MC-RDMA: Improving Replication Performance of RDMA-based Distributed Systems with Reliable Multicast Support

Chengyuan Huang †, Yixiao Gao †, Wei Chen †, Duoxing Li †, Yibo Xiao †, Ruyi Zhang †,
Chen Tian †, Xiaoliang Wang †, Wanchun Dou †, Guihai Chen †, Yi Wang ‡, and Fu Xiao‡
†State Key Laboratory for Novel Software Technology, Nanjing University, China
‡Nanjing University of Posts and Telecommunications, China

2023 IEEE 31st International Conference on Network Protocols (ICNP) | 979-8-3503-0322-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICNP59255.2023.10355619

***Abstract*—Remote Direct Memory Access has been widely adopted in distributed storage systems. However, it only supports unicast operations, which degrades the performance significantly for data replication because of bandwidth waste and CPU overhead. To address the problem, we propose *MC-RDMA*, a distributed and reliable multicast RDMA. It is compatible with existing unicast RDMA but supports lazy packet replication with reliable RDMA multicasting. The key idea of *MC-RDMA* is utilizing in-network programmable switches to build a NIC-transparent reliable multicast protocol for RDMA. *MC-RDMA* combines the address information of the IP and RoCEv2 into a sender-initialized multicast routing protocol. Besides, it synchronizes the hardware transmission states of multiple receivers by merging ACKs and NAKs. To verify the effectiveness of *MC-RDMA*, we implement it with Mellanox ConnectX-6 commodity RNICs and Intel Tofino P4 programmable switches. Experimental results show that *MC-RDMA* can double the sender bandwidth utilization and reduce the CPU overhead significantly compared to unicast-based RDMA replications. Moreover, it reduces the storage request latency by ~30% with realistic workloads and decreases the training time by ~50% in the distributed training system.**

***Index Terms*—RDMA, Storage System, Multicast, Data Replication**

I. Introduction

Remote Direct Memory Access (RDMA) [6]–[8], [22] has been widely used in distributed storage systems [32], [42], [46], [48]. It provides a way to directly access the remote Non-Volatile Memory (NVM) storage medium [18], [29]. It performs data operations such as replication and modification [2], [11], [36], without involving the target host's operating system, thus minimizing the overhead of software processing during data transmission.

Currently, commodity RDMA Network Interface Card (RNIC) only supports unicast operations [7], and the performance of RDMA-based data replication is lagged by two drawbacks of unicast traffic: *data redundancy* and *independent replicating states*. On the one hand, data redundancy in multiple replications incurs a large amount of bandwidth waste on the network links to replica servers. For example, in a three-replica primary-backup replication, the unicast-based replication wastes 50% bandwidth of the primary server uplink to transfer redundant replicated data to two backup replicas (§II-A). On the other hand, the independent replicating states reduplicate the CPU overhead during transmission, since the CPUs independently post the request, poll the completion, and keep tracking the delivery status of each unicast replication.

979-8-3503-0322-3/23/$31.00 ©2023 IEEE

The bandwidth waste and CPU overhead of unicast RDMA in replication can be reduced by introducing multicast [13], [16], [19], [31], [44] into RDMA. To make it efficient and readily deployable, there are several additional requirements:

- *Lazy data replication.* Delaying data replication from primary servers to the last points in the network, *i.e.*, the replication should happen in the network as late as possible along the network routing, thus saving most of the bandwidth from transferring duplicated data.
- *Reliable multicast RDMA operations.* With reliable multicast RDMA operations, the sender CPU only needs to post *one single* request and completion for each operation instead of posting and tracking the delivery status of each replication.
- *Compatibility with RDMA-Capable NICs.* The high performance of RDMA benefits from its hardware-offloaded protocol stack. To unleash the full potential of the hardware performance of RNICs, the multicast RDMA should be highly compatible with the current RDMA hardware.

To satisfy the requirements, the RDMA multicast system should address the following challenges. First, achieving lazy data replication requires the packet multicast protocol to support multi-level replications. The replicating switches are the bifurcated points of network paths to the receivers in a multicast group. The multi-level replication is similar to the IP multicast source tree [47], but the address information (IP, UDP, and RDMA) of the multicast packets should be modified during the replication. Second, to provide reliable multicast RDMA operations, the transport protocol should be carefully designed since RDMA operations have special semantics. For example, since the RDMA *WRITE* modifies the remote memory without passing the server CPU, duplicated software retries of RDMA *WRITE* may modify remote memory silently. Third, to be compatible with current RNICs, the key parts of multicast RDMA should follow the design and implementation

of unicast RDMA, including the hardware-offloaded reliability and congestion control.

In this paper, we present *MC-RDMA*, an reliable multicast RDMA that supports multicast message delivery and memory operation. It is designed for the distributed system with a high demand for data replication. Currently, *MC-RDMA* mainly focuses on the storage system with multiple data replicas to tolerate occasional node failures. However, we also show that *MC-RDMA* has the potential to accelerate communication beyond storage systems, such as the distributed training system (in §VII-F).

The key idea of *MC-RDMA* is utilizing in-network programmable switches to build a NIC-transparent reliable multicast protocol for RDMA. *MC-RDMA* combines multicast tree building and modification rule configuring in the multicast Bifurcation procedure (§IV-B). Meanwhile, it utilizes unicast RDMA hardware implementation to ensure the correctness of CPU-bypass RDMA memory operations at the transport layer. Finally, it synchronizes the hardware transmission states of receivers with in-network programmable switches [10] by designing ACKs merging and NAK processing according to the current RDMA protocol (§IV-C).

We implement *MC-RDMA* with Mellanox ConnectX-6 commodity RNICs and Intel Tofino [24] P4 programmable switches (§VI). In theory, *MC-RDMA* can support up to 3 (7) replications on Intel Tofino (II) and at least 320 hosts in distributed storage systems such as Ceph [48]. The experimental results show that *MC-RDMA* has similar performance to unicast RDMA with packet loss and the same capability of dealing with network congestion while achieving different fairness. In terms of communication acceleration, our results demonstrate that *MC-RDMA* can double sender bandwidth utilization and reduce CPU overhead significantly compared to unicast-based RDMA replications. It reduces the storage request latency by ~30% with realistic workloads. Moreover, the preliminary result shows that it cuts the training time by ~50% in the distributed training system.

In a nutshell, this paper makes the following contributions:

- We propose the novel idea of fully distributed reliable multicast RDMA, to reduce bandwidth waste and CPU overhead of replications in the distributed system.
- We introduce programmable switches to design reliable in-network multicast RDMA, which achieves *transparency* to endpoint RNICs and *compatibility* with the reliability and congestion control mechanism of unicast RDMA.
- We implement *MC-RDMA* with Mellanox ConnectX-6 RNICs and Intel Tofino P4 programmable switches. The evaluation results show that *MC-RDMA* improves bandwidth utilization and reduces CPU overhead significantly.

II. Background and Motivation

In this section, we first introduce two popular unicast-based replication mechanisms in distributed storage systems and show the bandwidth waste and CPU overhead of unicast-based replications (§II-A). Concentrating on the transport protocol for distributed storage systems, we then give a brief introduction to the user interfaces, transport primitives, and transport protocols of RDMA (§II-B). Based on the background knowledge, we discuss the opportunities (§II-C) and challenges (§II-D) for reliable RDMA multicast in datacenters.

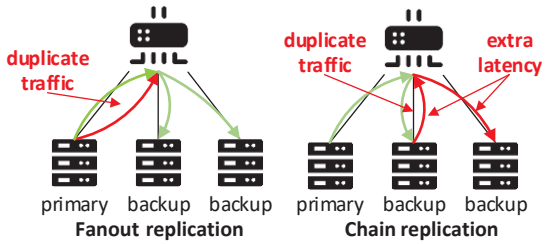


Fig. 1. The bandwidth duplication and latency overhead in unicast-based replications.

