Cepheus: Accelerating Datacenter Applications with High-Performance RoCE-Capable Multicast

Wenxue Li[1*], Junyi Zhang[2,3*], Yufei Liu[2], Gaoxiong Zeng[2], Zilong Wang[1], Chaoliang Zeng[1],
Pengpeng Zhou[2], Qiaoling Wang[2], Kai Chen[1,3]
[1]*iSING Lab, Hong Kong University of Science and Technology*, [2]*Huawei*,
[3]*University of Science and Technology of China*
{wlicv, zwangfb, czengaf}@connect.ust.hk, {kaichen}@cse.ust.hk
{zhangjunyi5, liuyufei16, gaoxiong.zeng, zhoupengpeng3, wangqiaoling}@huawei.com

***Abstract*—Modern datacenter applications widely exhibit multicast communication patterns. Meanwhile, RDMA is emerging as the de-facto networking architecture to meet the stringent performance requirement of applications. However, existing multicast approaches fail to efficiently collaborate multicast with commodity RDMA transport, either causing inefficient multicast traffic transmission or being trapped in the insufficient end-host transport protocol.**

In this paper, we propose Cepheus, which delivers performance gains from both multicast (i.e., traffic reduction and transmission hop minimization) and RDMA transport (i.e., ultra-low latency, high throughput and low CPU overhead). Cepheus reuses RoCE as its transport layer and provides a RoCE-capable multicast primitive via in-network assistance. At its core, Cepheus builds on and goes beyond the native multicast architecture by exploiting more switch functionalities to tackle the incompatibilities between multicast flow structure and RoCE processing logic. We prototype Cepheus on an FPGA board, as a building block attached to an Ethernet switch. Extensive experiments demonstrate Cepheus inter-operates with commodity RoCE protocol and outperforms existing RDMA multicast schemes, e.g., 5.2× faster multicast communication and 2.7× higher replication throughput for distributed storage.

I. Introduction

Modern datacenter applications are rife with one-to-many communication patterns that would substantially benefit from an efficient multicast primitive due to the ability of multicast to reduce traffic volume and minimize transmission hops. For example, distributed storage adopts multi-replications to ensure high availability, and thus multicast can be used to distribute replications among servers [70] and further accelerate the upper-layer storage protocol [13], [32]. In addition, one-to-many is a prevalent communication pattern in large supercomputing clusters and is frequently used in large-scale HPC applications [9], [25]. Furthermore, multicast can accelerate the parameter distribution process in distributed DNN training architectures, such as Parameter Server (PS) and the emerging In-Network Aggregation (INA). More use cases of multicast can be observed in streaming telemetry [45], [54], publish-subscribe systems [22], [34], and data-analytics platforms [39].

Despite being promising, existing multicast solutions are struggling to keep pace with the transport-offloading trend in datacenters. Remote Direct Memory Access (RDMA) is emerging as the de-facto networking technology, to meet the increasingly stringent communication requirements from applications, such as persistently high throughput, μs scale latency, and low CPU overhead. Many tech giants have adopted RDMA and deployed a huge number of RDMA NICs (RNICs) into their production datacenters [21], [24], [47], [77], which host various network-intensive applications [21], [30], [31], [37], [63]. RDMA over Converged Ethernet (RoCE)[1] is an RDMA transport protocol that is widely adopted in Ethernet datacenters. The *reliable connection* (RC)[2] mode of RoCE is mostly adopted because it supports complete transport functions. However, the mismatched abstraction between multicast flow structure and RoCE semantics prevents their effective collaboration. As a result, existing solutions fail to achieve performance gain from both multicast and RoCE (§II-C).

On one hand, *native multicast* (NMcast) (e.g., IP-based multicast [10], [16], [61]) relies on the switch to do replication and routing, and the sender only needs to send out one single copy of data. Therefore, NMcast can efficiently reduce traffic volume and minimize transmission hops. However, NMcast cannot support advanced layer-4 protocols such as TCP and RoCE, limiting its usage among datacenter applications. Some solutions propose building a custom transport layer for NMcast traffic, but this incurs significant performance, development, and management overhead, rendering it obsolete in modern datacenters.

On the other hand, many mainstream distributed frameworks (e.g., MPI [52], NCCL [51] and Spark [76]) develop their private *application-layer multicast* (AMcast) primitives on top of RoCE protocol. They use multiple RoCE connections to provide a layer-7 *logical* multicast interface to applications, where the traffic is indeed delivered by invoking multiple unicast (i.e., one-to-one) transmissions. AMcast is much more prevalent than NMcast due to the management feasibility from reusing RoCE. However, AMcast inevitably incurs redundant traffic and increased transmission hops, significantly degrading the overall communication performance [8], [16], [61].

In this work, we aim to design a high-performance multicast primitive that achieves (*i*) traffic reduction and transmission

*Equal contribution.

[1]RoCE has an extension version, RoCEv2. In this paper, we actually focus on RoCEv2 and use RoCE for convenient notation.

[2]In this paper, RoCE refers to its RC mode unless otherwise specified.

hop minimization as NMcast, (*ii*) ultra-low latency, high throughput, and low CPU overhead from reusing commodity RoCE, and (*iii*) interoperability with commodity RNICs and Ethernet switches for management friendliness. Our intuition is to build on NMcast (i.e., inherit its transmission-efficient multicast flow structure) and exploit more switch functionalities to deliver a RoCE-capable multicast stream that can be directly processed by commodity RNICs.

However, delivering a RoCE-capable multicast stream is challenging due to the mismatched abstraction between the multicast flow structure and RoCE semantics. First, multicast data packets are delivered in a one-to-many structure, with one stream stepwise converting to multiple streams at replication switches. This violates the one-to-one connection semantics of RoCE. Second, RoCE requires feedback from receivers to perform reliability guarantee, retransmission, and congestion control (CC). However, the feedback processing logic of RoCE is designed for a single feedback stream, and multiple feedback streams in multicast can confuse it and degrade the overall performance.

We propose Cepheus, a high-performance RDMA multicast primitive for datacenter applications (§III). (1) Cepheus reuses RoCE as its transport protocol and enables the multicast member to maintain only one RoCE connection. (2) Cepheus designs an on-switch accelerator to provide a RoCE-capable multicast primitive by effectively addressing the above challenges through: (*i*) preserving connection states in the network and bridging a one-to-many connection for multiple receivers; (*ii*) tailoring feedback in the network, enabling the sender to receive a compatible unicast-like feedback stream. (3) To enhance scalability, Cepheus manages feedback states in a hierarchical manner, bounding the switch memory overhead per multicast group, and supports efficient multicast source switching, significantly reducing the number of groups maintained on the switch.

We implement the Cepheus accelerator on an FPGA board and attach it as a building block to an Ethernet switch (§IV). Cepheus is evaluated through extensive testbed experiments and simulations (§V). Our results demonstrate that Cepheus interoperates seamlessly with commodity RoCE protocol and outperforms prior RDMA multicast schemes [2], [52], e.g., 5.2× lower multicast communication time, 2.7× higher replication writing throughput, and 12% lower completion time of HPC application. Furthermore, Cepheus maintains consistent performance in large-scale multicasts, exhibits satisfactory goodput under packet loss, and achieves fairness against unicast flows.

In summary, our key contributions are as follows:

- We designed Cepheus, a high-performance multicast protocol that leverages the benefits of both multicast and RoCE. Cepheus supports a RoCE-capable multicast primitive that reuses existing RoCE functions.
- We enhance the scalability of Cepheus through hierarchical feedback states management and efficient multicast source switching.
- We have implemented a Cepheus prototype and conducted experiments to demonstrate its interoperability with RoCE and its performance superiority in multicast communication.

II. Background and Motivation

We first exemplify the multicast pattern and summarize its requirements (§II-A), then introduce the basic concepts of RDMA network (§II-B), and finally reveal the insufficiencies of existing multicast solutions (§II-C).

A. Multicast Pattern and Requirements

As mentioned in §I, multicast is common in many applications, particularly in HPC, storage and DNN training. First, multicast is an essential element in supercomputers' total traffic. For instance, in a large IBM BG/Q supercomputing system (786,432-core) [9], multicast is the top-2 communication pattern, accounting for approximately 14% of total traffic (only less than allreduce (19%)). Moreover, multicast is common in HPC applications; for example, high-performance linpack (HPL) benchmark [25] is widely used to rank the supercomputer's computing capacity [68], and most of its communication traffic exhibits multicast pattern. Second, distributed storage offers high availability and durability using multiple (usually 3∼7) replications stored in different nodes to prevent data loss [21]. Previous works show that replication delivery significantly impacts the application's quality of service (QoS) [46]. Third, data parallelism is necessary for DNN training with a large dataset, such as Large Language Model (LLM) training [60]. In the PS architecture (one of the data-parallel approaches) [38], the aggregated gradients should be distributed from PS(s) to multiple workers. There is a similar distribution of gradients in the INA architecture [35].

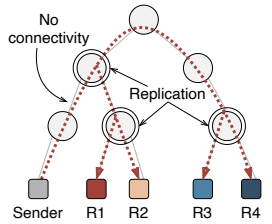
Requirements. The datacenter supports mixed traffic from different applications, including both large objects and small query messages, using multicast primitives. We aim to develop a *general* multicast mechanism that offers high throughput and low latency without being limited to specific applications.

