MCTCP: Congestion-Aware and Robust MultiCast TCP in Software-Defined Networks

Tingwei Zhu[†], Fang Wang[†*], Yu Hua[†], Dan Feng[†*], Yong Wan[‡], Qingyu Shi[†], Yanwen Xie[†]
[†]Wuhan National Lab for Optoelectronics
School of Computer, Huazhong University of Science and Technology, Wuhan, China
[‡]Computer Engineering College, Jingchu University of Technology
[*]Corresponding author: {dfeng, wangfang}@hust.edu.cn

Abstract—**Continuously enriched distributed systems in data centers generate much network traffic in push-style one-to-many group mode, raising new requirements for multicast transport in terms of efficiency and robustness. Existing reliable multicast solutions, which suffer from low robustness and inefficiency in either host-side protocols or multicast routing, are not suitable for data centers. In order to address the problems of inefficiency and low robustness, we present a sender-initiated, efficient, congestion-aware and robust reliable multicast solution mainly for small groups in SDN-based data centers, called MCTCP. The main idea behind MCTCP is to manage the multicast groups in a centralized manner, and reactively schedule multicast flows to active and low-utilized links, by extending TCP as the host-side protocol and managing multicast groups in the SDN-controller. The multicast spanning trees are calculated and adjusted according to the network status to perform a better allocation of resources. Our experiments show that, MCTCP can dynamically bypass the congested and failing links, achieving high efficiency and robustness. As a result, MCTCP outperforms the state-of-the-art reliable multicast schemes. Moreover, MCTCP improves the performance of data replication in HDFS compared with the original and TCP-SMO based ones, e.g., achieves 101% and 50% improvements in terms of bandwidth, respectively.**

I. Introduction

In recent years, with the development of cloud computing technology, applications in data centers have been significantly enriched. A large number of different distributed applications generate complex network traffic of one-to-one and one-to-many patterns, causing much pressure on data center network resources. More importantly, most of the one-to-many group communications in data centers are implemented through multiple unicast like TCP, which is inefficient. They generate a lot of replicated traffics, which not only waste network resources but also decrease the performance of applications.

A large number of typical one-to-many group communication scenarios exist in data centers, as a lot of distributed systems need to transfer the same data from one node to others for the sake of service reliability and performance. Distributed file systems adopt a replication mechanism to ensure the reliability of data storage, such as HDFS [1] in Hadoop, Ceph [2] in Red Hat and GFS [3] in Google. File chunks are replicated to several storage nodes, which are chosen by certain placement policies. In cooperative computations, the executable binaries or shared data are distributed to other collaborative servers, such as the DistributedCache in Hadoop MapReduce [4] and Broadcast variables in Apache Spark [5]. The queries in web search engine are redirected to a set of indexing servers to look up the matching documents, say Google or Bing.

These group communication scenarios in data centers have the following key characteristics, which raise new requirements for multicast solutions.

- **Small groups.** The group members are generally small, i.e., hundreds or fewer. For instance, distributed file systems mainly adopt three replicas, and the number of the worker nodes in data analytics applications is often from tens to hundreds [6].
- **Reliability.** Unlike the traditional group communication scenarios such as IPTV, which allow data loss to some extent, most of the group communications in data centers require strict reliability.
- **Sender-initiated.** Most of the group transmissions are push-style, where the sender determines the transmission, and the receivers do not know when and where to receive data in advance.
- **Efficiency.** Compared with the Internet environment, the DCN (Data Center Network) has the salient features of high bandwidth and low latency features and the applications in data centers are more performance sensitive and critical. Therefore, multicast schemes need to make full use of the benefits of DCN to achieve high efficiency. Moreover, due to the burst [7] and mixed nature of network traffic in data centers, congestion-awareness is very important in achieving efficient multicast routing.
- **Robustness.** A link failure may cause transmission pause in multiple receivers, and therefore, robust is important in multicast.