A. Replications in Distributed Systems

Primary-Backup Replication (PBR) is one of the most popular replication mechanisms in distributed storage systems [5], [12], [21], [48]. In PBR, one replica server is selected as the *primary*, and the remaining replica servers are called *secondaries* or *backups*. The read/write requests from clients are firstly sent to the primary node and then forwarded to secondaries. The replication of data from primary to backup may take many forms, *e.g.*, fanout [17] and chain replication (CR) [4], [38]. As illustrated in Figure 1, in fanout replications, the primary distributes data to the replicas in parallel and keeps checking the replicating states of the replicas. While in chain replication, data are replicated from the previous replica to the next one in a chain. Similarly, the parameter server in the distributed machine learning system iteratively synchronizes the updated model to all workers in the same group. It involves a larger-scale data replication than PBR, because of the larger number of workers belonging to a parameter server. The data replication from the parameter server to workers can also be conducted in the fanout/CR form.

Traditional unicast-based replication has inherent bandwidth waste because of duplicated data transfer and CPU overhead in the delivery of multiple replicas. Figure 1 demonstrates the bandwidth waste and latency penalty of the unicast-based fanout and chain replication. Fanout replication has a bandwidth bottleneck at the sender uplink since it sends the same data replications to two different receivers. Chain replication overcomes the bandwidth bottleneck at the sender by utilizing intermediate nodes for forwarding replica data. However, chain replication also has the longer replicating latency that is proportional to the number of replications, which is not preferred in most latency-sensitive applications.

B. RDMA Protocol

Transports and operations. RDMA provides two kinds of popular communication interfaces: one-sided memory operation (RDMA *READ*/*WRITE*) and two-sided message sending (RDMA *SEND*). Queue Pair (QP) is the entity of an RDMA connection on a network endpoint. Each Queue Pair consists of a Send Queue (SQ) and a Receive Queue (RQ). Users register memory buffers to the RDMA hardware driver and obtain *read*/*write* keys for registered memory. To initiate an RDMA *WRITE* operation of writing the data locating at the

local address at $buffer$ of size $length$ to the remote $addr$ memory address, the user posts Work Requests (WRs) with parameters $\langle addr, key, buffer, length\rangle$ to the RNICs. The $keys$ are verified by the local and remote RNICs for memory access protection. Users do not need to provide $addr$ for RDMA *SEND* operation because the data will be written into the memory pointed by the request that is pre-posted by the remote CPU into the remote RQ.

Reliability. RDMA protocol [6]–[8] dictates that the requester and responder should maintain the Packet Sequence Number (PSN) and expected Packet Sequence Number (ePSN) for each packet, respectively. The responder in Reliable Connection (RC) mode only accepts packets with PSN correctly matching ePSN and increases ePSN after successful receiving. The responder generates the *coalesced ACK* that contains the largest successfully received PSN to the requester. For example, $ACK = 5$ informs that the responder has successfully received all the packets with $PSN \leq 5$. Duplicated packets are silently dropped or re-processed according to the operation types [7]. The responder sends back a NAK packet containing the ePSN N to the requester when it receives an out-of-sequence packet. The requester then retransmits the packets ranging from the ePSN N to the largest PSN ever sent, which is also called Go-Back-to-N (GBN).

Congestion control and flow control. Current Ethernet and IP-based datacenter deploys *IP-routable RDMA over Converged Ethernet* (RoCEv2) [8]. DCQCN [49] is the built-in congestion control algorithm for RoCEv2 in Mellanox ConnectX RNICs. DCQCN uses ECN as the congestion signal: packets are marked in the egress queues of the switch according to the queue lengths. Once receiving an ECN-marked packet, the receiver sends Congestion Notification Packets (CNPs) to the sender to cut the flow rate. The deployment of RoCEv2 usually cooperates with the Priority-Based Flow Control (PFC) [23], a link-layer flow control mechanism that prevents packet drop due to congestion. A key design in PFC is reserving *headroom* buffers for in-flight incoming packets, which ensures that the switch buffer will not overflow.

C. Opportunities for Reliable Multicast RDMA in Datacenters

Today's datacenter network provides new opportunities for multicast RDMA protocol. First, for the distributed storage systems, the serving goal of multicast has shifted from large object delivery to many receivers, to data replication among a small group of replication nodes. The number of replication nodes in a replicated group, *e.g.*, Placement Group (PG) in Ceph [48], is small and fixed (*e.g.*, 3 replicas), which restricts the number of receiver states in multicast. Besides, the total number of PGs on each node is also limited (*e.g.*, 100 PGs per storage node is recommended [14]), which makes it possible to store and process the multicast groups inside the network.

Second, the low latency and rare packet loss [30], [50] in datacenter networks reduce the complexity of the multicast protocol. On the one hand, since packet loss is rare in a lossless network (*e.g.*, with PFC [23]), the out-of-synchronization receiver states are very rare. On the other hand, the ACK/NAKs can be forwarded back to the sender in time due to the low latency of RDMA. Thus, simple synchronization mechanisms for receiver states and retransmission mechanisms (*e.g.*, GBN) can still achieve high performance. In this environment, the utilization of intricate on-endpoint retransmission mechanisms or in-network cache is deemed unnecessary [13].

D. Challenges

Multi-level replication with packet modification. To achieve lazy data replication, *i.e.*, delaying data replication along the network path as last as possible, the multicast protocol should support multi-level replication on switches. The switches also have to modify the transport headers of replicated packets to make them correctly forwarded to the receivers. The IP-based multicast [25], [34] supports lazy packet replication but is hard to co-design with the transport-layer protocol, *i.e.*, RDMA. Besides, some complex IP multicast routing mechanisms such as broadcasting and pruning are not needed in a distributed storage system, where the number of replications does not change.

The reliable transport protocol for RDMA operations. Different from socket-based transport protocols such as TCP and UDP, the special semantics of RDMA operations incur more constraints in the transport layer. For example, the RDMA protocol ensures that the retransmission of RDMA *WRITE* will not be mistakenly written into the memory again after the completion of the original *WRITE* operation. However, since the RDMA *WRITE* modifies the remote memory without the involvement of the server CPU, software retries for RDMA operations may cause the data integrity problem. With the reliable multicast semantic, it should ensure the same results for multiple remote nodes.

Following unicast RDMA's design and implementation. The high performance of RDMA benefits from its hardware-offloaded protocol stack. Thus, the multicast RDMA should be highly compatible with the current RDMA hardware. Most multicast protocol designs should follow the implementation of unicast RDMA, including the hardware-offloaded reliability and congestion control (CC). However, the reliability of multicast differs from unicast since the receiving states of the packets from different receivers may be inconsistent. Besides, the CC of multicast flows is also different since the flows belonging to a multicast group may have their own congestion states on different paths. Finally, the non-programmability of RNICs is the biggest challenge for building the reliability and congestion control of multicast.

III. Overview

In this section, we present the building blocks and basic workflow of *MC-RDMA* (§III-A) and show the key ideas in *MC-RDMA* that solve the challenges for reliable multicast RDMA, including lazy data replication (§III-B), reliable multicast RDMA operations (§III-C), and compatibility with RDMA-capable NICs (§III-D).

