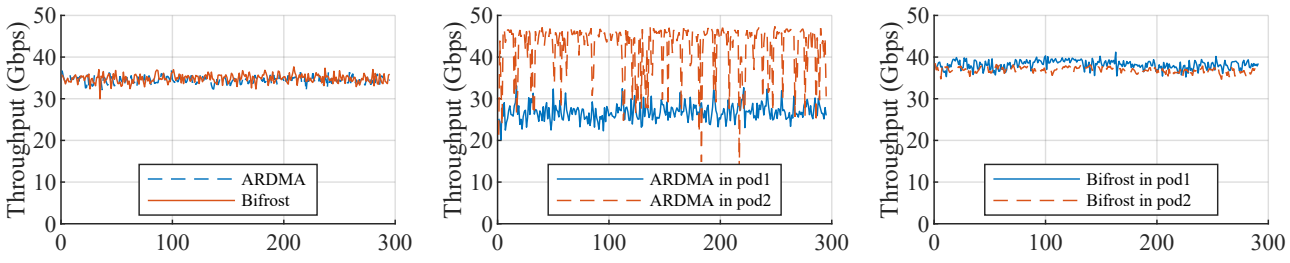
Figure 6(b) shows the corresponding CPU usage of all the stacks with different PRC sizes. The user-space TCP and Kernel TCP has higher CPU cost than Bifröst and ARDMA and Bifröst occurs more CPU cost than ARDMA. For 16KB small request sizes (transported via RDMA SEND operation), Bifröst (RC) and Bifröst (UC) consume 19.8% and 26.7% of CPU usage in a single core while ARDMA consumes 17.9%. This indicates that the *Reliability* mechanism contributes the majority of the additional CPU consumption. While for 1MB large request sizes (transported via RDMA WRITE_WITH_IMM operation), Bifröst (RC) and Bifröst (UC) consumes 28.0% and 54.2% of single-core CPU usage respectively while the ARDMA consumes 0.01%. The higher CPU cost on Bifröst (RC) compared with ARDMA is because one 1MB large message is split into multiple small chunks due to Bifröst chunking mechanism. Bifröst (UC) shows higher CPU cost than Bifröst (RC) because of the *Reliability* mechanism.

4.2.2 Performance with Packet Loss

To show the performance of *Reliability* mechanism of Bifröst, we test the throughput of Bifröst and ARDMA with manually configured packet loss. We pick two RNICs from cluster A in Table 4, turn off one port on each RNIC, and configure random packet drop on the ToR switch port connecting to the living port of the server RNIC. Figure 7 shows the throughput of ARDMA and Bifröst (UC) with varied request sizes (1KB~1MB) under two relatively high packet drop ratio (1/4096 and 1/1024) [25]. The I/O depth is also fixed at 8. Due to Go-back-N retransmission mechanism, the throughput of ARDMA decreases at small request sizes and dramatically drops to nearly zero at large request sizes. Benefiting from software selective retransmission, Bifröst always maintains ~20Gbps throughput without obvious performance loss across all RPC message sizes. Similar tests show that Bifröst is robust with higher loss rate, e.g., 1/256. However, higher loss rate is not practical in datacenters, we skip the discussion.

4.3 Intra and Inter-Pod Traffic

We also validate the performance gain of Bifröst in large-scale intra and inter pod tests on clusters A (pod1) and B (pod2) (Table 4). We use the common configuration in our clusters in the tests of ARDMA: PFC for RoCE traffic is enabled on ToR and Leaf switches and disabled on Spine switches. For Bifröst (UC) traffic, we use two kinds of configurations. For inter-pod full-mesh traffic, Bifröst is running on the lossy queue. For intra-pod full-mesh traffic, to verify the compatibility of Bifröst to PFC and DCQCN, we run Bifröst on the lossless queue. Both ARDMA and Bifröst (UC) are tested in a full-mesh traffic pattern. For each node, a client sends large RPC (512KB) requests to each server on other nodes with the IO depth of 8. The size of RPC response is 128 bytes. Each client and server uses 4 threads in the tests.



(a) Intra-pod full-mesh test. (b) ARDMA in inter-pod full-mesh test. (c) Bifröst in inter-pod full-mesh test.

Figure 8: The throughput of ARDMA and Bifröst in cross-pod scenarios.