B. RDMA Overview

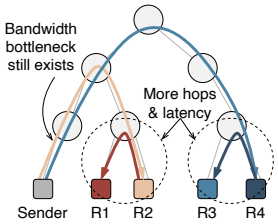
RoCE is an RDMA transport protocol that is implemented on RNICs and has been widely adopted by various tech giants at a large scale [21], [24], [47], [77]. RoCE offloads transport layer functions onto hardware to achieve high throughput and low latency with near-zero CPU overhead.

RoCE Transport Mode. Among the supported transport modes in RoCE, RC has various advantages. First, RC supports all RDMA operations, especially the zero-CPU-involved one-sided memory operations. Second, RC provides complete transport functions, e.g., (de)packetization, connection management, reliability guarantee, retransmission, and CC, to fully unleash the RNIC offloading capacity, thus achieving the best performance. As a result, RC mode is mostly adopted in today's datacenter applications [17], [49], [74].

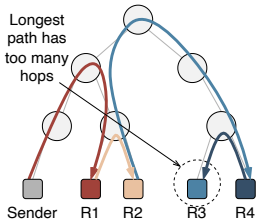
Non-programmability and SmartNIC. Currently, the complete RoCE transport functions are statically built-in with specialized circuits in RNICs, thus non-programmable, and



(a) Native multicast.



(b) Binomial tree (AMcast).



(c) Chain (AMcast).

	# Hops of longest path	Bandwidth bottleneck released?	Reusing commodity RoCE?	# End-host stack experience
NMcast	6 (min)	Yes	No	Once
Binomial Tree	8 (mid)	Partially	Yes	Many
Chain	14 (max)	Yes	Yes	Many
Cepheus	**6 (min)**	**Yes**	**Yes**	**Once**

(d) Cepheus vs. existing schemes.

Fig. 1: Existing multicast solutions and the comparision between Cepheus and them.

the non-requirement to modify RNIC is an important goal in various works [44], [49], [55]. In terms of SmartNICs, many popular SmartNICs provide programmability via on-NIC CPU cores and adopt the same RoCE specifications as the built-in transport protocol; thus, they only provide programmability on top of the transport layer while the transport layer remains non-programmable [3]. The FPGA-based SmartNICs provide full programmability, including the transport layer. However, building a custom transport layer for specific traffic incurs significant development overhead, as detailed later.

C. Insufficiencies of Existing Multicast Solutions

Existing multicast solutions cannot simultaneously achieve performance gain from multicast and RoCE protocol.

Native Multicast (NMcast). With NMcast, the sender only needs to send out one single copy of data, while the network replicates data at proper switches and eventually forwards the data to multiple receivers, as illustrated in Fig. 1a. This process follows a *multicast distribution tree* (MDT), where the replication is made as late as possible to reduce traffic volume. We compare different schemes under a 1-to-4 multicast ($S \rightarrow \{R_1, R_2, R_3, R_4\}$), shown in Fig. 1d. NMcast achieves the minimum number of hops, full bandwidth bottleneck release, and only goes through end-host processing stacks once.

However, due to the mismatched abstraction of NMcast data flow and transport's one-to-one connection semantics, NMcast cannot support standard layer-4 protocols such as TCP/RoCE [10], [16], [61], causing limited usage among applications. On the other hand, building a custom NMcast transport results in significant overhead. First, one approach is to build NMcast transport at software (as the existing RoCE is non-programmable), which foregoes the performant capability of hardware offloading. Second, FPGA-based SmartNIC enables the operators to build a hardware NMcast transport; however, this is less promising because the operators have to rebuild all the basic transport functions (some may already be baked-in in RoCE), causing non-trivial development overhead. In addition, the co-existence of different transports may cause unexpected behavior (e.g., unfair contention), making network management harder. As a result, custom NMcast transport is less promising in modern datacenters.

Application-layer Multicast (AMcast). AMcast develops a *logical* multicast interface using multiple one-to-one connections. AMcast reuses commodity RoCE as transport but inevitably incurs network overhead. The most straightforward AMcast method is for the sender to establish multiple connections with multiple receivers and independently transmit identical data. However, this causes a severe bandwidth bottleneck on the sender's outbound link [8]. Therefore, the *overlay multicast* approach that leverages a pipelined transmission strategy is more preferred [23], [51], [52], [76]. However, we show that it is challenging to achieve both high throughput and low latency with a single overlay multicast. We introduce two popular schemes below: Binomial Tree (BT) and Chain.

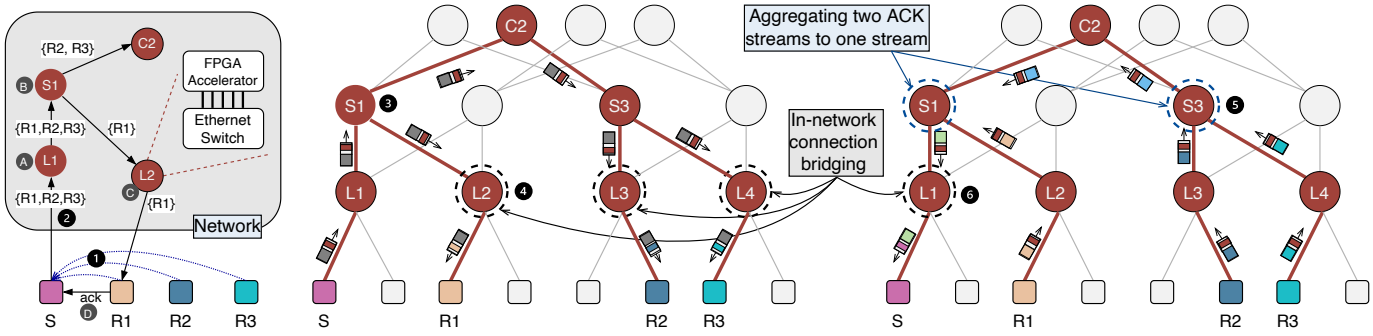
The BT approach takes a tree-like distribution method, which is recursive for a total of $O(log_2 N)$ steps (assuming a 1-to-N multicast), as shown in Fig. 1b. BT is good for short messages (i.e., latency-friendly) because it has a logarithmic latency form, but its bandwidth utilization falls far behind the optimal level, resulting in poor performance with large messages. The Chain approach forms nodes in a logical chain, and the intermediate nodes relay data to the next node after receiving data from the last node, as shown in Fig. 1c. Chain is throughput-friendly because it fully releases the tension on every node's link. However, Chain has a longer latency, which is linear to the number of nodes. We analyze BT and Chain in Fig. 1d. It is worth noting that messages in both BT and Chain go through the end-host stacks multiple times at every node, which affects latency too.

D. Goal, Challenge and Intuition

Our goal is to build on NMcast and exploit more proper in-network cooperations to provide a RoCE-capable multicast stream, which can achieve performance gains from both multicast and commodity RoCE. However, providing a RoCE-capable multicast stream is challenging.

C1: Connection semantics mismatch. Commodity RDMA relies on the destination Queue Pair (dstQP) field in the IB BTH header to correlate incoming packets to local connections [65]. This logic is designed for one-to-one connection and does not match the multicast data delivery pattern. This is because each receiver has a different QP Number (QPN), while the default NMcast replicates the entire packet [10], [28], resulting in all packets containing an identical dstQP that matches at most one receiver's connection.

Intuition: We can preserve necessary connection states in the network, follow the NMcast-like MDT for replication and forwarding, and bridge the sender connection to multiple receiver connections by modifying the replicated packet headers



(a) MFT Registration (b) Data Replication and Connection Bridging (c) Many-to-one ACK Aggregation

Fig. 2: Cepheus architecture and workflow. We take a three-layer fat-tree topology as an example. The sender S, three receivers (R_1 - R_3), and colored switches form an MDT. In (c), we take ACK as an example of feedback.

at the leaf switches. This enables the multicast data flow to transparently align with RoCE connection semantics.

C2: Issues with feedback interpretation. The feedback processing logic of RoCE [65] is designed to handle a single feedback stream from a single receiver. Thus, multiple feedback streams from multiple receivers can cause unexpected issues such as ACK and NACK inter-covering, and CNP magnification. These issues can lead to disturbance in RoCE's loss detection, retransmission, and rate-controlling routines.

Intuition: Our goal is to reliably deliver data packets to all receivers. So, we can aggregate ACK and NACK in the network to deliver a unicast-like stream that represents the overall loss condition among the group. Additionally, we can filter CNP[3] in the network to only reveal the most congested path to the sender. By doing so, we can proceed with the end-host reliability and CC routines correctly.

III. Cepheus

We first introduce the architecture and workflow of Cepheus (§III-A), and then we describe its *multicast forwarding table* (MFT) structure and connection bridging (§III-B), MFT registration (§III-C), RoCE-capable feedback handling (§III-D), and efficient multicast source switching (§III-E).

A. Architecture and Workflow

Fig. 2 depicts the architecture and workflow of Cepheus, which comprises three main phases: MFT registration, in-network data replication and connection bridging, and in-network feedback handling. Every switch in the *multicast distribution tree* (MDT) has its local MFT. All in-network processing functions are implemented in an FPGA board, as a building block attached to an Ethernet switch.

Hosts Establishing Connections. Each multicast task sets up a multicast group (MG) and is assigned a unique 32-bit McstID. The hosts in the MG follow the existing unicast-like procedure to establish one RoCE connection for this MG, where QP is used as the connection endpoint. A remote connection, identified by a <dstIP, dstQP> tuple, is required to activate the newly established QP. Cepheus assigns a virtual remote connection, i.e., no corresponding physical QP, to all established QPs: the dstIP and dstQP is set as McstID and a reserved value of 0x1, respectively. Commodity RNICs provide the API to specify the dstIP and dstQP without modifying the RNIC circuit [43].