A. Building Blocks and Basic Workflow

Figure 2 demonstrates the functionalities of control and data plane of *MC-RDMA*. The data plane of *MC-RDMA* consists of hosts with commodity RNICs and programmable switches (the left pink blocks). *MC-RDMA* does not modify RDMA

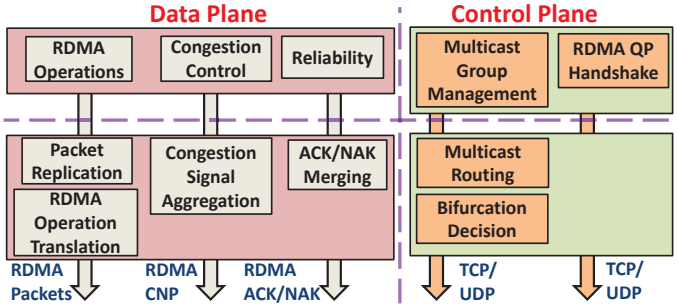


Fig. 2. *MC-RDMA* overview. The first/second row includes the elements in the physical host/switch, respectively. The first/second column includes the elements in the logical data/control plane, respectively.

TX/RX protocols on the host and provides multicast RDMA operations with reliability mechanisms and congestion control, based on the functionalities of unicast RDMA provided by commodity RNICs. The programmable switches are responsible for recognizing, replicating RDMA data packets, and translating RDMA operations. The programmable switches adapt the reliability and congestion control of unicast RDMA to the multicast traffic by aggregating congestion signals and merging ACK/NAKs.

The control plane of *MC-RDMA* is running on CPUs of hosts and switches (the right green blocks). It provides multicast group management and multicast connection establishment for multicast traffic. Specifically, the hosts connect and disconnect QPs in a multicast group. Also, it updates multicast group information and configures the multicast tables to the control plane of switches. The control plane of switches calculates the multicast routing and makes bifurcation decisions for replication.

The basic workflow of *MC-RDMA* is as follows: (1). Multicast group construction. The application constructs a multicast group including a sender and multiple receivers. (2). Connection establishment. The application sends the multicast request to the *Connection Management Daemon* (CMD) residing at the sender server. CMD handshakes with all the receivers, exchanging and configuring the initial states. (3). Switch configuration. CMD runs the multicast routing and configures the corresponding switches in the control plane. It helps to generate tables to guide the switch to accomplish key functionalities in the data plane, such as packet replication, ACK/NAK merging, *etc.* (4). Data transmission. The sender performs RDMA unicast to one of its receivers transparently. The data packets and ACK/NAK/CNP packets are manipulated in the network based on the switch configuration.

B. Lazy Data Replication

To achieve lazy replication for RDMA packets, *MC-RDMA* combines the address information of both the IP and RoCEv2 into a sender-initialized multicast routing protocol. First, *MC-RDMA* distinguishes the multicast group with the source IP and a *virtual QP number* that is unique on each server. Since ACK, NAK, and CNP packets in RDMA protocol do not have the source QP number in the protocol headers [7], the destination QP number (*i.e.*, the source QP number of multicast data packets) must be used in the multicast group ID to identify the multicast group. Second, *MC-RDMA* builds a multicast routing protocol based on unicast routing along with packet replication and address modification. Packet replications are only configured on the bifurcation points where receivers in the multicast groups have different next hops. Address modification is set on the last replication points to reduce the number of switches that stores the multicast information.

C. Reliable Multicast RDMA Operations

For each multicast group that contains a sender and multiple receivers, *MC-RDMA* establishes a connection that contains one QP on the sender (sender QP) and one QP on each receiver (receiver QPs) in the multicast group. The RDMA operations posted on the sender QP will be multicast to all receivers. For RDMA *SEND*/*WRITE* operations, the completion event of the send queue on the sender indicates that the operation has been successfully acknowledged by all receivers.

As mentioned in §II-D, the reliable transport protocol of RDMA operations should be carefully designed because of their special semantics. Thus, *MC-RDMA* utilizes *the hardware-offloaded reliability of unicast RDMA*, which is mature and has been verified by unicast traffic, to build the reliability of multicast RDMA. This is achieved by synchronizing unicast transmission states (§III-D).

D. Compatibility with RDMA-Capable NICs

Bundling RDMA QPs. The key step in establishing a unicast RDMA connection is exchanging the IP addresses, QP number (QPN), and initial SQ/RQ packet sequence number (PSN) between the sender and the receiver. With these attributes properly set into hardware, the SQ and RQ in the QP can transmit and receive packets with headers that carry corresponding information. Thus, we can *bridge* an SQ with multiple RQs by modifying the information used in the packets to different destination RQs. A multicast connection is established on a sender QP and multiple receiver QPs. The destination IP address of the send QP is the actual address of one of the receivers. On receiving these multicast packets, the packets are recognized, replicated and modified on each *bifurcated switch* (*i.e.*, the switches on the bifurcated points of paths to the receivers) to be properly routed to the receivers. When the replicated packets arrive at the receivers, these packets are the same as the unicast packets. Later, the receivers construct the ACK/NAK packets, and the destination IP address and the destination QP number of the receiver QPs are exactly those of the sender. Thus the receivers need no modification. The PSN of QPs are configured to match the packet sequence verification by the RX state machine on RNICs. As the PSNs of packets multicast to the server RQs are synchronized, the global states of server RQs are synchronously changed when receiving packets.

IV. Design

We present the detailed designs of *MC-RDMA* in this section.

A. Basic Endpoint Design

***MC-RDMA* user interface.** *MC-RDMA* operates RDMA QPs in the same manner as the unicast traffic on the data path. After the connection is established, the user can post

RDMA operations (RDMA $SEND$ and $WRITE$) into the sender QP. RDMA $READ$ operation is not supported since it is meaningless within the multicast semantic. The receivers should post receive buffers for receiving RDMA $SEND$ operations and register memory addresses to the switches for RDMA $WRITE$ address translation. The sender polls the completions from the completion queues as for unicast traffic. Different from RDMA unicast, each completion indicates that the posted operations are finished on all the replications.

Establishing connections. Each server runs a CMD that is responsible for local multicast group management and multicast connection establishment. It works as an agent to manage multicast connections while keeping transparent to the commodity RNIC. CMD can be implemented in $RDMA\_CM$ kernel module if the user applications use $RDMA\_CM$ to establish connections. When the sender tries to connect a multicast group $\langle SRC, DST_1, DST_2, ..., DST_N\rangle$ including a sender SRC and N receivers DST_i, it sends a request that includes multicast group information $\langle SRC, DST_1, DST_2, ..., DST_N\rangle$ and the handshake information $\langle SRC, QPN, PSN\rangle$ of the source QP to its CMD. Then, the CMD connects to destination QPs with the handshake information and allocates an ID for the multicast group MC_ID, which is also the virtual destination QP number of the sender QP. In this way, the behavior of CMD is transparent to the sender QP. To synchronize the initial states of receiver QPs, we configure the initial $PSNs$ of all QPs to the same value. After the handshakes with receiver QPs are done, the CMD queries the bifurcated switches from the control plane of *MC-RDMA* (see §IV-B). Then, it updates the multicast information $\langle SRC, MC\_ID\rangle$, destination address and QP numbers $\langle DST_1, QPN_1, ...DST_N, QPN_N\rangle$ to the bifurcated switches. Also, it sets the ACK aggregation information $\langle DST_1, QPN_1, ...DST_N, QPN_N, SRC, QPN\rangle$ to the Top of Rack (ToR) switch of the sender. Later, CMD returns the handshake response $\langle DST_1, MC\_ID, PSN\rangle$ to the sender QP. After configured with the handshake response, the connection establishment is completed and the sender is ready to send RDMA operations.

B. Lazy Data Replication

Packet multicasting. The replication and modification of packets are processed by programmable switches on the data plane, such as the Barefoot Tofino programmable switches in our implementation. With the multicast rules configured by the control plane, packets with $\langle SRC, MC\_ID\rangle$ are replicated and forwarded to different egress ports by utilizing the packet replication engine in the Traffic Manager of the switches. The destination IPs and QP numbers of the replicated packets are modified according to the multicast key ($\langle SRC, MC\_ID\rangle$) and the identification of the replication.