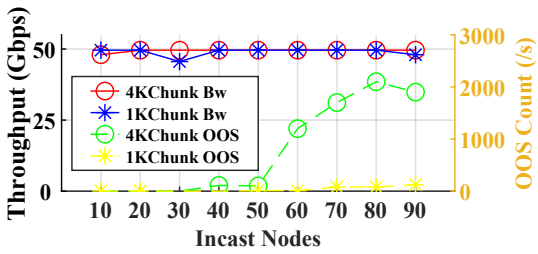


Figure 9: Bifröst performance in large-scale incast test.

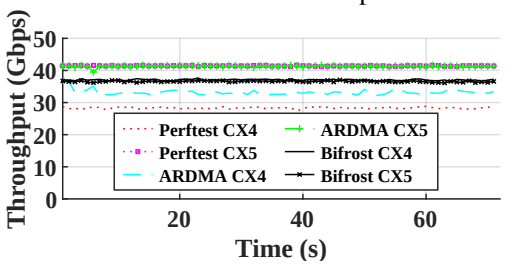


Figure 10: Bandwidth difference with CX4/5 hybrid deployment.

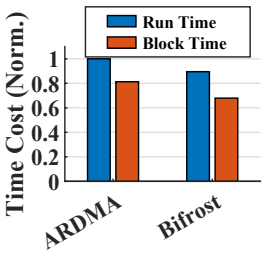


Figure 11: Performance of big data application with ARDMA and Bifröst.

Figure 8(a) shows the average sending throughput of each host with ARDMA and Bifröst (UC) in the intra-pod test. The results show that Bifröst (UC) with the software congestion control can cooperate with the hardware congestion control. The throughput of Bifröst (UC) is comparable to ARDMA and stable at about 35Gbps.

Figure 8(b) and 8(c) show the average sending throughput of all hosts with ARDMA and Bifröst (UC) in an inter-pod traffic test. The throughput of ARDMA 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 ARDMA is caused by packet loss as lossless is not configured in inter-pods switches. The clients in pod1 suffers more throughput loss because there are more servers in pod2 and the pod1 clients send more cross-pod traffic in this full-mesh traffic pattern. As a comparison, the throughput of Bifröst (RC) is steady and balanced between two pods. In summary, Bifröst can achieve higher and more stable throughput than ARDMA in the inter-pod tests.

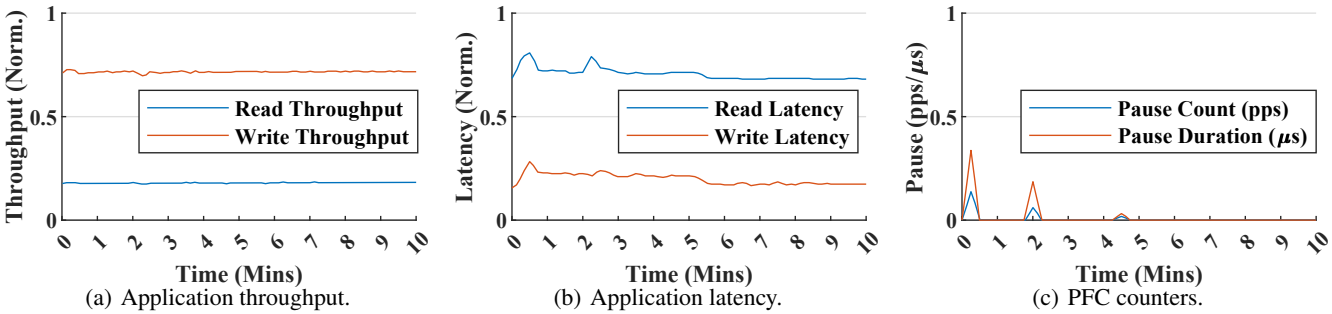
4.4 Large-scale Incast

We build a group of incast tests by varying the incast nodes in cluster B (Table 4) to test the performance of Bifröst's congestion control. We measure the throughput and Out-of-Sequence (OOS) counter, which indicates packet drop counters due to buffer overflow in network. We configure Bifröst running on the lossy queue, i.e., disabling PFC and DCQCN. For incast node N, we started RPC clients on N machines and connected to one RPC server on one machine. Each RPC client has 8 threads connected to 8 threads of a server and each client thread issues 32KB RPCs to the server with 8 I/O depth. The number of QPs on each client is $8 * 8 = 64$ and the number of QPs on the server is $N * 64$. Thus, the maximal incast degree is ~6000 when the node number is 90 in this test. According to our observation, we find that the size of minimal chunk influences the efficiency of congestion control. Thus, we also vary the minimal chunk size in this experiment.