Previous reliable multicast solutions fail to meet all the requirements in aforementioned small groups multicast scenarios, mainly for the following reasons. First, the majority of previous reliable multicast solutions are receiver-initiated application-layer protocols (based on UDP), which suffer from high software overhead on end hosts and mismatch to the sender-initiated mode. Second, traditional IP multicast routing algorithms, such as PIM-SM [8], are not designed to build optimal routing trees. They are not aware of link congestion, and thus apt to cause significant performance degradation in burst and unpredictable traffic environment [7]. Third, traditional multicast group management protocols, such as Internet Group Management Protocol (IGMP) [9], fail to be aware of link failures. A failure in multicast spanning trees can suspend transmission and lead to significant performance loss or business interruption.

The emergence of SDN (Software-Defined Networking) [10], brings new ideas for solving routing efficiency issues of reliable multicast in data centers. A centralized control plane called SDN-controller provides global visibility of the network, rather than localized switch level visibility in traditional IP networks. Therefore, multicast routing algorithms can leverage topology information and link utilization to build optimal (near-optimal) routing trees, and be robust against link congestion and failures.

To meet all the aforementioned requirements, we develop an SDN-based sender-initiated, efficient, congestion-aware and robust reliable multicast solution, called MCTCP, which is mainly designed for small groups. The main idea behind MCTCP is to manage the multicast groups in a centralized manner, and reactively schedule multicast flows to active and low-utilized links. Therefore, the multicast routing can be efficient and robust. To eliminate the high overhead on end hosts and achieve reliability, we extend TCP as the host-side protocol, which is a transport-layer protocol.

Specifically, MCTCP consists of two modules, including the HSP (Host-Side Protocol) and the MGM (Multicast Group Manager). The HSP is a sender-initiated protocol, where the sender defines the transmission and the receivers need not to know the multicast address or subscribe it in advance. By notifying the MGM each time establishing or closing a session, it is easy for the MGM to keep states of all the sessions. Therefore, the MGM can calculate and adjust Multicast Spanning Trees (MSTs) for each session based on real-time link status to achieve congestion-aware and robustness.

As for the access bottleneck and single point failure problems which centralized approaches may suffer from, we have two considerations. First, the availability of the SDN-controller is out of the scope of this paper, and we can use multiple controllers to relieve these problems. Second, we can significantly relieve the pressure of the SDN-controller by keeping long term connections when using MCTCP, which is feasible in most of the bandwidth-hungry applications such as HDFS.

Our design goal is to make MCTCP as flexible and convenient as TCP, efficient and robust for one-to-many small group communications, even in burst and unpredictable traffic environments. To verify the applicability of MCTCP, we also implement multicast-based HDFS, which is a version of HDFS using MCTCP for data replication.

This paper makes the following contributions.

- We propose MCTCP, a transport-layer reliable multicast transmission scheme mainly for small groups in SDN-based data centers, which is efficient on both host-side protocol and multicast routing. We design a centralized Multicast Group Manager (MGM) to ensure multicast routing efficiency and robustness by reactively scheduling multicast flows. Therefore, the MSTs can dynamically bypass the congested and failing links, making MCTCP more suitable for the unpredictable network environment.
- We implement MCTCP in real systems. Experimental results confirm its functionality of congestion-awareness and failure-resistance. Experiments under background traffic in patterns of two realistic workloads, including the web search [11] and the data mining [12], demonstrate that MCTCP outperforms the state-of-the-art reliable multicast schemes in transmission bandwidth.
- We implement the multicast version of HDFS using MCTCP and TCP-SMO [13], called HDFS-M and HDFS-T, respectively to improve performance of data replication. Compared with HDFS-T and HDFS-O (the original pipeline-based HDFS version), HDFS-M decreases the per-packet latency by ∼13%-47% and ∼70%-83%, and improves the throughput by ∼20%-50% and ∼30%-101%, respectively under web search background traffic.

The rest of paper is organized as follows. Section II presents the motivation of this paper. Section III presents the design of MCTCP. Section IV describes the implementation details. Section V describes our experimental evaluation of MCTCP and the performance comparisons among the state-of-the-art approaches. Section VI presents the related works. Finally, we draw conclusion in Section VII.

II. MOTIVATIONS

A large number of small group mode communications exist in DCN, which are widely presented in distributed systems. These small group communications carry a large amount of data, raising new challenges for reliable multicast schemes.