Multicast routing and bifurcation. The routing and bifurcation of *MC-RDMA* are distributed on the control plane of in-network switches. For each multicast group with key $\langle SRC, MC\_ID\rangle$ and destinations $\langle DST_1, DST_2, ..., DST_N\rangle$, the control plane runs *multicast bifurcation procedure* (algorithm 1) to locate the bifurcated switches, configure the rules of multicast, and modify the corresponding packets. Since each bifurcation creates one or more new replicas of the original multicast packet that costs additional bandwidth, the basic principle of bifurcation is *bifurcating as late as possible*, *i.e.*, *lazy replication* mentioned before. Function `Bifurcation` is the part of the algorithm running on switch s that decides how the multicast $\langle src, mc\_id\rangle$ $\langle dst_1, ...\rangle$ bifurcates in the network.

As shown in algorithm 1, D_{vv} represents a vector of D_v and D_v is a vector of destinations dst that have the same next hop on switch s. In the initial state, D_{vv} only contains a vector D_1 that has one element dst_1, which means D_{vv} has one next-hop group initially (line 2). Later, each dst_i will be

Algorithm 1: Multicast Bifurcation Procedure

Input: Mulitcast key $\langle SRC, MC\_ID\rangle$, destination vector $\langle DST_1, DST_2, ..., DST_N\rangle$
Output: Multicast tables built on bifurcated switches

1 **Function** `Bifurcation(`$\langle dst_1, ...\rangle$, *switch* s`)`**:**
2 $D_{vv} \longleftarrow \langle\langle dst_1\rangle\rangle$
3 **foreach** dst_i *in* $\langle dst_2, ...\rangle$ **do**
4 $r \leftarrow FALSE$
5 **foreach** D_i *in* D_{vv} **do**
6 **if** *NextHop(s,*dst_i*) = NextHop(s,*D_i*[0])* **then**
7 $D_i.push\_back(dst_i)$;
8 $r \leftarrow TRUE$; **break**;
9 **end**
10 **end**
11 **if** *!r* **then**
12 construct a destination vector D;
13 $D_{vv}.push\_back(D.push\_back(dst_i))$;
14 **end**
15 **end**
16 **if** $D_{vv}.size() > 1$ **then**
17 **foreach** D_i *in* D_{vv} **do**
18 Add rule: *replicate packets that match* $\langle SRC, MC\_ID\rangle$ *to NextHop(s,*D_i*[0])*;
19 Add rule: *destination IP* $\leftarrow$ D_i*[0]*;
20 **if** $D_i.size() = 1$ **then**
21 Add rule: *destination QPN* $\leftarrow$ *the* QPN *of* $D_i[0]$;
22 **end**
23 **end**
24 **end**
25 **foreach** D_i *in* D_{vv} **do**
26 **if** $NextHop(s, D_i[0])! = D_i[0]$ **then**
27 `Bifurcation(`D_i*,NextHop(s,*$D_i[0]$*))`;
28 **end**
29 **end**
30 **End Function**
31 $S \leftarrow NextHop(SRC, DST_1)$
32 `Bifurcation(`$\langle DST_1, ..., DST_N\rangle$,$S$`)`;
33 **return**

attempted to be contained in a vector D_i sequentially (line 3–15). Specifically, if the next hop of dst_i is the same as that of a vector D_i, dst_i will be merged into D_i (line 5–10). On the contrary, if the next hops of all D_i are different from that of dst_i, a new vector D_i will be constructed and the elements in it share the new next hop (line 11–14). Note that we use a boolean variable r to decide whether to construct a new vector and its initial value is $False$. Then, if there are multiple next-hop groups on the switch, the control plane will add the multicast rule "*the packets should be replicated on the switch*" and the modification rule "*modify the destination IP address of replicated packets to one of their groups*" to the data plane (line 18–19). If a next-hop group only contains one destination, it means this destination shares no common links with other multicast packets. Thus, the control plane should add the rule of destination QP number modification to match the real QP number at the receiver (line 21). The matching key of these multicast and modification rules is the $\langle src, mc\_id \rangle$ fields in the packet header. The destinations in the same next-hop group are sent to the next-hop switches as the input to execute `Bifurcation` iteratively (line 27). The algorithm starts `Bifurcation` at the ToR switch of the sender (line 31–32) and traverses the multicast tree until the destinations.

C. Transmission States Synchronization

ACK/NAK merging. The ACK merging is based on the coalescing ACK mechanism and the retransmission mechanism that RDMA adopts. Since each ACK acknowledges all the packets with smaller PSNs than the ACK carries, the merged acknowledgment state of the receiver QPs in a multicast group is the minimum PSN of all the ACKs. The switch records the ACKs of each receiver QP in a multicast group and takes the one with the minimum PSN as the response to the sender. The challenges of implementing the ACK merging strategy include the *one-time data access* and *data isolation among pipelines* features of programmable switches.

One-time data access. In P4 programmable switches, packets are processed with match-action tables, while each table can only be accessed with one SALU operation (read or swap) in one stage. Therefore, in our design, the switch locates the incoming ACK's ordered number in the multicast group and updates the recorded ACK number from the same receiver with the new value without reading it. Since the ePSN of the receiver QP is monotonously increasing, updating the ACK record from the same receiver without comparing it to the old value is always correct. Then the ACK number is compared to the recorded ACK numbers of other receiver QPs in the multicast group. If the ACK number is the smallest one, its SRC and QPN will be modified by the switch to match the verification on the sender. Later, this ACK packet will be forwarded to the sender. For example, for records $\langle ACK_1 = 4, ACK_2 = 2, ACK_3 = 5 \rangle$, the ACK number from the second receiver $ACK_2' = 4$ replaces the recorded ACK_2 without comparing and is compared with the other recorded ACK numbers $ACK_1 = 4, ACK_3 = 5$. Since it is the smallest ACK, $ACK = 4$ is forwarded to the sender QP.

Data isolation among pipelines. Today's programmable switches utilize multiple parallel pipelines to meet high packet processing rates [45]. However, since the switch hardware resources are isolated among ports on different pipelines, the processing logic, and metadata used in the procession of different pipelines are not shared [45]. Thus, stateful operations that cross pipelines are not able to implement in general. Although *MC-RDMA* merges ACKs that come from different ingress ports on different pipelines, all the ACKs are sent to the same port connected to the sender. Thus, *MC-RDMA* places the processing logic of ACK merging on the egress port to avoid data isolation. Making ACK aggregation on the same egress port is the other reason why we place ACK merging on the sender ToR switches.

NAK process. In RDMA, the receiver generates a NAK to the sender when it receives a packet with a larger PSN than expected. A NAK represents both a negative acknowledgment for the PSN it carries and *acknowledgments to previous PSNs*. Therefore, directly forwarding a NAK to the sender QP may mistakenly acknowledge some packets of other sub-flows in the same multicast group and the NAK should be handled carefully similar to the ACK. *MC-RDMA* modifies the number of the NAK to the current merged expected PSN (*i.e.*, $\min\{NAK - 1, ACKs\} + 1$) and forwards it to the sender QP. For example, for recorded ACKs $\langle ACK_1 = 1, ACK_2 = 2, ACK_3 = 5 \rangle$, $NAK_2 = 4$ will be modified to $NAK = \min\{3, 1, 2, 5\} + 1 = 2$ to reply to the sender QP. This modification keeps the correctness of the multicast protocol but may fall back more than the ideal, which causes some throughput loss. The maximal throughput loss caused by NAK fallback is determined by the differences of ACKs ($\max\{ACKs\} - \min\{ACKs\}$), which attributes to the RTT difference of the paths to receivers. However, the throughput loss caused by NAK fallback is endurable since packet loss is rare in today's datacenter and RTT difference is also small.