Figure 9 shows the throughput and Out-of-Sequence counters with different incast nodes and varied minimal chunk size. With 4KB or 1KB as minimal chunk size, the incast bandwidth is always 50Gbps for all incast nodes. However, with the minimal chunk size of 4KB, the number of OOS starts increasing when the incast node is 60 (about 4000 : 1 incast). The reason of OOS is that the 4KB minimal chunk size limits the minimal congestion window size for each QP, and it is too large in such large-scale incast. Based on the analysis from above, using a smaller minimal chunk size (1KB) almost removes the occurrence of OOS in the large-scale incast. In summary, Bifröst is able to handle at least about 6000:1 incast (90 nodes) with stable bandwidth at 50Gbps and almost 0 packet drop.

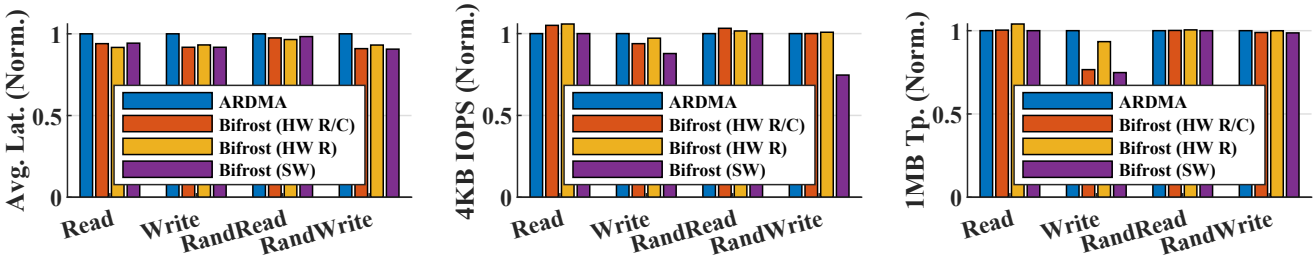
4.5 Heterogeneous RNICs

To show the behaviour of RDMA and Bifröst in heterogeneous environment, we test with 8 CX-4 and 8 CX-5 RNICs in cluster C (Table 4). We enable PFC and DCQCN and disable all RoCE-acceleration features on all CX-5 RNICs since CX-4 does not support them. To test the difference of control path among heterogeneous RNICs, we run *perftest* (ibtools) [18] and ARDMA with the standard lossless configuration. We first explore the RDMA traffic among heterogeneous RNICs with *perftest*. In a full-mesh traffic pattern with perftest, the throughput of CX-4 and CX-5 is 28Gbps and 41Gbps respectively. The throughput gap is 13Gbps (37.7%). We try to tune some configurations on PCIe (such as max read request



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

Figure 12: Metrics when upgrading from ARDMA to Bifröst.



(a) Average latency. (b) 4KB size IOPS. (c) 1MB size throughput.

Figure 13: Performance of ARDMA and Bifröst in Block Storage Service.

allowed on PCIe) and DCQCN (such as CNP interval and rate decrease period). However, it does not show any improvement in this test.

Then we test ARDMA and Bifröst in the RPC benchmark with a full-mesh traffic pattern. In comparison with ARDMA, Bifröst disables the hardware control path and runs with the software control path. Figure 10 shows the average bandwidth of CX-4 and CX-5 RNICs with ARDMA and Bifröst as time goes on. The average send throughput of CX-4 and CX-5 RNICs with ARDMA is 33.2Gbps and 41.3Gbps respectively. The line rate of these RNICs is equal but the actual throughput gap between CX-4 and CX-5 is 8.1Gbps (21.8%). The throughput of CX-4 and CX-5 RNICs with Bifröst is 37.1Gbps and 36.6bps and the throughput gap is 0.5Gbps (1.3%).

The bandwidth gap difference between perftest and ARDMA is due to the benchmark model. In perftest, the requests sent to different servers on the same node are issued by different clients, while in ARDMA they are issued by the same client sharing a fixed IO depth. Bifröst eliminates the throughput gap between CX-4 and CX-5 RNICs by replacing the hardware congestion control with a software congestion control, which minimizes the performance difference introduced by control path of heterogeneous hardware.