Applicability and Flexibility: Many group communications are generated in distributed systems during runtime. For each group communication, the sender knows the information of all receivers, while the receivers are not aware of the sender until the communication starts. For example, in distributed storage systems, the clients are active transmitters and the data nodes are passive receivers during data replication. Therefore, in these scenarios, the sender-initiated multicast schemes have better applicability and flexibility than the receiver-initiated schemes. But most conventional multicast schemes are receiver-initiated, which are not suitable for these scenarios.

Efficiency: High efficiency is required in data center networks. But existing reliable multicast schemes cannot meet the efficiency requirements for most of the applications for two reasons. **First**, the software overhead on host side protocols of multicast becomes prominent in the high bandwidth, low latency network environment of data centers. The majority of existing reliable multicast solutions are application-layer schemes, which are UDP-based and implemented in user space, thus resulting in poor efficiency. **Second**, current data centers are built with high link density, and the network traffic is bursty and unpredictable [7]. Since existing reliable multicast schemes are based on distributed routing algorithms, which cannot make full use of network resources to achieve

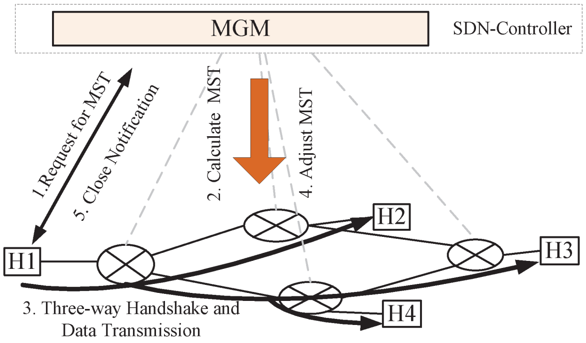


Fig. 1: Illustration of MCTCP.

optimal routing efficiency, they are extremely vulnerable to network congestion, and easy to cause significant performance degradation.

Robustness: With the increasing size of data centers, failures frequently occur [12] [14]. Traditional multicast management protocols are not aware of link failures, which generally consume 10-60 seconds (depending on the query interval) to detect. Any link failure in multicast trees can lead to significant performance loss.

To address the challenges above, we design MCTCP to achieve straightforward deployment, reliability, efficiency and TCP-friendliness by extending TCP as the host-side protocol. By leveraging the centralized control and global view of SDN, MCTCP can calculate and adjust the MST of each group based on the real-time link status to achieve efficient routing and robustness.

III. MCTCP Design

MCTCP consists of two modules, i.e., the HSP (Host-Side Protocol) and the MGM (Multicast Group Manager). The HSP is an extension of TCP, leveraging the three-way handshake connection mechanism, cumulative acknowledge mechanism, data retransmission mechanism and congestion control mechanism to achieve reliable multipoint data delivery. The MGM, located in the SDN-controller, is responsible for calculating, adjusting and maintaining the MSTs for each multicast session. It keeps monitoring the network status (e.g. link congestion and link failures) and creates maximal possibility for MCTCP to avoid network congestion and to be robust against link failures.

The schematic of MCTCP is shown in Fig. 1. The sender establishes connection with multiple receivers explicitly before data transmission. First, the sender requests to the MGM for calculating the MST. Second, the MGM calculates and installs the MST. Third, the sender starts three-way handshake with receivers, and begins data transmission after that. Fourth, the MGM will adjust the MST once link congestion or failures are detected. Fifth, the sender notifies the MGM after data transmission finishes.

A. Host-Side Protocol

1) Session Establishment: The sender requests to MGM for calculating MST when establishing a new session. Since the receivers do not obtain the multicast address in advance, the first handshake must be realized by using unicast address. We put the multicast address in the SYN packet (in the TCP options field). After receiving the SYN packet, the receivers get the specific multicast address, and join the group (just put the multicast address into the interested list, but not send IGMP messages), so that they can receive the multicast messages.