D. Congestion Control

Congestion signal aggregation. *MC-RDMA* utilizes the entire hardware implementation of the unicast-based RDMA congestion control (CC), *i.e.*, DCQCN. Since *MC-RDMA* bridges RDMA QPs without modifying the RDMA protocol on the endpoints, each part of unicast congestion control still works. The receiver RNIC generates CNPs according to the ECN marking on the packets and sends them back to the sender. Then the sender cuts the flow rate according to the DCQCN algorithm implemented on the RNIC. The difference between *MC-RDMA* CC and DCQCN is that it aggregates the CNPs from all the sub-flows in the same multicast group. Note that this may magnify the loss effect as all CNPs pass through the switch directly and may cause overly throughput reduction, which needs to be fixed in the future. However, the aggregated throughput of all sub-flows is still considerable (verified in §VII-C and §VII-D). All the CNPs from different multicast receivers will be modified on the source IP address (to make it valid to the sender QP) and forwarded to the sender.

The effect of multiple congested sub-flows. Since all sub-flows in a multicast group are synchronized, the most

severely congested sub-flow determines the flow rate of the whole group. However, because *MC-RDMA* aggregates all the CNPs from multiple sub-flows, the number of congested sub-flows affects the convergence flow rate. Because DCQCN is an Additive Increase Multiplicative Decrease algorithm, the CNP receiving rate of two flows at the same congestion point should be equal when the flow rates converge.

PFC configuration with packet replication. PFC reserves headroom buffer for each ingress port of switches to prevent buffer overflow caused by in-flight packets. Packets are replicated by the traffic manager in the switch, which sits between the ingress and egress ports. As a result, replication will not impact the ingress ports and no additional headroom buffer is required. Meanwhile, since an egress port can only receive a replicated packet when replication occurs, the egress buffer consumption is also mild for each egress port. Hence, the PFC configuration for *MC-RDMA* can be configured the same as that in the unicast scenario.

E. One-sided RDMA Operations

Memory address translation. Compared to two-sided RDMA *SEND* operation, one-side RDMA operations (*READ* and *WRITE*) that bypass remote host CPUs are preferred in RDMA systems to further decrease latency and CPU overhead. However, one-sided RDMA operations require the remote address of the memory region, which is generally different on different multicast receivers. In *MC-RDMA*, the programmable switches translate the memory address carried in the $RETH$ header of the first packet in each RDMA operation according to the memory address translation tables. The receivers register the memory region for RDMA and send the information of the memory (the beginning address and keys) to the sender. The sender gathers all the information of receivers and updates the memory address offset $offset$ to the local address and access key, $\langle offset, rkey \rangle$ of each receiver, to the control plane with algorithm 1. We use TCAM on the switches to support range-based match for memory address translation, thus the maximal number of terms of memory translation can be millions.

V. Discussion

In this section, we discuss some practical issues of the real deployment of RDMA multicast in datacenter, including deployment scale of programmable switches and scalability.

A. Deployment Scale of Programmable Switches

The minimal deployment prerequisite of *MC-RDMA* in RDMA clusters is that all the ToR switches are replaced with programmable switches in order to achieve packet multicast and ACKs merging. Bifurcation only occurs on the sender and receiver ToR switches, which is decided by the position of the receivers. The Bifurcation algorithm 1 should be modified to adapt to the situation where the Bifurcation occurs on a non-programmable switch. Compared to the network with sole programmable switches, this deployment is more feasible currently with the consideration of the cost and the switching rate of programmable switches. Though the bandwidth deduplication is not complete under this configuration, *MC-RDMA* still saves the bandwidth inside racks and CPU overhead.

When only deploying programmable switches at the top of each rack, the network multicast among leaf and spine switches can also be achieved by replacing the Bifurcation of programmable switches with IP multicast that is supported by non-programmable switches. Accordingly, the programmable switches and the Bifurcation algorithm should be adapted to be compatible with the IP multicast protocols. Besides, the IP multicast protocol must have a certain converge time when adding a multicast group to support the connection establishment of RDMA QPs. Some multicast routing algorithms, *e.g.*, flooding and pruning, do not have such property. Supporting existing IP multicast in *MC-RDMA* is our future work.

B. Scalability

The scalability of *MC-RDMA* refers to the ability to adapt to the number of members in a multicast group and the number of multicast groups in a storage cluster. In theory, the Intel Barefoot Tofino programmable switches can support at most 65535 members in each multicast group. However, as we referred in §VI-B, the implementation of ACK merging limits the number of receivers in the multicast group to 3 on Tofino (7 on Tofino II). Since the number of replications is commonly limited in distributed storage (*e.g.*, 2 or 3), the number of multicast members is not a resource bottleneck. With the limited number of replicas, the number of multicast groups is also limited in distributed storage systems. If only ToR switches are programmable switches, there are two types of bifurcation on the ToR switches in three-replica primary-backup replication: three replicas are under the same ToR switch or the primary is under a ToR switch while two backups reside in different ToR switches. Assumes that the distributed storage systems are deployed with $16N$ servers under N 32-ports ToR switches, 16 servers reside in a rack in this situation. The total number of bifurcation rules on each switch is $16N \times 16 \times 15/2 = 1920N$ (three replicas are in this rack) plus $16 \times 16N \times 16(N-1)/2 = 2048N(N-1)$ (three replicas are in different racks). Our implementation on Intel Barefoot Tofino can support at most ˜40000 multicast rules for three-replica multicast groups, which can support storage 64-80 hosts with *all possible replication patterns at the same time*. As mentioned in §II-C, since some distributed storage systems limit the number of replication group M on each host, the number of bifurcation rules in the second case is reduced from $2048N(N-1)$ to at most $16M$, in which case *MC-RDMA* can support more than 320 hosts.

VI. Implementation

A. Control Plane

We implement an RDMA library (including the CMD daemon) based on the communication library of Ceph [15] with ∼8000 lines of code in C++ and the control plane of programmable switches with ∼3000 lines of code in Python/C++. We use TCP as the out-of-band communication to exchange multicast group information and handshake messages.

Table configuration. On the switches, we use BFRuntime [41] to configure tables on the data path. The rules of ACK merging and NAK processing in §IV-C are implemented with static table configurations initialized at the start time

of the switches. The multicast and modification tables are configured at the time of connection establishment with the Bifurcation algorithm 1.

Multicast routing and ECMP support. The routing and bifurcation of multicast are built based on the unicast routing. On our implementation of the Barefoot Tofino switches, the routing table of unicast is configured on the ingress pipeline while multicast is configured in the switch Traffic Manager, thus the multicast and unicast can not share the same routing table. When the control plane of a router reports a routing change, it re-runs `Bifurcation` for all affected multicast groups and spread it to downstream switches.

Equal Cost Multi-path (ECMP) [43] is supported by data-center switches and used along with CLOS network topology. RDMA only supports flow-level ECMP since it only tolerates limited out-of-order packet delivery [37], [39], [40]. To support ECMP with *MC-RDMA*, the $NextHop$ in algorithm 1 can return sets of switches, thus the key bifurcation rule (line 6), should compare the sets of next hops rather than a single next hop. Only when the two switch sets are the same, the dst_i will be joined to the D_i. Otherwise, the new D_i will be constructed even if the two switch sets are partially equal, which is determined by the ECMP group (line 11–14). Later, the replication rules are added based on the D_{vv} set the same as before (line 16–24). Finally, the switch will send bifurcation inputs to all the next-hop switches in ECMP (line 31–32).

B. Data Plane

Our implementation is on Barefoot Tofino programmable switches with ∼900 lines of code in P4.