4.6 Production Evaluation

4.6.1 Non-Stop Upgrade

Many cloud services are running on the existing CX-4 online clusters. As a result, the cost of upgrading Bifröst by suspending all services and changing network configurations is unbearable. For example, the storage service must give high compensation and suffers from reputation loss if the system available time violates the service level agreement (SLA) [8, 11]. Bifröst is able to run in a lossless fabric with enabling DCQCN and PFC in hardware. Thus, we divide the upgrade process into three steps. ① Upgrading the business services to the version that supports both lossless RDMA and Bifröst. By default, the business services are still running on top of lossless RDMA network stack. ② Switching the network stack from lossless RDMA to Bifröst by configuring the real-time flag of the business services to choose Bifröst when issuing next RPCs. We can do that as Bifröst can also run with PFC and DCQCN. ③ Disabling PFC and DCQCN on switches and RNICs.

Figure 12 shows the application performance (throughput and latency) and PFC counters during an upgrade process. The software upgrading (step ①) is done before 0.5 minute of the time in the figure. At 0.5 minute of the time in the figure, we switched the software stack from ARDMA to Bifröst (step ②). Figure 12(a) shows that the throughput of application read and write operations have a slight jitter (<2%) when switching to Bifröst. Figure 12(b) shows that the latency increased about 10% for less than 30s during the transition time. Figure 12(c) shows that the PFC packets (packets per second, pps) and pause duration time (μs) experienced some extremely low jitters. At 5.5 minute of the time in the figure, we disable DCQCN and PFC (step ③). The throughput is not influenced and the latency decreases a little ($\sim$ 3%). When running Bifröst with and without PFC and DCQCN, all the metrics are healthy.

4.6.2 Big-data Applications

ServiceX is a Map-Reduce-like big-data application that runs on top of the distributed storage service provided by our system. One of most popular tasks in ServiceX is the sorting

task which needs to shuffle a large amount of data among mapper and reducer nodes through network. In this model, the delayed progress of any mapper nodes could impact the completion of the shuffle processing. The completion time of the shuffle processing decides the performance of the sorting task and desires a faster and more stable network transmission.

To show the impact of Bifröst on the real system, we run Bifröst and ARDMA in ServiceX on an existing lossless cluster (cluster D in table 4). The evaluation task is sorting 1TB data. The number of mapper and reducer nodes are both 1K and 1GB data are processed per mapper/reducer node. We collect two key metrics: the average running time of mapper and the average block time of mapper (*i.e.*, the time of transferring data in network). Figure 11 shows the results collected by using ARDMA and Bifröst individually. Compared to ARDMA, Bifröst reduces the average running time about 16% and accelerates the average block time about 10% for ServiceX due to more efficient congestion control.

4.6.3 High-performance Block Storage Service

Our system also serves for latency-sensitive applications such as the Enhanced SSD (ESSD) product of Elastic Block Storage (EBS). ESSD provides block storage service (virtual disks) with near-local access time on the cloud. Here we compare the latency, IOPS and throughput of an EBS application running with ARDMA or Bifröst in cluster E (Table 4). The workload is from real ESSD storage application. Read/Write/RandRead/RandWrite are operations of sequential/random disk reading/writing. There are three kinds of configuration of Bifröst here: Bifröst with hardware reliability and hardware congestion control (Bifröst HW R/C), Bifröst with hardware reliability and software congestion control (Bifröst HW R), and Bifröst with software reliability and congestion control (Bifröst SW). ARDMA is tested with RoCE lossy accelerations enabled. Bifröst is tested with these features disabled and running on the lossy mode.

Figure 13 (a) shows the normalized value of single-operation latency of ARDMA and Bifröst. The average latency of Bifröst and ARDMA is very close among all operation types. Although with software involvement, the latency of Bifröst is still slightly lower ($1\% \sim 8\%$) than ARDMA due to the fast software stack of Bifröst. In detail, Bifröst (HW R/C) has the steadily lowest average latency due to less software involvement. Bifröst (SW) and Bifröst (HW R) have slightly higher latency ($< 3\%$) due to the overhead of software stack.

Figure 13 (b) also shows the normalized IOPS for ARDMA and Bifröst for 4KB operations. The performance of Bifröst with hardware reliability (HW R/C and HW R) and ARDMA is generally comparable. The Bifröst (SW) with software reliability and congestion control has IOPS loss in Write and RandWrite due to software cost in reliability. Figure 13 (c) shows the normalized throughput for ARDMA and Bifröst for 1MB operation. The performance of Read/RandRead/RandWrite operations of ARDMA and Bifröst are comparable. While, for Write operation, all the Bifröst variants suffer from throughput loss due to software chunking overhead. Bifröst with hardware reliability and software congestion control (HW R) has the highest throughput since the software congestion control performs better than DCQCN, which makes compensation for the performance loss.