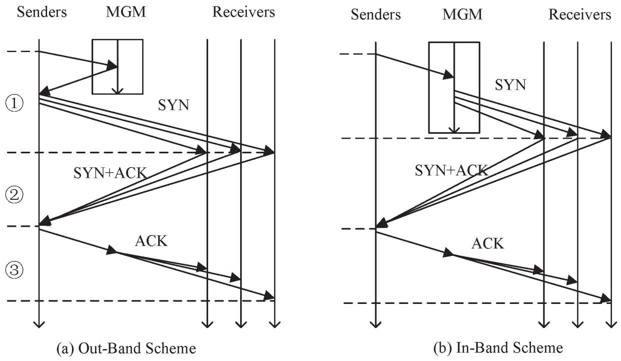


Fig. 2: The Procedure of MCTCP Session Establishment with three receivers.

There are two alternative schemes, the out-band and the in-band schemes. For the out-band scheme, the sender requests to the MGM before three-way handshake. After calculating the MST, the MGM notifies the sender to start three-way handshake. For the in-band scheme, the SYN packet is reused to request MST for calculation, and redirected to the MGM. After receiving the SYN packet and calculating MST, the MGM dispatches the SYN packet to all the receivers in unicast. Fig. 2 illustrates the procedure of connection establishment.

The out-band scheme suffers from time overhead of an extra RTT to controller. Hence, this scheme is suitable for the large amount data transmission scenes, in which the overhead of session establishment is negligible. The in-band scheme has no extra time overhead, but brings much pressure on the SDN controller. This scheme is more suitable for extremely small membership and delay-sensitive scenes.

2) Data Transmission: When a session is established, data transmission begins.

Packet Acknowledgement. The sender maintains a sliding window and processes the acknowledgement from receivers. The send window advancement is decided by the slowest receiver. As MCTCP is mainly designed for small group scenarios, the ACK-implosion problem existed in traditional large member reliable multicast could be ignored.

Packet Retransmission. The sender manages a timer for each session, and will retransmit the packets in multicast if the timer expires or packets loss is detected. Since the efficient and robust multicast forwarding achieved by MGM can significantly reduce the packet loss, the emergence of retransmission in MCTCP will be largely decreased.

Congestion Control. We use existing congestion mechanisms in TCP directly, so that we can make full use of the existing rich and mature congestion algorithms, evolving along with TCP. Since the send window advancement is decided by the

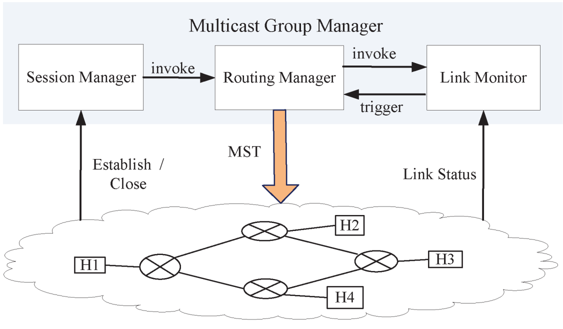


Fig. 3: The Multicast Group Manager.

slowest receiver, the congestion status of the whole session will be determined by the most congested receivers.

Node failure. If no acknowledgement is received from a certain receiver in a threshold time during data transmission, we consider the receiver fails. The failing receiver, which may encounter crash or network failure, should be cleaned out from the multicast session in order to ensure the transmission of the rest receivers. Therefore, the applications should be responsible for fault recovery.

3) Session Close: After data transmission is completed, the sender closes the multicast session initiatively, and then notifies the MGM.

B. Multicast Group Manager

MCTCP uses a logically centralized approach to manage multicast groups. The MGM located in SDN controller manages the multicast sessions and MSTs. By keeping the global view of the network topology and monitoring the link status in real-time, the MGM can adjust the MSTs in case of link congestion or failures. Specifically, the MGM consists of three sub-modules, including the session manager, the link monitor and the routing manager, as shown in Fig. 3.

1) Session Manager: The session manager is responsible for maintaining the states of all groups. When establishing or closing a multicast session, the sender informs the session manager. Hence, the session manager can keep track of all the active multicast sessions. If a multicast session is closed, the MST will not be cleared immediately, but just be marked inactive. Therefore, a session with the same sender and receivers can reuse the MST. The session manager periodically cleans up the inactive MSTs.