Packet replication and ECMP support. The Barefoot Tofino supports two-level replication. The first-level replication engine replicates the original packets into packets with different Receiver IDs (RIDs). The second-level replication replicates packets with different RIDs into different ports. At each level of replication, a hash function can be configured to randomly select members in this replication. We implement multicast at the first-level replication and implement ECMP at the second level of replication.

ACK merging. Our implementation on the Barefoot Tofino switch supports at most 3 members in each multicast group because of the limitation of processing stages of programmable switches. Barefoot Tofino only has 12 stages, while adding each group member in a multicast group adds 2 additional stages for ACK merging procession. The number of the maximal multicast groups can be increased to 7 on the Barefoot Tofino II switch since it has 20 stages. 2–3 multicast members are enough for replication in distributed storage systems.

VII. Evaluation

In this section, we investigate the performance of *MC-RDMA* from the following perspectives: the basic performance gain of *MC-RDMA* (§VII-B), the effectiveness of congestion control (§VII-C), the endurance to packet drop (§VII-D), performance with realistic workloads (§VII-E), and performance on distributed training system (§VII-F). We use both a real testbed and simulations (NS-3 [1]) in the evaluation. Fanout and chain replication (CR) are developed for comparison.

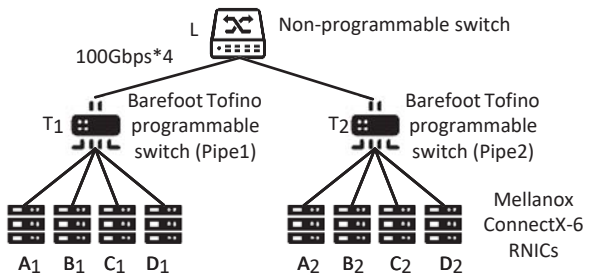


Fig. 3. Testbed topology.

A. Experiment Setups

Configurations. Figure 3 shows the topology used in our testbed and simulation. 8 servers with 100Gbps Mellanox ConnectX-6 RNICs (A_1–D_2) are connected to a two-level network. There are 4 100Gbps links connected from each Barefoot Tofino switch to the leaf non-programmable switch to ensure no blocking inside the network. The leaf switch is a non-programmable switch.

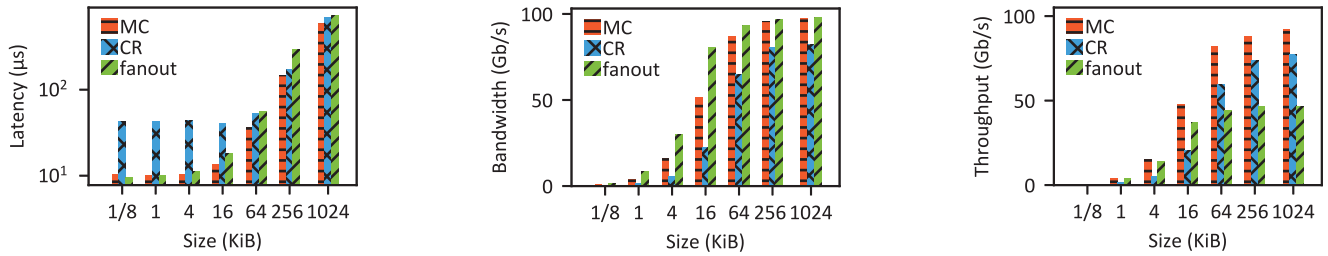
In the experiments, we bind the sending thread (*i.e.*, for posting requests) and the RDMA thread (*i.e.*, for polling and processing completions) on the client to two CPU cores of the same NUMA node. The reason for restricting the CPUs on the same NUMA node is to accelerate the access of the two threads to the same data in the send completions.

B. Single-client Latency and Throughput

As a basic performance validation, we measure the performance of *MC-RDMA* and other unicast-based replication protocols (fanout and CR) in a three-replica primary-backup replication in different racks ($A_1 \rightarrow D_1, A_2$). Figure 4(a) shows the latencies of different request sizes from 128B to 1024KiB with IO depth (the number of in-flight requests) 8. For small request sizes, CR has longer latencies because it costs more replication hops to replicate the data while *MC-RDMA* and fanout only need one RTT. However, for large requests, the latency of fanout replication increases significantly compared to *MC-RDMA* and CR since it costs twice the bandwidth of the sender to send data to different receivers. Figure 4(b) shows the sender RNIC bandwidth consumption in the same test. The bandwidth consumption of fanout is twice of the others until it saturates the link rate. *MC-RDMA* consumes higher bandwidth compared to CR with different request sizes since it provides accordingly larger requests throughput with smaller request completion time, which is shown in Figure 4(c). Note that the latency of *MC-RDMA* in the experiment is always the shortest among all the methods. This is because the CPU usage of *MC-RDMA* is more than 2× effective with fewer request posting, completion polling, and cross-connection data access. Due to the multicast interface and semantic, *MC-RDMA* only operates on exactly one QP, resulting in less software processing overhead.

C. Congestion Control

To validate the effectiveness of congestion control among multicast and unicast flows, we start 3 flows (multicast flow f_{m1}: $A_1 \rightarrow C_1, D_1$, f_{m2}: $C_1 \rightarrow D_1, A_2$, and unicast flow f_{u1}: $D_1 \rightarrow A_2$) separately. Figure 5 shows the throughput of each flow in testbed. The rates of flow f_{m1} decrease to $1/2$ and



(a) Single-client latency (testbed). (b) Single-client bandwidth (testbed). (c) Single-client throughput (testbed).

Fig. 4. Performance of *MC-RDMA*, CR and fanout in single-client tests.

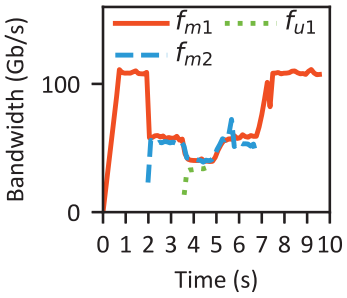


Fig. 5. Fairness of congestion control among *MC-RDMA* and RDMA unicast flows (testbed).

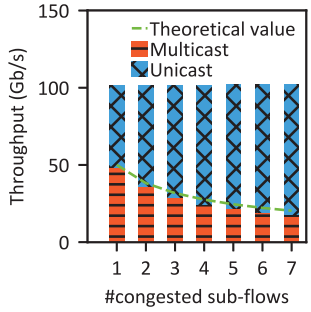
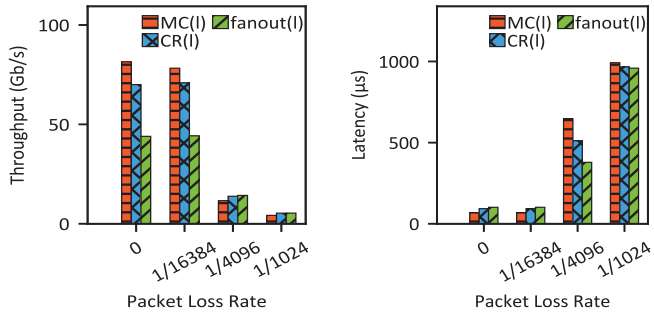


Fig. 6. The converged rate of *MC-RDMA* with the varied number of congested subflows (simulation).



(a) Throughput (testbed). (b) 99^{th} Latency (testbed).

Fig. 7. Throughput and latency of *MC-RDMA*, CR, and fanout with packet loss on the long path (l).

$1/3$ when f_{m2} and f_{u1} start respectively, which reveals that multicast flows can share bandwidth fairly with other flows.