MFT Registration. After connection establishment, Cepheus registers the MFT to switches to form an MDT. The MFT registration is performed in the control-plane (i.e., out-of-band), which comprises a controller and several lightweight agents on switches and hosts. The host that maintains the controller is referred to as the leader host[4]. The controller collects the IP and QPN states of other hosts (except the leader host) in the MG (❶), fits these states into the packet format of a self-defined MFT Registration Protocol (MRP), and transmits them to switches for building MFT and to other hosts for affirming their multicast membership (❷). The involved hosts confirm their participation by answering confirmations to the controller (Ⓓ).

Data Replication and Connection Bridging. The multicast sender (i.e., the source) transmits data through the performant RoCE protocol. Then, each switch in the MDT follows its local MFT to replicate (❸) and forward data to multiple receivers eventually. Whether a switch port is part of the MDT and on which switches the MDT needs branching are determined during the MFT registration. The leaf switches are responsible for modifying the BTH header to bridge connections for different receivers (❹). For instance, as shown in Fig. 2b, S transmits data only once; L_1 and C_2 are involved in the MDT but do not replicate, S_1 and S_3 replicate data packets to multiple downstream paths, and the leaf switches, $L_{2,3,4}$, bridge connections for $R_{1,2,3}$.

Feedback Handling. After receiving data packets, receivers generate normal ACK/NACK/CNP packets following the standard RoCE logic. The feedback packets traverse the MDT inversely, and the switches aggregate ACK/NACK and filter CNP when there are multiple input feedback streams (❺), eventually forwarding a single RoCE-capable feedback stream

[3]Modern RNICs commonly incorporate DCQCN [77] as the built-in CC mechanism, which uses CNP as the congestion signal. Therefore, in this paper, we consider CNP as the primary congestion signal.

[4]For ease of understanding, readers may perceive the leader host as the multicast source. In fact, the leader host can be any host in the MG.

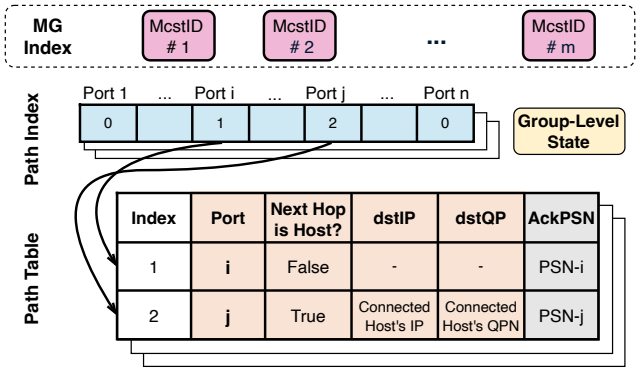


Fig. 3: MFT structure in Cepheus.

to the sender. The leaf switch connected to the sender modifies the packet header before forwarding the final feedback (❻). As shown in Fig. 2c (ACK as the example), L_2, L_3, L_4, and C_2 only forward ACK to next-hop switches, as there is only one ACK stream as input. S_3 and S_1 perform ACK aggregation since there are multiple input ACK streams.

B. MFT Structure and Connection Bridging

The Cepheus MFT guides the overall in-network processing logic, including the MDT routing paths, data replication and feedback handling. To ease the presentation, we first introduce the MFT structure and its correlated connection bridging, followed by the MFT registration process in §III-C.

*1) **MFT Structure**:* Fig. 3 illustrates the MFT structure in Cepheus, which comprises two components: *(i) Path Index* and *(ii) Path Table*. Cepheus maintains one MFT for each MG and many MGs can co-exist. The MFT of an MG is indexed by MG's McstID, which is assigned as the dstIP of all connections in this MG. Therefore, the data packets' dstIP can be used to index its corresponding MFT.

Path Index. This is an array that identifies whether a port is involved in the MDT. If the value in the i_{th} position is zero, $Port_i$ is not involved in MDT; otherwise, its non-zero value indicates the $Port_i$'s entry index in the Path Table. The size of the Path Index is n (# of switch physical port), which means the number of entries in the Path Table is at most n.

Path Table. Each entry of the Path Table represents an outgoing path. There are two types of paths with the next-hop device as a host and a switch, respectively. If the next hop is a host, Cepheus maintains dstIP and dstQP in this entry; otherwise, the two values are marked as invalid. Note that all entries, regardless of whether the next-hop is a host, maintain an AckPSN that records the largest Packet Sequence Number (PSN) of received ACKs from this path (explained later).

*2) **Data Replication and Connection Bridging**:* The switch uses its local MFT to route and replicate data along the MDT. The switch indexes the associated MFT using data packet's dstIP. Then, the switch utilizes the Path Index to iterate all entries in the Path Table. If there is more than one entry, the switch needs to replicate the data. For each entry, if its next hop is a switch, the switch just forwards the replication through this entry's port; if its next hop is a host, the switch

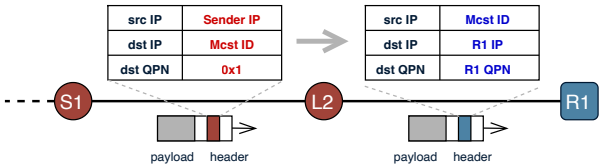


Fig. 4: Connection bridging in S_1-L_2-R_1 path.

first bridges the host's connection by modifying connection-related fields and then forwards the replication.

We take the S_1-L_2-R_1 path in Fig. 2b as an example and illustrate the connection bridging process at leaf switch L_2 in Fig. 4. Firstly, the dstIP and dstQP are changed from McstID and 0x1 to R_1's IP and QPN to match R_1's QP identification. Additionally, Cepheus changes the srcIP from the sender's IP to McstID. As a result, when R_1 generates feedback, the feedback's dstIP will be the data packet's srcIP, i.e., McstID. Thus, feedback packets can also index the associated MFT by their dstIP.

One-sided RDMA WRITE. The WRITE operation allows a host to write to a memory slot on a remote host without involving the CPU. The remote Memory Region (MR) information, e.g., remote Virtual Address (VA) and Remote Key (RKey), is indicated in the first packet of the WRITE request. The WRITE responder's RNIC checks whether the WRITE request matches its local MR and only executes the request when they match. Since the MR information is generally different on different receivers, Cepheus stores MR information in the MFT and modifies the WRITE request header for different receivers, enabling multicast WRITE. We introduce an extra progress to inform the switch about the MR information of different receivers.

C. MFT Registration

As mentioned in §III-A, Cepheus develops a custom MFT Registration Protocol (MRP) to register MFT. MRP is a UDP-based protocol and consists of four main steps.

First, the controller in the leader host gathers the connection information of all receivers in the MG through an out-of-band protocol (e.g., TCP) (❶). The information is then encapsulated into the MRP packet layout and sent to the leader host's leaf switch (❷). The packet layout is illustrated in Fig. 5. Cepheus sets the MRP packet's dstIP as McstID and assigns it a specific UDP port for MRP packet classification. The packet's payload includes the *metadata* and detailed connection information of the receivers, such as their IP and QPN. Due to the MTU limitation (usually 1500 Bytes), each MRP packet can only contain information from a maximum of 183 nodes. Therefore, if an MG has more than 183 members, the connection information must be spread across multiple MRP packets. The seq and total fields indicate the sequence number and the total number of MRP packets, respectively.

Second, upon receiving the MRP packet, each switch builds its local MFT and forms a part of MDT. An MRP packet (p) carries a McstID and connection information of multiple receivers ($p.nodes$). For each node in $p.nodes$, the switch finds its multicast routing port through the *unicast forwarding table*.

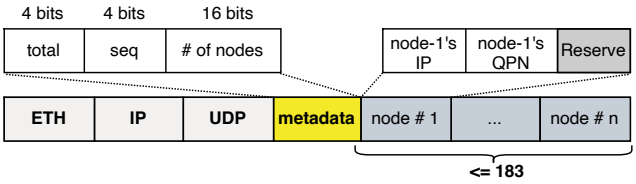


Fig. 5: MRP packet format.

Each time a routing port is determined, its Path Index value is marked, and its corresponding Path Table entry is filled.

For each node n in $p.nodes$: (1) If n is not directly connected, the switch finds the set of its routing ports/paths (Set_n). Then, if one port in Set_n has already been involved in the MDT (i.e., its Path Index value is non-zero), the switch reuses it as n's routing port to delay replication and saves bandwidth. Otherwise, the switch selects the port with the *lowest utilization* (i.e., least used by MGs) to perform load balancing at the group level. (2) If n is directly connected, the connected port is n's routing port and the switch fills n's connection information found in the MRP packet into its Path Table entry. We show an example in Fig. 2a, where at L_1, three nodes share an outgoing path to avoid replication (Ⓐ); at S_1, R_2 and R_3 share a path while R_1 must go on a separate path, so S_1 makes replication (Ⓑ); at L_2, as R_1 is directly connected to it, L_2 records R_2's connection information in L_2's local MFT (Ⓒ).

Third, after building the local MFT, the switch generates one or more new MRP packets to downstream devices for building their local MFTs. Each new MRP packet through a port only contains connection information of the receivers that are routed through that port. For example, S_1 sends the MRP packet that contains $\{R_2, R_3\}$ and $\{R_1\}$ to C_2 and L_2, respectively (Ⓑ). The MRP packets are forwarded hop-by-hop and eventually arrive at the receivers.