2) Link Monitor: Link Monitor is responsible for monitoring network link status, and estimating the weight of each link periodically.

The primary challenge here is to estimate weight of each link at low overhead. We focus on the full-duplex network where all the links have the same bandwidth B. Assume we measure M bytes transferred over a link within interval Δt, and then the measured traffic rate R is $M/\Delta t$, and the measured weight is $W_m = R/B$. If W_m<1, it means the traffic rate has not reached the link capacity. In this case, the measured rate indicates the real load of the link, and the $weight = W_m$. If $W_m = 1$, it means the traffic rate reaches the roof of the link, but the real load may be heavier than the measured one. Observing that when the rate reaches the link capacity, more flows could incur heavier load. Therefore, in this case, we need to estimate the link weight based on the number of flows. To this end, we introduce a parameter F, which represents the rate of a single flow. Hence, n flows can reach the maximum traffic rate $n \cdot F$ without the limitation of link capacity, and the corresponding $weight = n \cdot F/B$. In practice, the measured rate R may not reach the link capacity B precisely, but can only be approximate to B. Suppose $R = \alpha \cdot B$, if α is larger than a threshold η, such as 95%, we consider it has been reaching the link capacity. We summarize the equation for weight calculation as follow.

$$weight = \begin{cases} R/B & R < \eta \cdot B \\ n \cdot F/B & R \geq \eta \cdot B \end{cases} \quad (1)$$

As Eq. 1 shows, we need to measure the per-link rate R, the flow number of each link n and the single flow rate F in the network. To measure R, we simply poll the port status of all the switches, such as using the *'get_port_stats'* function in OpenFlow. To measure n, we keep the state of all the flows, by recording a flow when initiated and removing it when finished. We can prevent polling the flow tables from switches by enabling the *'Flow Removed'* message when a flow expires in the OpenFlow-based SDN networking. The F actually means how much load a single flow can introduce. It is a network parameter which can be configured statically based on experience or can be easily estimated during runtime.

3) Routing Manager: The routing manager is responsible for calculating and adjusting MSTs. When establishing a new multicast session, the routing manager calculates the minimum cost MST based on the current link utilization. When a link overloads or a failure occurs, the adjustment for all MSTs over the link will be triggered. We divide the routing manager into two parts, the routing calculation and the routing adjustment. The MST should be calculated quickly during session establishment. In the case of link congestion, the MST should be adjusted in the best-effort way. When the link fails, all the relevant MSTs should be quickly updated.

Routing calculation. The members of a group are assigned by the sender, and no dynamically join/leave is allowed in MCTCP once the session begins. We use minimum-cost path heuristic algorithm (MPH) [15] for routing calculation. The MPH algorithm inputs a set of sender/receiver nodes and all-pair shortest paths which are calculated by Dijkstra algorithm, and outputs a minimum cost MST. In order to speed the MST calculation, we calculate the shortest paths which end with the edge switches in advance, called GSP (Global Shortest Paths).

Routing Adjustment. When the link monitor detects link overloads, i.e., the link weight is larger than a preset threshold, the routing adjustment will be triggered. In routing adjustment, the GSP will be recalculated using Dijkstra algorithm, and the relevant MSTs will be recalculated using MPH algorithm.

We compare the total cost of the new calculated MST with the current one to decide whether to install the new MST. Suppose we have updated the group to the new MST, and then the group will generate new load on the new links. Therefore,

Algorithm 1 Multicast Routing Adjustment.

```
//Update the link weight, and find out the overloaded links
for $l$ in $L$ do
    $l.weight = l.rate < B?l.rate/B : l.flow * F/B$;
    if $l.weight > W_{thr}$ then
        $C.append(l)$; //Store the overloaded links in $C$
    end if
end for
for $g$ in $G$ do
    if $g.link$ in $C$ then
        $n = g.newMst()$; // Calculate the new MST
        // Check whether to update
        if $checkMst(n.L, g.L)$ then
            for $l$ in $\{n.L$ - $g.L\}$ do
                $l.weight+ = F/B$;
            end for
            for $l$ in $\{g.L$ - $n.L\}$ do
                $l.weight- = F/B$;
            end for
            $g.update()$; // Update the MST
        end if
    end if
end for
```