As we discussed in §IV-D, the converged rate of a congested multicast flow is affected by the number of congested sub-flows. To verify our conclusion, we initiate a multicast flow with 7 receivers $A_1 \rightarrow B_1, ..., D_2$ and observe the flow rate with different numbers of congested sub-flows in simulation. For example, with the configuration of 2 congested sub-flows, there are two unicast flows $C_1 \rightarrow B_1$, $D_1 \rightarrow C_1$ that compete with the multicast flow on the downlinks to B_1 and C_1. Figure 6 shows the throughput of the multicast and unicast flows with 1–7 congested sub-flows with the DCQCN configuration for 100Gbps [49] ($K_{min} = 50KB$, $K_{max} = 500KB$, $P_{max} = 0.01$). The theoretical values are the predicted flow rate according to our discussion in §IV-D, which conforms to the simulation results.

D. Packet Loss

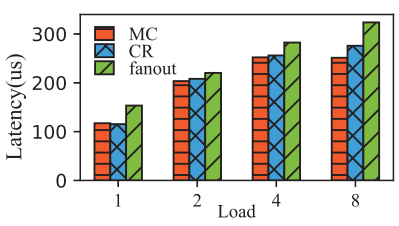
To validate the correctness and effectiveness of the reliability of *MC-RDMA*, we test with manual configuration of packet drop on the switch. In this experiment, there exists a multicast flow from A_1 to B_1, A_2. We compare the throughput and latency of different methods, by tuning the packet loss rates on the long path ($A_1 \rightarrow A_2$) in testbed. Figure 7(a) and 7(b) show the test results with different packet loss rates on the long path. In general, the results demonstrate that the throughput loss of *MC-RDMA* with packet loss is similar to CR and fanout, which is acceptable since packet loss in the datacenter is usually rare. Because of the NAK fallback mechanism in *MC-RDMA* (§IV-C), a larger packet loss rate can result in a slightly larger performance penalty. For instance, with loss rates $1/4096$ and $1/1024$, the throughputs of *MC-RDMA* degrade more than those of the others, but the differences are still negligible.

E. Distributed Storage System Workloads

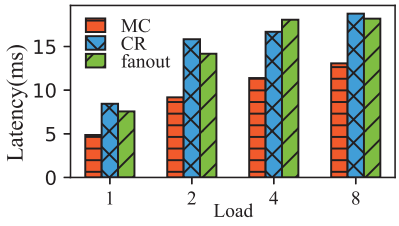
To investigate the performance gain of *MC-RDMA* compared to unicast RDMA in distributed storage systems, we test *MC-RDMA*, CR, and fanout replication with two real workloads, *i.e.*, the storage workload from Alibaba cloud [35] and the web search workload released by the previous work [3], in our large-scale simulation.

We use a 3-layer fat-tree network topology ($k = 6$), consisting of 54 servers, to reveal the performance of all methods at a large scale. Each server has a 100Gbps access link and each ToR switch connects 9 servers in a rack. Each link has a propagation latency of $500ns$ and the maximum end-to-end round-trip latency is $6\mu s$. 27 servers of the first three racks are clients and the remaining 27 servers are storage nodes. For *write* operation, a client first writes a message to a primary storage node randomly, and then the primary storage node replicates it to two other replicas in a multicast group. For *read* operation, a client reads a message from one of the replicas in a multicast group. The request message size is generated according to the above two realistic workloads. Each request is generated in a Poisson process and we vary the inter-arrival time of requests to form different levels of load. The proportion of the read and write requests is 1:1.

As depicted in Figure 8(a) and Figure 8(b), the average latency of *read* operation of *MC-RDMA* is always the shortest with different network loads. Specifically, the performance gap between *MC-RDMA* and the others becomes larger with the increase of the network load. This is because, with a larger network load, network congestion occurs more frequently. The saved bandwidth of *MC-RDMA* in the network can mitigate the congestion and is used to accelerate the transmission from the storage nodes to the client nodes, resulting in the shortest latency. Moreover, we observe that the latency decrease of web search workload is more obvious (~30% latency decrease



(a) Alibaba Storage Workload (simulation).



(b) Web Search Workload (simulation).

Fig. 8. The average latency of *read* operation of *MC-RDMA*, CR and fanout in a $k = 6$ fat tree.

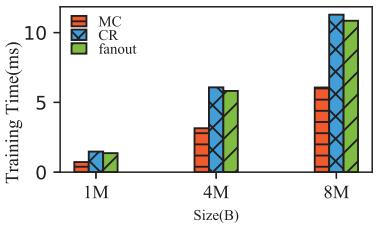


Fig. 9. The average training time for each iteration with *MC-RDMA*, CR, and fanout (simulation).

compared to the others with network load 8). We attribute this to the much larger message size of the Web Search workload and the network link will be saturated easier with this workload. Besides, we test the performance with the ToR-only and full network support for multicast (discussed in §V-A), using the Alibaba Storage workload, respectively. The results show that the reduced average latency with full network support can vary from 33% to 47% with different network loads, compared to the ToR-only support. Note that the latency of *write* operation of all methods is similar, and we omit it for page limit.

F. Distributed Machine Learning Scenario

To show that *MC-RDMA* has the potential to accelerate more data-intensive applications with data replication, we simulate the distributed machine learning system with the same setup in §VII-E, except for the communication pattern among nodes and the message generation process.

We adopt the parameter server (PS) training framework to simulate the training process of distributed machine learning. In the PS framework, a parameter server aggregates gradients from a group of workers, averages the aggregated gradients, performs stochastic gradient descent to update the model, and distributes the model to workers, iteratively. This process can be simplified as the iterative collection of multiple fix-sized messages from a group of workers and the delivery of the message back to the workers. In our simulation, 5 servers in a pod are used as the parameter servers and each parameter server communicates with 6 servers in other pods regularly. The messages in this simulation are of large sizes, *i.e.*, 1MB, 4MB, and 8MB. This is because of the popular large model, which is also confirmed by the related work [28] (the model is partitioned into multiple 4MB messages).

As shown in Figure 9, *MC-RDMA* achieves the shortest training time in an iteration, with all message sizes. Specifically, with a larger message size, the reduction of the training time is more significant. For example, with 8MB message size, the training time of *MC-RDMA* is almost half of the others, *i.e.*, 6.1 versus 11.3 and $10.8ms$. This is because a larger message can cause more severe congestion, which lags the gradient distribution process dramatically. Therefore, *MC-RDMA* can obtain better performance with a larger message size and a larger scale of the training cluster, which is in agreement with the trend of the development of machine learning.

VIII. Related Work

IP-level Multicast Solutions. Traditional multicast protocols [13], [16], [19], [20] are designed for large object delivery based on IP-level multicast. Thus these multicast protocols require high scalability to the number of receivers. For example, the ACK implosion problem [19], [20] claims that since there can be as many acknowledgments as the number of receivers, the efficiency gained by the use of the broadcast of the network is negated in processing a large number of acknowledgment packets [20]. Thus protocols are designed as NAK-based [19], receiver-driven [13], [20], with the help of in-network cache [13], or with novel labeling mechanisms [16]. On the contrary, *MC-RDMA* is a transport-layer multicast protocol that is highly compatible with an existing unicast protocol, *i.e.*, RDMA. Besides, it is designed for datacenter distributed storage systems that have a small and fixed number of members in a multicast group.

RDMA-based Multicast Solutions. Existing RDMA-based multicast solutions are not designed to cooperate with in-network switches sophisticatedly [9], [26], [27], [31], [33]. [26], [27], [33] use RDMA to boost the atomic multicast in the distributed systems, which guarantees the order of a group of message deliveries to multiple receivers. [9] uses the chain replication, which involves an RNIC in each node along the chain. The above approaches do not take advantage of the in-network switches to save the scarce bandwidth of the access link. [31] explores the one-sided RDMA multicast in a centralized way. It relies on a centralized coordinator to configure the programmable switches globally. In contrast, *MC-RDMA* is a fully distributed approach and tackles unique challenges, such as congestion control and routing algorithm.