Finally, if a receiver receives the MRP packet that contains its IP address, this receiver will send a confirmation (ACK) to the controller, confirming its participation (Ⓓ). After the controller collects all confirmations, the MFT registration is finished and data-plane multicast transmission can start.

D. RoCE-capable Feedback Handling

In this work, we primarily focus on three types of feedback: ACK, NACK, and CNP. Our goal for feedback handling is to provide a RoCE-capable feedback stream that accurately guides the loss detection, retransmission, and rate control at the sender. Our key principles of feedback handling are:

- The sender only receives an ACK with AckPSN when *all* receivers have received all packets with $PSN \leq AckPSN$, matching RoCE's ACK coalescing scheme.
- The sender only receives a NACK with ePSN when *all* receivers have received all packets with $PSN < ePSN$, preventing the NACK inter-covering issue.
- Multi-stream CNPs should be filtered to enable the sending rate to match the most congested receiver.

ACK Aggregation. The ACK-related states in the MFT (Fig. 3) include (*i*) the group-level states, including the largest PSN of aggregated ACK from this switch (AggAckPSN), and the port from which the aggregated ACK should be sent (AckOutPort); (*ii*) the port-level states, including the largest PSN of ACK received from each port/path (AckPSN).

Upon receiving an ACK from a path of MDT, the switch firstly updates this port's AckPSN if the PSN of the incoming ACK is larger than the currently recorded one[5]. Then, Cepheus checks whether the *Trigger Condition* is satisfied; if yes, Cepheus will generate an aggregated ACK, send it out through AckOutPort, and update the AggAckPSN. The aggregated ACK's PSN is the minimum AckPSN for all ports, which is found by iterating through all MFT entries. As a result, every time the sender receives an aggregated ACK containing a PSN, the sender can confirm that all receivers have received all packets before this PSN.

Trigger Condition. Once generating an aggregated ACK, Cepheus records the port that owns the minimum AckPSN as triPort. Then, each time the triPort receives an ACK with a larger PSN than AggAckPSN, the aggregated ACK generation is triggered. Because of this Trigger Condition, not every ACK packet will trigger the aggregated ACK generation, so the number of ACKs received by the sender is reduced, mitigating the well-known ACK exploding issue [57].

Filtering Retransmitted Packets. The retransmitted packets also follow the MDT to be routed and replicated, similar to normal data packets. In addition, when receiving a retransmitted packet, for each $Port_i$ in MFT, Cepheus checks if this packet's PSN is greater than AckPSN of $Port_i$; if yes, Cepheus forward this packet through $Port_i$, otherwise not. This filtering mechanism saves bandwidth and prevents receivers from receiving duplicate retransmitted packets.

NACK Aggregation. When receiving an out-of-order packet, the receiver will generate a NACK packet to notify the multicast sender. Cepheus follows standard RoCE rules, and thus, the NACK contains the receiver's expected PSN (ePSN), and each NACK acknowledges all packets with $PSN < ePSN$. This rule must be carefully handled to prevent the NACK inter-covering issue. For instance, assuming there are two receivers: R_1 loses p_1 and R_2 loses p_2 ($p_1.PSN < p_2.PSN$). Cepheus must guarantee that p_1's NACK is forwarded to the sender first; otherwise, p_2's NACK will cover the loss of p_1, and the sender won't retransmit p_1 anymore.

Cepheus switch only forwards a NACK with ePSN when all receivers have acknowledged all packets with $PSN < ePSN$. Cepheus records the minimum ePSN in a group-level state MePSN, and updates it when receiving a NACK. When Cepheus generates an aggregated ACK, Cepheus checks if the minimum AckPSN for all ports equals $MePSN - 1$; if yes, Cepheus generates a NACK packet with ePSN as MePSN and sends it out through AckOutPort. By this checking, Cepheus makes sure that the NACK packet would not cover any preceding loss. Once a NACK is generated, MePSN is marked as invalid (discard previous history) and any incoming NACK

[5]Note that the received ACK can be an aggregated ACK from the connected switch or the original ACK from the connected host. Cepheus does not differentiate this when aggregating ACK.

could update it. Note that if there are not enough NACKs to trigger retransmission, the safeguard timeout at the end host will work to ensure reliability.

Bounded Memory Overhead Per Group. The most naive approach to managing ACK/NACK states is to track the per-receiver states on each passing switch or only on leaf switches. However, this approach has severe scalability issues as the state tracking overhead will linearly increase with the MG size. By contrast, as described above, each switch in Cepheus tracks only one NACK state and adopts a per-path tracking manner for ACK states. ACK states are aggregated layer-by-layer in Cepheus to form a hierarchical ACK tracking structure.

For instance, the table in S_1 (Fig. 2c) does not contain separate entries for R_1, R_2, and R_3 but only has two entries for paths to L_2 and C_2. The entry for the C_2 path contains the aggregated ACK states of R_2 and R_3. As a result, the AckPSN of C_2 path represents the minimum AckPSN of R_2 and R_3. Therefore, the Path Table size on a switch is *fixed* to at most n entries (# of switch ports), regardless of the size of the MG. We calculate that *1K* MGs at most cost 0.69MB of memory per switch (assuming 64 ports per switch), which is considered acceptable.

Congestion Control. We reuse the built-in DCQCN [77] mechanism in commodity RNICs to regulate the multicast sending rate. We adopt the *single-rate scheme* for multicast CC, matching the source sending rate with the most congested path in the MDT[6]. To this end, we enhance the switch with a CNP filtering mechanism while remaining the end-host DCQCN unchanged. Specifically, we maintain a *congestion counter* for each link at the switch, recording the congestion degree (i.e., number of CNPs during a given time window) from the receivers or downstream switches. We then perform *signal filtering* to only pass through the CNPs from the most congested link (forming a path when cascading each switch link end-to-end). Further, a *periodic aging* mechanism is added to update the congestion counter to match the frequently changing network dynamics. Note that there are different implementations of congestion signals (e.g., [12] reuses ACK to carry the congestion bits) and our CNP filtering mechanism works for them all with little modifications. We omit some technical details here.

Flow Control. We employ the native PFC [1] scheme without any modifications in our setup. The PFC operates in the following manner within Cepheus: Assuming a switch supports a multicast group (one ingress port to n egress ports), when an egress port receives a PAUSE frame, it is suspended. Thus, the corresponding ingress port halts replication among this group. Subsequently, the ingress port's queue grows and will send a PAUSE frame to its upstream entity if necessary. This working logic aligns with the design principle of Cepheus CC, where the multicast rate is controlled by the most congested path of receivers. In the flow control scenario, the on/off status of the multicast source is likewise controlled by the most congested path of receivers.

[6]There are single-rate and multi-rate schemes historically. Single-rate scheme [57], [75] is simpler at end-host as it only needs to maintain one sending stream with a rate matching the most congested receiver. Multi-rate scheme [41] maintains multiple sending streams with different rates at end-host to better adapt to receivers with different congestion degrees. However, the cost is bandwidth inefficiency because multiple streams are forwarded independently to corresponding receivers in the network.

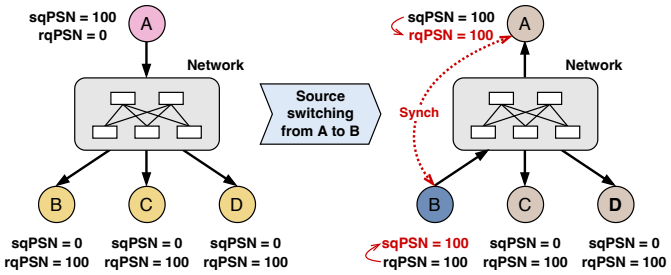


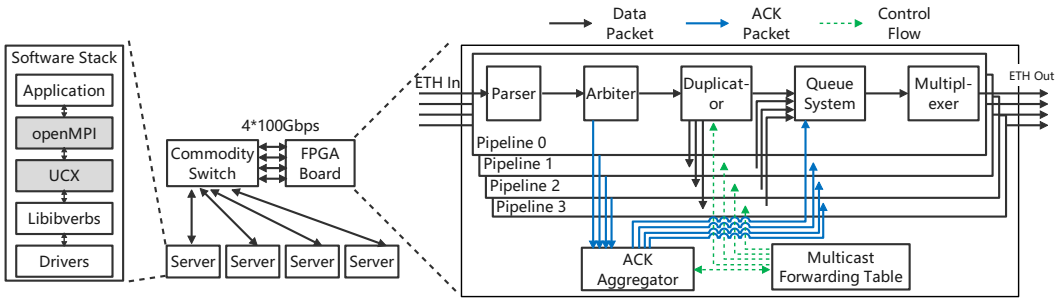
Fig. 6: Multicast source switching from A to B.

E. Multicast Source Switching

Multicast source switching inside an MG is common in datacenter applications. For example, in HPL [25], several nodes form an MG first and different nodes play as the multicast source in different epochs. To support this, the naive approach is to establish multiple MGs each with an independent MFT registered on switch. However, this approach would significantly increase the number of MFT on the switch, degrading scalability. By contrast, Cepheus efficiently supports the inside-MG multicast source switching with only one MFT and without reestablishing end-host QPs, significantly reducing the number of groups maintained on the switch.

In terms of the switch, when the source of an MG changes, the switches can detect this by recognizing the change of the incoming port of multicast data packets, which is recorded for later feedback forwarding. There are no other modifications to Cepheus' in-network logic.