In conclusion, in supporting Block Storage Service, Bifröst has lower latency and comparable IOPS in most configurations. While, the software cost is also non-negligible compared to ARDMA in CPU-sensitive scenarios (like Write operations). Using Bifröst with hardware reliability and software congestion control is the best choice.

5 Related Work

Pure hardware solutions. Lossy RDMA solutions such as Mellanox CX-5/6 [46,47] and IRN [38] rely on new powerful hardware to run on lossy Ethernet. They can not be deployed with earlier RNICs (CX-4, for example). Besides, they are lack of flexibility because of the hardware implementations.

Pure software solutions. Pure software solutions such as SoftRoCE [17], eRPC [25] and SNAP [35] provide high flexibility and customizability without relying on specific hardware. However, they suffer from latency and CPU overhead [35].

Hardware & software co-design solutions. RoGUE [32] enables software congestion control for RDMA but relies on specific reliability (RC) to provide reliable service. RoGUE uses large static chunk size (64KB), and has poor performance in the scenario of packet loss and large-scale incast. 1RMA [43] is a high-performance network system that provides congestion control and reliability in software. 1RMA also enables one-sided RDMA operations. 1RMA [43] is designed on novel hardware with RDMA READ-like operation. However, it can't work on commodity RNICs, thus it has little help for existing RDMA systems. Table 5 in Appendix B summarizes differences between Bifröst and other network frameworks.

6 Conclusion

We present Bifröst, is a highly compatible and flexible PFC-free RDMA framework that solves a set of problems raised in production RoCEv2 clusters. These problems include PFC dependency, interconnectivity of heterogeneous RNICs and a jack-of-all-trades hardware-built-in congestion control scheme for all applications. Our evaluation on the testbed and production clusters show that Bifröst achieves high performance and flexibility in many scenarios, including packet loss, heterogeneous hardware, large-scale incast, and distributed systems. Bifröst are gradually deployed in our production cluster, where we observed a non-stop and smooth transition from the lossless RDMA to the PFC-free Bifröst, which preserves the service level agreement to our customers.

References

[1] Apache hadoop. http://hadoop.apache.org.

[2] Marcos K Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote memory in the age of fast networks. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 121–127, 2017.

[3] Alicloud. Alibaba smartnic. https://www.alibabacloud.com/blog/593986?spm=a2c5t.10695662.1996646101.searchclickresult.22a86035mGrhpG, 2021.

[4] Amazon. Amazon AWS. https://aws.amazon.com/.

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

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

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

[8] Amazon AWS. Amazon lightsail instance and block storage service level agreement service level agreement. https://aws.amazon.com/lightsail/sla-lightsail-instances-and-block-storage/, 2021.

[9] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350, 2006.

[10] 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*, page 26. ACM, 2016.

[11] Alibaba Cloud. Elastic compute service (ecs) service level agreement. https://www.alibabacloud.com/help/doc-detail/42436.htm, 2019.

[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[13] Jianbo Dong, Zheng Cao, Tao Zhang, Jianxi Ye, Shaochuang Wang, Fei Feng, Li Zhao, Xiaoyong Liu, Liuyihan Song, Liwei Peng, et al. Eflops: Algorithm and system co-design for a high performance distributed training platform. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 610–622. IEEE, 2020.

[14] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. Farm: Fast remote memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, pages 401–414, 2014.

[15] Yixiao Gao, Qiang Li, and et al. When storage meets rdma. In *NSDI*. USENIX, 2021.

[16] Yixiao Gao, Yuchen Yang, Tian Chen, Jiaqi Zheng, Bing Mao, and Guihai Chen. Dcqcn+: Taming large-scale incast congestion in rdma over ethernet networks. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 110–120. IEEE, 2018.

[17] Github. Softroce. https://github.com/SoftRoCE, 2016.

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

[19] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with infiniswap. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 649–667, 2017.

[20] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 202–215. ACM, 2016.

[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 *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 451–463. ACM, 2017.

[22] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, volume 8, 2010.

[23] 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.

[24] 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 *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 463–479, 2020.