IX. Conclusion

MC-RDMA is a distributed and reliable multicast RDMA system that supports multicast message sending and memory operations. It utilizes programmable switches to achieve in-network reliable RDMA multicast, which reduces bandwidth waste and CPU overhead in replication. Moreover, *MC-RDMA* is highly compatible with current RDMA hardware and is handy for current RoCEv2 deployment.

X. Acknowledgement

We thank our shepherd Kai Chen and the anonymous reviewers for their valuable suggestions. This project is partially supported by the National Key Research and Development Program of China under Grant Number 2022YFB2901502, the National Natural Science Foundation of China under Grant Numbers 62325205, 62072228 and 62172204, and the Jiangsu Funding Program for Excellent Postdoctoral Talent.

REFERENCES

[1] Ns-3 network simulator. https://www.nsnam.org/, 2022.

[2] Marcos K. Aguilera, Kimberly Keeton, Stanko Novakovic, and Sharad Singhal. Designing far memory data structures: Think outside the box. In *ACM HotOS*, page 120–126, 2019.

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *ACM SIGCOMM Computer Communication Review*, page 63–74, 2010.

[4] Sérgio Almeida, João Leitão, and Luís Rodrigues. Chainreaction: A causal+ consistent datastore based on chain replication. In *ACM EuroSys*, page 85–98, 2013.

[5] Amazon. Amazon AWS. https://aws.amazon.com/, 2022.

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

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

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

[9] Jonathan Behrens, Sagar Jha, Ken Birman, and Edward Tremel. Rdmc: A reliable rdma multicast for large objects. In *IEEE DSN*, pages 71–82, 2018.

[10] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, pages 87–95, 2014.

[11] Matthew Burke, Sowmya Dharanipragada, Shannon Joyner, Adriana Szekeres, Jacob Nelson, Irene Zhang, and Dan R. K. Ports. Prism: Rethinking the rdma interface for distributed systems. In *ACM SOSP*, page 228–242, 2021.

[12] Apache Caasandra. Apache Cassandra Data Replication. https://docs.datastax.com/en/archived/cassandra/2.0/cassandra/architecture/architectureDataDistributeReplication_c.html.

[13] Jiaxin Cao, Chuanxiong Guo, Guohan Lu, Yongqiang Xiong, Yixin Zheng, Yongguang Zhang, Yibo Zhu, and Chen Chen. Datacast: A scalable and efficient reliable group data delivery service for data centers. In *ACM CoNEXT*, page 37–48, 2012.

[14] Ceph. Ceph placement groups. https://docs.ceph.com/en/quincy/rados/operations/placement-groups/placement-groups, 2014.

[15] Ceph. Ceph open-source code. https://github.com/ceph/ceph, 2022.

[16] Khaled Diab, Parham Yassini, and Mohamed Hefeeda. Orca: Server-assisted multicast for datacenter networks. In *USENIX NSDI*, pages 1075–1091, 2022.

[17] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. No compromises: Distributed transactions with consistency, availability, and performance. In *ACM SIGOPS*, page 54–70, 2015.

[18] Zhuohui Duan, Haodi Lu, Haikun Liu, Xiaofei Liao, Hai Jin, Yu Zhang, and Song Wu. Hardware-supported remote persistence for distributed persistent memory. In *ACM SC*, 2021.

[19] A. Erramilli and R. P. Singh. A reliable and efficient multicast for broadband broadcast networks. In *ACM SIGCOMM*, page 343–352, 1987.

[20] Sally Floyd, Van Jacobson, Steve McCanne, Ching-Gung Liu, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *ACM SIGCOMM*, page 342–356, 1995.

[21] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SOSP*, pages 29–43, 2003.

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

[23] IEEEStd. Ieee.802.11qbb. priority based flow control, 2011.

[24] INTEL. Opentofino. https://github.com/barefootnetworks/Open-Tofino, 2020.

[25] Salekul Islam and J. Willam Atwood. The internet group management protocol with access control (igmp-ac). In *IEEE LCN*, pages 475–482, 2006.

[26] Joseph Izraelevitz, Gaukas Wang, Rhett Hanscom, Kayli Silvers, Tamara Silbergleit Lehman, Gregory Chockler, and Alexey Gotsman. Acuerdo: Fast atomic broadcast over rdma. In *ACM ICPP*, 2023.

[27] Sagar Jha, Lorenzo Rosa, and Ken Birman. Spindle: Techniques for optimizing atomic multicast on rdma. In *IEEE ICDCS*, pages 1085–1097, 2022.

[28] 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 *USENIX OSDI*, pages 463–479, 2020.

[29] Anuj Kalia, David Andersen, and Michael Kaminsky. Challenges and solutions for fast remote persistent memory access. In *ACM SoCC*, page 105–119, 2020.

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

[31] Xin Zhe Khooi, Cha Hwan Song, and Mun Choon Chan. Towards a framework for one-sided rdma multicast. In *ACM ANCS*, page 129–132, 2021.

[32] Daehyeok Kim, Amirsaman Memaripour, Anirudh Badam, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Shachar Raindel, Steven Swanson, Vyas Sekar, and Srinivasan Seshan. Hyperloop: Group-based nic-offloading to rate replicated transactions in multi-tenant storage systems. In *ACM SIGCOMM*, page 297–312, 2018.

[33] Long Hoang Le, Mojtaba Eslahi-Kelorazi, Paulo Coelho, and Fernando Pedone. Ramcast: Rdma-based atomic multicast. In *ACM Middleware*, page 172–184, 2021.

[34] Dan Li, Jianping Wu, Ke Xu, Yong Cui, Ying Liu, and Xiaoping Zhang. Performance analysis of multicast routing protocol pim-sm. In *IEEE AICT/SAPIR/ELETE*, pages 157–162, 2005.

[35] 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 *ACM SIGCOMM*, 2019.

[36] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. Octopus: an RDMA-enabled distributed persistent memory file system. In *USENIX ATC*, pages 773–785, 2017.

[37] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting network support for rdma. In *ACM SIGCOMM*, page 313–326, 2018.

[38] MongoDB. Chain Replictaion in MongoDB. https://www.mongodb.com/docs/manual/tutorial/manage-chained-replication/, 2022.

[39] NVIDIA. Mellnox connectx-5. https://www.nvidia.com/en-us/networking/ethernet/connectx-5/, 2022.

[40] NVIDIA. Mellnox connectx-6 dx. https://www.nvidia.com/en-us/networking/ethernet/connectx-6-dx/, 2022.

[41] ONF. P4 runtime. https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html, 2017.

[42] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang. The ramcloud storage system. *ACM Transactions on Computer Systems*, 33, 2015.

[43] Fiqih Rhamdani, Novian Anggis Suwastika, and Muhammad Arief Nugroho. Equal-cost multipath routing in data center network based on software defined network. In *IEEE ICoICT*, pages 222–226, 2018.

[44] Mahadev Satyanarayanan and Ellen H. Siegel. Parallel communication in a large distributed environment. *IEEE transactions on computers*, 39:328–348, 1990.

[45] Vishal Shrivastav. Stateful multi-pipelined programmable switches. In *ACM SIGCOMM*, page 663–676, 2022.

[46] Yacine Taleb, Ryan Stutsman, Gabriel Antoniu, and Toni Cortes. Tailwind: Fast and atomic RDMA-based replication. In *USENIX ATC*, pages 851–863, 2018.

[47] Yan Wang and Jun-Hui Zheng. A new type of secure multicast based on source tree. In *IEEE ICWAMTIP*, pages 216–219, 2012.

[48] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *USENIX OSDI*, pages 307–320, 2006.

[49] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *ACM SIGCOMM*, page 523–536, 2015.

[50] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. Understanding and mitigating packet corruption in data center networks. In *ACM SIGCOMM*, page 362–375, 2017.