For the end-host, we enhance a *PSN Synchronization* procedure to synchronize old and new source nodes' states. Note that each RoCE QP maintains two PSN records. The *Send Queue PSN* (sqPSN) is used to record the output packets, and the *Receive Queue PSN* (rqPSN) is used to verify the input packets. The sender' sqPSN should equal the receiver's rqPSN at the beginning of the transmission in a connection. For example, as shown in Fig. 6, node A has multicasted 100 packets to nodes B, C, and D. Assuming that all nodes' sqPSN and rqPSN start from 0, the sqPSN of A and rqPSNs of B, C, D become 100 when the transmission ends. When the multicast source switches to B, if B starts transmission immediately, the PSN of outgoing packets would be B's current sqPSN, i.e., 0. These packets would be dropped by C and D as their rqPSNs are already 100. Therefore, the *PSN Synchronization* procedure is used to maintain the consistency between sqPSN and rqPSN when the multicast source changes. In particular, the old source node assigns its rqPSN as its sqPSN, and the new source node assigns its sqPSN as its rqPSN. This problem can also be addressed by using Dynamic Connected Transport (DCT) [18], which synchronizes the PSN of the sender and the receiver with the DC Connect packet.



(a) Cepheus testbed configuration and accelerator pipeline.

Resource	Usage	Proportion
LUT	53169	4.8%
Register	15391	0.7%
BRAM	188	4.9%

(b) FPGA resource usage.

Fig. 7: Cepheus testbed consists of a commodity switch, an FPGA board, and four servers. The FPGA board implements four processing pipelines, each capable of line-rate traffic processing for a 100Gbps Ethernet interface.

IV. Implementation and Testbed

The testbed implementation comprises (*i*) an FPGA multicast accelerator, (*ii*) a commodity non-programmable Ethernet switch, and (*iii*) a set of software APIs exposed to applications at the end-host. Additionally, we discuss our rationale for selecting FPGA over a P4 programmable switch.

FPGA Accelerator. We have implemented all in-network functions, including MFT registration, data packet duplication, header modification, and feedback handling logic, on an FPGA board equipped with an FPGA chip [71] and four 100Gbps Ethernet interfaces. The FPGA board implements four processing pipelines and the implementation consumes only a small portion of FPGA resources, shown in Fig. 7b (note that the MFT is maintained in BRAM).

The FPGA board functions as a multicast accelerator, integrated as an external building block connected to the Ethernet switch; both sides spare four ports to connect. This multicast accelerator is fully compatible with legacy Ethernet switches, which are configured with Access Control List (ACL) rules to direct multicast traffic towards the FPGA board.

The processing pipeline of the FPGA accelerator is shown in Fig. 7a. Upon packet arrival, the FPGA board uses specific header fields to identify the multicast data (ACK)[7] in the *Parser* and *Arbiter*. Subsequently, the data (ACK) packets are duplicated (aggregated) by the *Duplicator* (*ACK Aggregator*), guided by the *MFT*. The duplicated or aggregated packets are then pushed in the *Queue System*, where we can perform physical-queue-level isolation. Then, queues await for the *Multiplexer* to schedule in case of queue competition. Finally, the processed data (ACK) packets are transmitted back to the commodity switch and forwarded based on their destination IP. During processing, the *MFT* is accessed as needed through internal control flows.

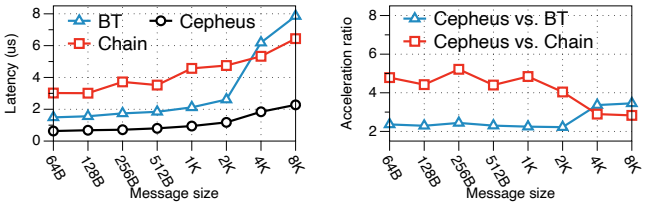
End-host APIs. We integrate Cepheus to MPI. Specifically, we add a new MPI_Bcast implementation in OpenMPI (v4.1.1) [52] and modify UCX (v1.11.2) [69] for RoCE QP creation, virtual destination assignment, and MFT registration. When the new MPI_Bcast is called, the MPI process calls UCX to establish RoCE QPs and register MFT to switches. When transmitting data, UCX invokes verbs [42] API to initialize RDMA operations. These software modifications are transparent to upper-layer applications and do not require any RNIC or RDMA driver modification.

Testbed Setting. We connect four commodity Dell servers to the Ethernet switch, forming a small-scale testbed, as illustrated in Fig. 7a. Each server is equipped with a ConnectX-5 RNIC with a 100Gbps interface.

Discussion on ASIC Implementation. Although we implement the Cepheus prototype with an FPGA accelerator external to the switch, it is better to integrate Cepheus into the switch chip to avoid occupying switch ports. Discussions with switch chip experts led to the conclusion that the integration is feasible, with minimal additional complexity and acceptable area cost. Specifically, We intend to process data packets (including packet replication and header modification) in an inline approach. For these data packet related operations, we can leverage the native IP multicast functionalities, which are already standardized in Ethernet switches, thus introducing negligible overhead. For feedback handling, as its throughput and latency demand is low, we plan to implement it in a dedicated Network Processor (NP) [11], serving as a small module mounted into the general switch processing pipeline (i.e., a look-aside approach), to reduce area and power costs. The resource usage in Fig. 7b suggests the extra ASIC overhead should be minimal even if the feedback handling is implemented in an inline approach.

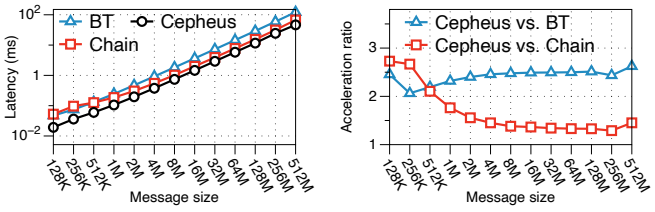
Discussion on P4 Programmable Switch. We opted for FPGA over P4 programmable switch options (e.g., Tofino [19]) for two reasons. Firstly, achieving compatibility with legacy Ethernet switches is one of our primary objectives, given the presence of numerous already-deployed Ethernet switches in datacenters. And we have the plan to integrate Cepheus into switch ASIC; the FPGA implementation serves as a preliminary validation step. Secondly, some functionalities of Cepheus are challenging to implement on P4 programmable switches. For example, the ACK aggregation requires the switch to traverse a table and find the minimal PSN, and the data replication requires the switch to recalculate the ICRC after header modification, both of which are difficult to implement on a P4 programmable switch. Note that we do not assert FPGA as the best choice in all scenarios.

[7] In Fig. 7a, we use ACK to represent all types of feedback.



(a) End-to-end latency. (b) Acceleration ratio.

Fig. 8: [Testbed] MPI-Bcast JCT of small messages.



(a) End-to-end latency. (b) Acceleration ratio.

Fig. 9: [Testbed] MPI-Bcast JCT of large messages.

V. Evaluation

We evaluate Cepheus through extensive testbed experiments and simulations. The testbed configuration is detailed in §IV. For the testbed experiments, we first examine Cepheus using micro benchmarks (§V-A); then we deploy Cepheus in two realistic applications to demonstrate its end-to-end improvement (§V-B). The simulations (§V-C) first evaluate Cepheus in the large-scale multicast, followed by an examination under various packet loss rates. Finally, the performance of Cepheus CC is evaluated. The key results are:

- Cepheus significantly accelerates multicast communication for both small and large messages compared to BT, Chain, and RDMC [2].
- Cepheus enhances the overall application performance, improving the replication distribution throughput in a storage system by 2.7× and accelerating the end-to-end HPL [25] completion by up to 1.12×.
- Cepheus achieves consistent performance in large-scale multicast, maintains good throughput under realistic packet loss rate, ensures fair sharing with unicast flow, and its CC can adapt to changing congestion bottlenecks.

A. Micro Benchmark

We integrate Cepheus into OpenMPI [52] and UCX [69] and provide a new MPI-Bcast implementation. Our testbed comprises four servers; one acts as the MPI-Bcast source, while the other three serve as receivers, forming an MG of size 4. We compare Cepheus with two OpenMPI AMcast algorithms[8], BT and Chain, which are oriented for small and large messages, respectively [67]. We also compare Cepheus with RDMC. We use the MPI-Bcast job completion time (JCT) as the metric and calculate Cepheus' acceleration ratio. We measure various message sizes from 64B to 512MB.

[8]The implementations of BT and Chain in NCCL [51] and Spark [76] are similar in OpenMPI.

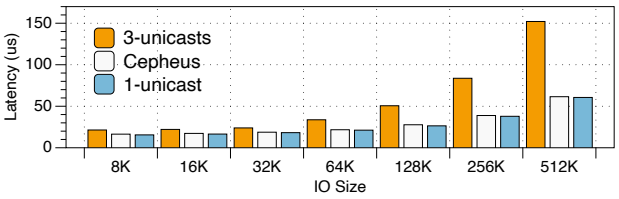


Fig. 10: [Testbed] Single IO latency under various IO sizes.

Scheme	1-unicast	3-unicasts	Cepheus
8KB IOPS(M)	1.188	0.413	1.167

TABLE I: [Testbed] Replication writing throughput in IOPS.

Lower Latency. As shown in Fig. 8, Cepheus achieves about 3∼5.2× and 2.5∼3.5× lower latency for small messages compared to Chain and BT, respectively. Chain gets the worst latency for small messages because its chain-like distribution manner causes a long transmission distance (linear to the MG size), which impacts small messages significantly. BT is regarded as latency-friendly because it adopts a tree-like distribution with a logarithmic latency form ($O(log_2 N)$). However, it still falls behind Cepheus. This is because Cepheus adopts an NMcast-like MDT to transmit data, thus achieving the minimized transmission distance and the lowest latency.