[25] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter rpcs can be general and fast. In *USENIX NSDI*, pages 1–16, 2019.

[26] 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.

[27] Anuj Kalia, Michael Kaminsky, and David G Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided (rdma) datagram rpcs. In *OSDI*, volume 16, pages 185–201, 2016.

[28] Anuj Kalia Michael Kaminsky and David G Andersen. Design guidelines for high performance rdma systems. In *2016 USENIX Annual Technical Conference*, page 437, 2016.

[29] 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*, pages 1–16, 2019.

[30] 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. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 514–528, New York, NY, USA, 2020. Association for Computing Machinery.

[31] 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 *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 183–196, New York, NY, USA, 2017. Association for Computing Machinery.

[32] Yanfang Le, Brent Stephens, Arjun Singhvi, Aditya Akella, and Michael M Swift. Rogue: Rdma over generic unconverged ethernet. In *SoCC*, pages 225–236, 2018.

[33] 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*, page 3. ACM, 2015.

[34] 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 *Proceedings of the ACM Special Interest Group on Data Communication*, pages 44–58. ACM, 2019.

[35] 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 '19, page 399–413, New York, NY, USA, 2019. Association for Computing Machinery.

[36] 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*, pages 103–114, 2013.

[37] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, et al. Timely: Rtt-based congestion control for the datacenter. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 537–550. ACM, 2015.

[38] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for rdma. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 313–326. ACM, 2018.

[39] 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*, pages 30–43, 2018.

[40] 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.

[41] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. Supercomputing on nitro in aws cloud. *IEEE Micro*, PP:1–1, 08 2020.

[42] 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 16)*, pages 317–332. USENIX Association, 2016.

[43] 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 *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 708–721, 2020.

[44] 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*, pages 1–15, 2017.

[45] 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.

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

[47] NVIDIA Networking (Mellanox Technologies). Connectx-6 dx ethernet smartnic. https://www.mellanox.com/files/doc-2020/pb-connectx-6-dx-en-card.pdf, 2020.

[48] NVIDIA Networking (Mellanox Technologies). Cx-5 lossy features. https://community.mellanox.com/s/article/How-to-Enable-Disable-Lossy-RoCE-Accelerations, 2021.

[49] 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.

[50] 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*, pages 87–104. ACM, 2015.

[51] Jian Yang, Joseph Izraelevitz, and Steven Swanson. Orion: A distributed file system for non-volatile main memories and rdma-capable networks. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies*, FAST'19, page 221–234, USA, 2019. USENIX Association.

[52] 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. volume 45, pages 523–536, New York, NY, USA, August 2015. Association for Computing Machinery.

[53] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. Ecn or delay: Lessons learnt from analysis of dcqcn and timely. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 313–327. ACM, 2016.

APPENDIX

A Design Details

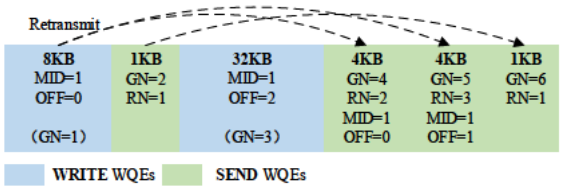


Figure 14: The numbering system of Bifröst. A **WRITE** WQE is numbered with message id (MID), offset (OFF) and global number (GN) while a **SEND** WQE is numbered with GN and reliability number (RN). Retransmissions of **WRITE** WQEs are transmitted via **SEND** of unit size and have one global number and two kinds of reliable numbers.

A.1 Software Reliability

A.1.1 Chunk sequence number

The global sequence number is a 64-bit value and all the RDMA WQEs have a unique global sequence number. Note that Bifröst 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 by the senders and not transmitted to the receiver, while the **SEND** WQEs also carry the global sequence numbers to the receiver.

WRITE WQEs and **SEND** WQEs have separate reliable sequence number space. The retransmission WQEs share the same reliable sequence number with the original WQE. Note that Bifröst use RDMA **SEND** 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 identified both by the sender and receiver.

The reliability sequence number of WRITE WQEs consists of 1-bit hint, 21-bit message id and 10-bit chunk offset. The hit bit is set when the WQE is the last WQE in the congestion window. We align the *chunk_size* to $UNIT\_SIZE$ (*e.g.*, 4KB). The chunk offset represents the offset between the start address of this chunk ($addr_c$) and of its belonged message ($addr_m$), *i.e.*

$$addr_c = addr_m + chunk\_offset * UNIT\_SIZE$$

The receiver can validate the integrity of the message by checking whether it has received data of all chunk offsets covered the message size. 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 does not carry the GN to the receiver while the **SEND** does.

In the case that 8KB **WRITE** WQE and the 1KB **SEND** WQE is retransmitted. The 8KB **WRITE** WQE is split into 2 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

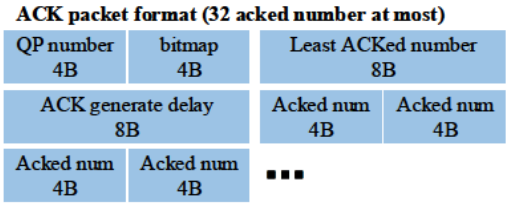


Figure 15: Software ACK format.

Figure 15 shows the software ACK format. The number carried *num_carried* in the ACK can be calculated by the length of the packet:

$$num\_carried = (ack\_length - 24B)/4B.$$

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. For example, an ACK of size 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.

To shorten the size of ACK packets, acked number is 32-bit. The reliable sequence number of a **WRITE** WQE and the global sequence number of a **SEND** WQE will be acked back to the sender. Note that the the reliable sequence number of a **WRITE** WQE 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 Bifröst if received. This window size is large enough in practice. The silently dropped WQEs, if there are, are detected by timeout.

Functionalities	Lossless RDMA	Lossy RDMA	eRPC	1RMA	Bifröst
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 5: Comparison on transport features of Bifröst and other network solutions.

A.2 Hardware Reliability

A.2.1 The Racing Problem

A racing phenomena occurs when QPs turn into error state occasionally: the sender may return a failure of a WQE while the receiver successfully receives it. This case happens when the **SEND** QP turns into the error state with inflight successful operations. In such case, the sender will post a duplicated WQE mistakenly on the other QP of the connection. Bifröst solves this problem by explicitly synchronizing the states of all inflight operations on senders and receivers. Posting WQEs into another QP may breaks the reliability due to repeated transmitting. There is also a corner case that we have experienced: the sender times out when the hardware ACK is on the flight. In such case, a WQE will also be posted repeatedly on the other QP of this connection. By retransmitting WQEs with RDMA **SEND**, whether the message has been submitted is checked before the data are copied into destination memory.

A.3 QP Recovery

Bifröst not only triggers the QP switch in RC mode, but also switches the QP by the timeout in UC mode. The network congestion or network link failure can cause the timeout events or the QP retries failure. Bifröst starts the retransmission on the backup QPs for RC and UC transport and recoverying the original QPs in the background. When recoverying the original QPs, a straighforward way is re-synchronizing the states, i.e., packet sequence number, between two end points. Bifröst also changes source UDP port [2]. The routing algorithm, e.g., ECMP algorithm, usually takes the 5 tuples to determine the traverse path of a packet in network. The source UDP port modification can change the path that a packet flows through the network, and thus, bypass a highly congested link or a failure point.

A.4 RTT Delay

Note that Bifröst 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 16 reports the 99^{th} percentile of these

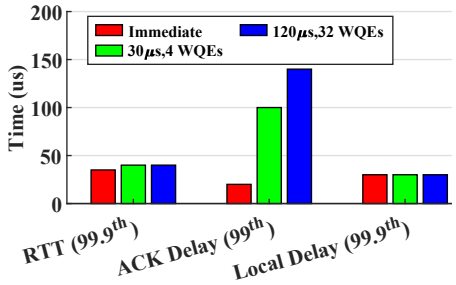


Figure 16: Evaluation of optimizations on ACK designs.

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 Bifröst prioritizes to poll the ACK completion queue and Bifröst 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.

A.5 HW/SW Clock Synchronization

The timestamps are generated by the NIC hardware clock. Except from obtaining timestamps from completions events to calculate, Bifröst may also need time for other usages, *e.g.*, setting retranmission timers. However, querying current time from RNICs is a time-consuming operation (*e.g.*, costs 1μs in CX-4). Thus Bifröst maintains a software clock based on *rdtsc()* and synchronizes the clock with hardware clock. When Bifröst sends or receives an operation and the clock is not corrected for 100μs, then Bifröst 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%.

B Comparison with Other Network Frameworks

Table 5 shows the detailed difference between Bifröst and other network frameworks.

[2] The L4 transport protocol of RoCEv2 is UDP.