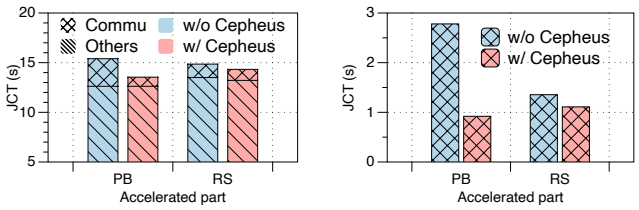
Higher Throughput. As shown in Fig. 9, Cepheus demonstrates a throughput improvement of 1.3∼2.8× and 2∼2.8× compared to Chain and BT, respectively. Chain is throughput-friendly and performs better with large messages than BT, but still falls behind Cepheus. Note that Chain could achieve optimal throughput with large messages only when using an infinitely-small slice size. In practice, it is not feasible to slice data into extremely small portions because each intermediate host must process every slice [53], resulting in substantial CPU overhead. Thus, in this experiment, we set the number of slices to four, which is equal to the number of hosts, a common configuration in real-world use cases. With this configuration, Cepheus achieves the highest throughput for large massages resulting from its efficient bandwidth utilization without relying on impractical assumptions.

Comparision to RDMC. RDMC [2] proposes an AMcast algorithm, which is designed specifically for large messages. In our evaluation, we compare Cepheus with RDMC using a large 256MB multicast communication. The results show that Cepheus achieves a significant JCT of *24.4ms*, while RDMC achieves approximately *35ms*. This is because, despite the fact that RDMC incorporates specific optimizations for large messages, RDMC continues to rely on multiple unicast transmissions to convey data, which inevitably leads to redundant traffic and affects its overall throughput.

B. Realistic Applications

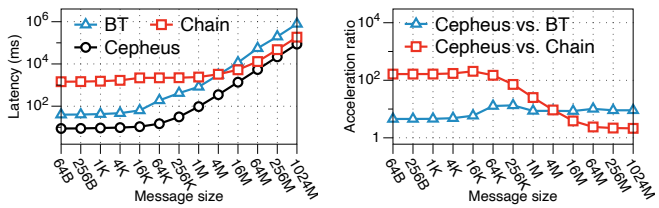
We deploy two realistic applications with multicast patterns and evaluate their performance with and without Cepheus.

*1) **Storage Data Replication**:* We integrate Cepheus into our proprietary distributed storage system to assess the performance of storage data replication. For our testbed, we designated one node as the client, while the other three nodes



(a) HPL end-to-end JCTs. (b) HPL communication time.

Fig. 11: [Testbed] HPL JCT. "Others" includes the PF phase and other computation processes.



(a) FCT of multicast flows. (b) Acceleration ratio.

Fig. 12: [Simulation] FCT of a 512-scale multicast.

served as storage servers, enabling a *three-replica writing* process. We compare Cepheus with the default 3-unicasts writing approach, where the client maintains independent RoCE connections with three servers. Additionally, the one-to-one (1-unicast) writing is measured as a baseline reference. The RDMA WRITE operation is used in all solutions.

Writing Throughput. We set the IO size as 8KB and let the client keep writing data to storage servers. We measure the average IOPS achieved by the client. As Table I shows, Cepheus achieves nearly optimal writing IOPS (1.167M IOPS), comparable to the ideal 1-unicast (1.188M). The 3-unicasts only achieve 0.413M IOPS (only 35% of Cepheus). Correspondingly, the application goodput with Cepheus achieves 76.5 Gbps (1.1M×8KB), while 3-unicasts only reach 26.24 Gbps. The advantage of Cepheus results from its efficient bandwidth utilization. Cepheus almost eliminates the overhead brought by multi-copies, achieving comparable performance to one-to-one writing. Note that the throughput bottleneck of consistent 8KB writing lies in the storage protocol stack at end-host, therefore both Cepheus and the one-to-one writing baseline cannot reach $>$ 90Gbps throughput.

Single IO Latency. We measure the single IO latency over different IO sizes, shown in Fig. 10. The latency is defined as the end-to-end time between the client submitting the IO write request and receiving the completion notice from the storage protocol stack. The result shows that Cepheus achieves lower latency than 3-unicasts, and delivers a comparable latency to the ideal 1-unicast. For instance, Cepheus reduces the IO latency by 23% and 60% in 8KB and 512KB, compared to 3-unicasts. Note that Cepheus achieves lower latency with larger IO sizes. As the result shows, the gap between Cepheus and 3-unicasts is enlarged as IO size increases.

The advantage of Cepheus for small IO sizes stems from the reduced number of times the data goes through the storage protocol stack and RoCE stack. In the case of 3-unicasts, the identical data is transmitted three times, leading to the data traversing the end-host stacks thrice. On the contrary, Cepheus ensures that the message only experiences the end-host stack once, effectively minimizing the IO latency.

*2) **HPL**:* We integrate Cepheus into HPL [25] and build a small HPL cluster using our testbed prototype. The overall HPL procedure includes three main phases: *Panel Factorization (PF)*, *Panel Broadcast (PB)* and *Update*. *PB* is a standard multicast transmission, and *Update* includes a communication step, called *Row Swap (RS)*, which can be implemented with multicast. In HPL, nodes are arranged as a 2-D Matrix. *PB* takes place among each row while *RS* takes place among each column. The total volume of multicast traffic of HPL is around tens of GBytes. We compare Cepheus with the original HPL implementation, where *PB* and *RS* are recommended to use the *increasing-ring* and *long* algorithms (both are AMcast algorithms), respectively.

End-to-end Performance. We use Cepheus to accelerate the *PB* and *RS* stages separately and measure the corresponding end-to-end HPL JCT (including all three phases mentioned above). The results are shown in Fig. 11a. For accelerating *PB* and *RS*, we form a logic 1×4 and 4×1 matrix[9], respectively; that's why the computation time in the two settings is slightly different. As illustrated by the results, Cepheus reduced the HPL JCT by 12% and 4% when *PB* and *RS* were individually accelerated, respectively. Note that HPL is computation-intensive, and prior works [64], [66] have demonstrated the challenge of achieving further improvements in HPL JCT solely from the computational aspect. In this work, we adopt a different approach by focusing on enhancing HPL's communication performance to improve its overall end-to-end performance, resulting in a significant 12% enhancement.

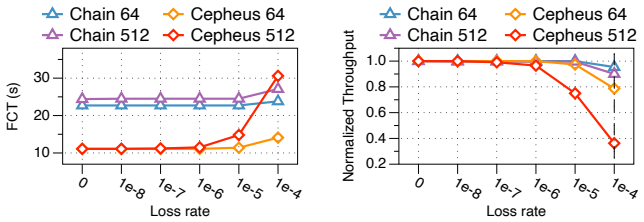
Communication Time. With the same setting above, we measure the communication time solely to highlight Cepheus' acceleration on HPL communication. As shown in Fig. 11b, Cepheus reduces the communication time of *PB* and *RS* by 67% and 18%, respectively. We further conduct a large-scale HPL simulation (up to 128*128 nodes) as a supplement to this testbed experiment, where Cepheus maintains consistent performance.

C. Simulations

We provide complementary experiments using ns-3 [50]. We first evaluate Cepheus against BT and Chain over a large multicast scale as supplements to §V-A. We then simulate a lossy environment and measure Cepheus over different packet loss rates. We finally evaluate the fairness of Cepheus against unicast flows and Cepheus CC's converging capability.

Setting. We simulate a large-scale 3-layer fat-tree topology with 1024 servers and a 1:1 oversubscription ratio. Each server

[9]Due to the testbed scale, we could only arrange nodes as 4×1, 2×2, or 1×4. There is no multicast communication between the 2×2 arrangement, which forces us to measure PB and RS separately, with 1×4 and 4×1 node arrangements, respectively.



(a) FCT of multicast flows. (b) Normalized throughput.

Fig. 13: [Simulation] FCT and normalized throughput under packet loss.

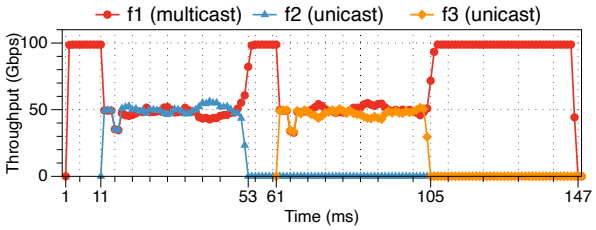


Fig. 14: [Simulation] Throughput dynamics of multicast and unicast flows.

is equipped with one 100Gbps RNIC, and each switch has 64 100Gbps ports. The retransmission and CC are go-back-N and DCQCN, same as Mellanox ConnectX-5 RNIC.

Large-scale Multicast. We simulate an extremely large-scale multicast communication with a group size of 512. We compare Cepheus with BT and Chain, measure the flow completion time (FCT) under various flow sizes from 64B to 1024MB, and calculate Cepheus' acceleration ratio. Results shown in Fig. 12 are consistent with results in §V-A.

For short flows, Cepheus achieves up to 164× and 4.5× lower latency compared to Chain and BT, respectively, which is a better improvement compared to in small-scale. This is because the minimized transmission distance of Cepheus has a greater effect with a larger scale. The prolonged transmission distance in Chain rapidly deteriorates its performance in large multicast scales, ultimately causing a 164× deficiency.

For large flows, Cepheus achieves up to 2.1× and 8.9× higher throughput, compared to Chain and BT, respectively. Cepheus achieves better improvement over Chain and BT on a larger scale. This is because Chain and BT require the host to transmit identical data multiple times, where the overhead increases with the size of MG. Cepheus maintains efficient bandwidth utilization under MGs of any size.

Loss Tolerance. We evaluate Cepheus under different packet loss rates, from 10^{-8} to 10^{-4}, which are emulated via randomly discarding packets in the middle switches. We choose two multicast scales (64 and 512) to evaluate and set the flow size as 128MB. Cepheus is compared to Chain. We measure the FCT and normalized throughput compared with the setting without loss, shown in Fig. 13. The results demonstrate that Cepheus consistently maintains better FCT than Chain when the scale is 64, but it performs worse under a 512 scale with a loss rate of 10^{-4}. We acknowledge that Cepheus throughput decreases largely than Chain. This is because the multicast sender in Cepheus is responsible to retransmit lost packets for multiple receivers, while the sender in Chain is only responsible for one receiver.

The loss tolerance of Cepheus is highly influenced by the end-host retransmission scheme. With the go-back-N retransmission, Cepheus' loss tolerance is limited. Therefore, we recommend deploying Cepheus in a lossless network with PFC [1] enabled, where the loss rate should be kept below 10^{-4} even during traffic bursts [26], [77]. Moreover, the recently-proposed IRN [48] can substantially enhance Cepheus' tolerance to higher loss rates. To further ensure sustainable delivery, we propose a safeguard scheme in §V-D.

Fairness and Convergence. We randomly select 16 nodes (denoted as $N_{0\sim15}$) from the topology, and simulate three flows: f_1 is a N_0-to-$N_{1\sim15}$ multicast flow, f_2 is a N_1-to-N_2 unicast flow, and f_3 is a N_3-to-N_4 unicast flow. Three flows initiate sequentially as shown in Fig. 14. N_2 and N_4 are the bottlenecks of f_1 and f_2, f_1 and f_3, respectively. We measure the average throughput of f_1, f_2, and f_3 in each millisecond.

As the result shows, Cepheus multicast flow achieves fair sharing with unicast flow and it can adapt to changing congestion bottlenecks. Specifically, Cepheus multicast flow (f_1) consumes bandwidth fairly when competing with unicast flows. Cepheus grabs total bandwidth at the beginning, then quickly converges to a fair sharing with f_2. Cepheus can adapt to different bottlenecks. As the result shows, after f_2 ends, Cepheus grabs total bandwidth again and quickly converges to a new fair sharing with f_3.

D. Safeguard Fallback

The realistic deployment of Cepheus must consider the possibility of extreme accident instances. For example, the MFT registration process may encounter insufficient switch memory, leading to failed initialization. Additionally, the switch or server misconfiguration could result in an abnormally high packet loss rate even in a lossless network, causing severe throughput degradation for Cepheus.

To prepare for such accidents and ensure sustainable traffic delivery, Cepheus employs a *safeguard fallback* scheme. This scheme automatically switches back to the default AMcast approach when detecting anomalies, such as facing MFT registration failures or experiencing abnormal throughput reduction below a predefined threshold (e.g., 50%) compared to normal conditions due to packet loss. The failure handling is not the primary focus of this paper, and we acknowledge that our current fallback scheme is preliminary. A more fine-grained fallback should be helpful; for example, when just a few nodes fail, a co-working approach of Cepheus and AMcast is promising. We leave the compresensive failure handling as our future work.

VI. Discussions

Comparison with SwitchML and [33]. Cepheus differs significantly from both SwitchML [58] and [33]. The multicast support in SwitchML is application-specific and tightly integrated with its dedicated framework, making it challenging

to adapt for broader applications. Moreover, technically, the RDMA multicast support in SwitchML relies on RoCE's Unreliable Connection (UC) transport, leaving retransmission and other transport functionalities at the application-layer level; this foregoes the performant capability of hardware offloading. In contrast, Cepheus is designed as a multicast primitive for general applications, for satisfying both latency- and throughput-oriented requirements (as mentioned in §II-A). Moreover, Cepheus leverages RoCE RC transport, thus inheriting all RC's advantages (as mentioned in §II-B).

Similar to SwitchML, [33] also operates in UC mode. While its goal is to adapt RDMA multicast for general applications, [33] falls short in addressing various practical requirements. Specifically, it lacks support for reliability, scalability, routing paths selection, etc. In contrast, Cepheus presents a comprehensive design that addresses all of these requirements.

Comparison with SHARP. Cepheus takes a distinct design principle compared to SHARP [59] in supporting reliable multicast. SHARP treats switches and end-hosts the same. As a result, SHARP switches implement all layer-4 transport functionalities (i.e., a complete layer-4 stack), totally resembling RNICs' behavior, which introduces significant complexity to SHARP switches. Moreover, SHARP switches independently retransmit packets, necessitating packet buffering space until the end-host confirms receipt, inducing a non-trivial memory overhead on the switch. In contrast, Cepheus is designed to minimize the additional switch overhead. Similar to the traditional network's division of labor, Cepheus switch is assigned an assisting role and only handles essential tasks, such as data replication, header modification, and feedback handling, which are much simpler than the full layer-4 transport. The minimal additional switch overhead is evident in the resource usage of our FPGA prototype (Fig. 7b).

Another significant distinction is that SHARP is dedicated to IB networks and lacks support for RoCE in its commercial product, while Cepheus is specifically designed for RoCE networks. Additionally, note that the GPU-direct RDMA is a memory management technique that is orthogonal to the transport layer processing. Cepheus can support GPU-direct RDMA without any obstacles.

Scalability and Resource Limit of FPGA Accelerator. The resource usage in Fig. 7b shows that our FPGA implementation uses only a small portion of resources, indicating our accelerator is not demanding in computation and memory resources. The key bottleneck resource limiting the scalability of the FPGA accelerator is the transceiver capacity. The FPGA chip [71] we used has 900Gbps capacity, thus supporting a maximum 1-to-8 (each with 100Gbps) fan-out factor per switch. Modern FPGAs offer even greater capacity. For instance, the Xilinx Virtex UltraScale+ [72] offers 2.6Tbps capacity, allowing a maximum 1-to-25 fan-out factor per switch. It's worth noting that Cepheus supports multi-rack multicast with bounded per-switch memory overhead; this means that we can scale the size of multicast groups by scaling the network topology without increasing the per-switch overhead. For example, with a 1-to-8 per-switch fan-out, each multicast group can reach a size of nearly 150, while a 1-to-25 per-switch fan-out can support group sizes of up to 4394 in a standard 1:1 oversubscription 3-layer fat-tree topology. Based on this analysis, we believe the current FPGA capacity meets our requirements for scalability.

Security Consideration. In this work, we focus on the private supercomputer clusters (or private cloud) owned by a single organization and thus there is no/rare attacker (even in control plane messages, e.g., MRP) since Internet attackers can be blocked by gateways and firewalls. The security consideration for the public cloud is out of the scope of this work.

VII. Related Works

Internet Multicast. Multicast has been widely applied in large-scale Internet applications [4], [5], [7], [73]. Prior works for the Internet [6], [14], [15], [27], [56] mostly focus on the multicast routing, i.e., to find promising multicast paths, inside ISPs. For instance, Yeti [14] supports effective multicast routing for large-scale ISPs, by creating labels that represent forwarding information of multicast graphs and utilizing them to make forwarding decisions. Most Internet multicasts merely provide best-effort delivery, which only works for applications without reliability requirements.

Datacenter Reliable Multicast. Multicast can be used to provide various upper-layer semantics, such as atomic multicast [29], [36] that necessitates access order guarantee. Some works focus on providing reliability upon NMcast for datacenter applications. However, they mainly rely on dedicated software transport protocols [20], [57], [62], [75], leading to notable performance drawbacks and increased development complexity. In contrast, Cepheus leverages the baked-in RoCE protocol to handle multicast traffic efficiently, delivering high-performance reliable communication.

Table-free Multicast Routing. Several previous works attempt to explore a table-free multicast routing scheme on switches [16], [40], [61]. For example, Elmo [61] encodes the multicast routing link into rules formatted as packet header, enabling the switch only maintain rule parsing logic. Orca [16] utilizes the server's large memory space to assist the switch-based forwarding, reducing the switch's burden.

However, these works only focus on layer-3 unreliable routing without incorporating any layer-4 functionalities. Cepheus is orthogonal with these works, aiming to empower multicast with prominent RoCE functionality rather than solely emphasizing the reduction of memory overhead. Furthermore, our scalability enhancement schemes enable Cepheus to support a substantial number of groups with minimal switch memory overhead.

VIII. Conclusion

We present Cepheus, a high-performance RoCE-capable multicast solution that delivers performance gains from both multicast and RDMA transport. Cepheus opens the door for efficiently leveraging the widely adopted RDMA transport with in-switch assistance to accelerate collective communication patterns. While this work mainly focuses on multicast; for

future works, we plan to extend Cepheus for more collective communication primitives, such as many-to-one (e.g., MPI-Reduce) and many-to-many (e.g., MPI-Alltoall).

ACKNOWLEDGEMENTS

We would like to thank the anonymous HPCA reviewers for their constructive feedback and suggestions. This work is supported in part by Hong Kong RGC TRS T41-603/20R, GRF 16213621, ITF ACCESS, NSFC 62062005, and Key-Area Research and Development Program of Guangdong Province (2021B0101400001). Qiaoling Wang and Kai Chen are the corresponding authors.

REFERENCES

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

[2] J. Behrens, S. Jha, K. Birman, and E. Tremel, "Rdmc: A reliable rdma multicast for large objects," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 71–82.

[3] BlueField, https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[4] BT IPTV, https://bit.ly/3ssCvTz.

[5] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li, "Celerity: A low-delay multi-party conferencing solution," in *Proceedings of the 19th ACM international conference on Multimedia*, 2011, pp. 493–502.

[6] S.-H. Chiang, J.-J. Kuo, S.-H. Shen, D.-N. Yang, and W.-T. Chen, "Online multicast traffic engineering for software-defined networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 414–422.

[7] T. W. Cho, M. Rabinovich, K. Ramakrishnan, D. Srivastava, and Y. Zhang, "Enabling content dissemination using efficient and scalable multicast," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 1980–1988.

[8] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM computer communication review*, vol. 41, no. 4, pp. 98–109, 2011.

[9] S. Chunduri, S. Parker, P. Balaji, K. Harms, and K. Kumaran, "Characterization of mpi usage on a production supercomputer," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 386–400.

[10] J. Crowcroft and K. Paliwoda, "A multicast transport protocol," in *Symposium proceedings on Communications architectures and protocols*, 1988, pp. 247–256.

[11] P. Crowley, M. A. Franklin, H. Hadimioglu, and P. Z. Onufryk, *Network Processor Design: issues and practices*. Morgan Kaufmann, 2003, vol. 1.

[12] H. Dai, B. Fu, and K. Tan, "PFC-Free Low Delay Control Protocol," Internet Engineering Task Force, Internet-Draft draft-dai-tsvwg-pfc-free-congestion-control-01, Apr. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-dai-tsvwg-pfc-free-congestion-control/01/

[13] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé, "Netpaxos: Consensus at network speed," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015, pp. 1–7.

[14] K. Diab and M. Hefeeda, "Yeti: Stateless and generalized multicast forwarding," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 1093–1114.

[15] K. Diab, C. Lee, and M. Hefeeda, "Oktopus: Service chaining for multicast traffic," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*. IEEE, 2020, pp. 1–11.

[16] K. Diab, P. Yassini, and M. Hefeeda, "Orca: Server-assisted multicast for datacenter networks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 1075–1091.

[17] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "{FaRM}: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.

[18] Dynamic Connected Transport, https://patents.google.com/patent/US8213315B2/en?q=(dynamic+connected+transport)&assignee=mellanox&oq=dynamic+connected+transport+mellanox.

[19] Edge-core Networks, https://www.edge-core.com/productsInfo.php?cls=1&cls2=180&cls3=181&id=335.

[20] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *IEEE/ACM transactions on networking*, vol. 5, no. 6, pp. 784–803, 1997.

[21] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan *et al.*, "When cloud storage meets {RDMA}," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 519–533.

[22] N. Garg, *Learning Apache Kafka*. Packt Publishing, 2015.

[23] A. Gibiansky, "Bringing hpc techniques to deep learning," *Baidu Research, Tech. Rep.*, 2017.

[24] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 202–215.

[25] HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, https://netlib.org/benchmark/hpl/.

[26] S. Hu, Y. Zhu, P. Cheng, C. Guo, K. Tan, J. Padhye, and K. Chen, "Tagger: Practical pfc deadlock prevention in data center networks," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, 2017, pp. 451–463.

[27] L.-H. Huang, H.-C. Hsu, S.-H. Shen, D.-N. Yang, and W.-T. Chen, "Multicast traffic engineering for software-defined networks," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[28] Infiniband architecture volume 1, general specifications, release 1.6, https://www.infinibandta.org/specs.

[29] S. Jha, L. Rosa, and K. Birman, "Spindle: Techniques for optimizing atomic multicast on rdma," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 1085–1097.

[30] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. 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)*, 2020, pp. 463–479.

[31] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using rdma efficiently for key-value services," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 295–306.

[32] A. Katsarakis, V. Gavrielatos, M. S. Katebzadeh, A. Joshi, A. Dragojevic, B. Grot, and V. Nagarajan, "Hermes: A fast, fault-tolerant and linearizable replication protocol," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 201–217.

[33] X. Z. Khooi, C. H. Song, and M. C. Chan, "Towards a framework for one-sided rdma multicast," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, 2021, pp. 129–132.

[34] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, no. 2011. Athens, Greece, 2011, pp. 1–7.

[35] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. M. Swift, "Atp: In-network aggregation for multi-tenant learning." in *NSDI*, vol. 21, 2021, pp. 741–761.

[36] L. H. Le, M. Eslahi-Kelorazi, P. Coelho, and F. Pedone, "Ramcast: Rdma-based atomic multicast," in *Proceedings of the 22nd International Middleware Conference*, 2021, pp. 172–184.

[37] H. Li, Y. Zhang, D. Li, Z. Zhang, S. Liu, P. Huang, Z. Qin, K. Chen, and Y. Xiong, "Ursa: Hybrid block storage for cloud-scale virtual disks," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–17.

[38] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[39] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded mapreduce," in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2015, pp. 964–971.

[40] X. Li and M. J. Freedman, "Scaling ip multicast on datacenter topologies," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, 2013, pp. 61–72.

[41] X. Li, S. Paul, and M. Ammar, "Layered video multicast with retransmissions (lvmr): Evaluation of hierarchical rate control," in *Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications.*

Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98, vol. 3. IEEE, 1998, pp. 1062–1072.

[42] Libibverbs, https://github.com/linux-rdma/rdma-core/blob/master/Documentation/libibverbs.md.

[43] Linux manual page, https://man7.org/linux/man-pages/man3/ibv_modify_qp.3.html/.

[44] S. Liu, Q. Wang, J. Zhang, W. Wu, Q. Lin, Y. Liu, M. Xu, M. Canini, R. C. Cheung, and J. He, "In-network aggregation with transport transparency for distributed training," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 376–391.

[45] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[46] R. Miao, L. Zhu, S. Ma, K. Qian, S. Zhuang, B. Li, S. Cheng, J. Gao, Y. Zhuang, P. Zhang *et al.*, "From luna to solar: the evolutions of the compute-to-storage networks in alibaba cloud," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 753–766.

[47] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *Proceedings of the 2015 ACM SIGCOMM Conference*, 2015, p. 537–550.

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

[49] S. K. Monga, S. Kashyap, and C. Min, "Birds of a feather flock together: Scaling rdma rpcs with flock," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 212–227.

[50] ns-3, a discrete-event network simulator for internet systems, https://www.nsnam.org/.

[51] NVIDIA Collective Communication Library (NCCL), https://developer.nvidia.com/nccl.

[52] OpenMPI: Open Source High Performance Computing, https://www.open-mpi.org/.

[53] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed dnn training acceleration," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 16–29.

[54] P. Phaal, S. Panchen, and N. McKee, "Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks," Tech. Rep., 2001.

[55] W. Reda, M. Canini, D. Kostic, and S. Peter, "Rdma is turing complete, we just did not know it yet!" in *Proceedings of NSDI*, vol. 22, 2022.

[56] B. Ren, D. Guo, G. Tang, X. Lin, and Y. Qin, "Optimal service function tree embedding for nfv enabled multicast," in *2018 IEEE 38th international conference on distributed computing systems (ICDCS)*. IEEE, 2018, pp. 132–142.

[57] L. Rizzo, "pgmcc: a tcp-friendly single-rate multicast congestion control scheme," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 17–28, 2000.

[58] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik, "Scaling distributed machine learning with In-Network aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 785–808. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/sapio

[59] Scalable Hierarchical Aggregation and Reduction Protocol (SHARP), https://docs.nvidia.com/networking/display/sharpv300.

[60] T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," *arXiv preprint arXiv:2211.05100*, 2022.

[61] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, and M. Hira, "Elmo: Source routed multicast for public clouds," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 458–471.

[62] S.-H. Shen, L.-H. Huang, D.-N. Yang, and W.-T. Chen, "Reliable multicast routing for software-defined networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 181–189.

[63] J. Shi, Y. Yao, R. Chen, H. Chen, and F. Li, "Fast and concurrent {RDF} queries with {RDMA-Based} distributed graph exploration," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 317–332.

[64] R. Shi, S. Potluri, K. Hamidouche, X. Lu, K. Tomko, and D. K. Panda, "A scalable and portable approach to accelerate hybrid hpl on heterogeneous cpu-gpu clusters," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2013, pp. 1–8.

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

[66] G. Tan, C. Shui, Y. Wang, X. Yu, and Y. Yan, "Optimizing the linpack algorithm for large-scale pcie-based cpu-gpu heterogeneous systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2367–2380, 2021.

[67] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[68] Top 500 List, https://www.top500.org/lists/top500/2022/06/.

[69] Unifiex Communication X, https://openucx.org/.

[70] Using reliable multicast for data distribution with opendds, https://bit.ly/3bWFefo.

[71] Virtex UltraScale, https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale.html/.

[72] Virtex UltraScale+, https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus-58g.html.

[73] Y. Wang, W. Wang, D. Liu, X. Jin, J. Jiang, and K. Chen, "Enabling edge-cloud video analytics for robotics applications," *IEEE Transactions on Cloud Computing*, 2022.

[74] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng *et al.*, "{SRNIC}: A scalable architecture for {RDMA}{NICs}," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1–14.

[75] J. Widmer and M. Handley, "Extending equation-based congestion control to multicast applications," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001, pp. 275–285.

[76] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica *et al.*, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[77] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.