we should update the weight of the new links, i.e., an addition of F/B to each. If the reduction of the total cost between the new MST and the current one is no less than a reduction threshold, we think it is worth updating, and the new MST will be installed to switches. Specifically, consider an MST $M(V, L)$, where V and L denote the set of nodes and links, respectively. Each link $l \in L$ is associated with a weight $w(l)$. Let C denotes the cost of an MST, which is the sum of all link weight in the MST, and C_{thr} denotes the reduction threshold. For the current MST $M(V_{cur}, L_{cur})$, the cost $C_{cur} = \sum^{l \in L_{cur}} w(l)$. For the new MST $M(V_{new}, L_{new})$, the cost $C_{new} = \sum^{l \in L_{new} \cap L_{cur}} w(l) + \sum^{l \in L_{new} - L_{cur}} (w(l) + F/B)$. If $C_{cur} - C_{new} \geqslant C_{thr}$, the new MST will be installed. By default, we let the $C_{thr} = F/B$.

To reduce the overhead of routing adjustment, we calculate the GSP only once during each adjustment cycle. Here we consider the situation where there are multiple groups need to be adjusted. Consider that when the MST of a group has been adjusted, the weight of the associated links will change. Therefore, we need to adjust the weight of the changing links. For example, the links set of a group G_1's MST is $L_{cur} : \{l_1, l_2, l_3\}$ before adjustment, and are changed to $L_{new} : \{l_1, l_3, l_4\}$ after adjustment. Then the traffic which on link l_2 will switch to link l_4, and the weight of l_2 and l_4 will change. We correct the weight of l_2 and l_4 by $l_2 = l_2 - F/B$, and $l_4 = l_4 + F/B$, respectively. As a result, other groups which wish to adjust their MST to l_2 or l_4 could take the effect of group G_1 into account. The pseudocode of Routing Adjustment is shown in Algorithm 1 and 2.

Algorithm 2 checkMST(L_{new}, L_{cur})

```
// Calculate the $L_{cur}$
for $l$ in $L_{cur}$ do
    $C_{cur}+ = l.weight$;
end for
// Calculate the $L_{new}$
for $l$ in $\{L_{cur} \cap L_{new}\}$ do
    $C_{new}+ = l.weight$;
end for
for $l$ in $\{L_{new} - L_{cur}\}$ do
    $C_{new}+ = l.weight + F/B$;
end for
if $C_{cur} - C_{new} \geqslant C_{thr}$ then
    return $TRUE$;
end if
return $FALSE$;
```

IV. Implementation Details

We implement MCTCP on a Linux platform, the HSP as a kernel-level module, and the MGM as an application on Ryu [16], a popular open source SDN controller. In order to achieve straightforward deployment, the HSP adopts the same semantics as TCP, and provides common socket APIs. Therefore, the application programmers can use MCTCP as easy as TCP in programming. To verify the applicability of MCTCP, we apply MCTCP on HDFS to optimize the data replication mechanism.

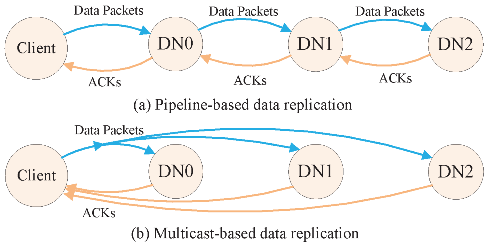
A. Prototype Implementation

We add a new transport-layer protocol by assigning a new protocol number (e.g. 106) to MCTCP when implementing the HSP prototype, in order to avoid modification in the kernel source code. Therefore, the HSP works as a kernel module which can be loaded and unloaded as needed. We implement the MGM module on Ryu. The MGM only processes the MCTCP traffic (e.g. with protocol number 106), and a general routing module processes the non-MCTCP traffic. For the MCTCP traffic, we use the load-based algorithm for routing calculation, always seeking for minimum cost, since we can adjust the MSTs if the load changes. For the non-MCTCP traffic, we use the distance-based algorithm, and the routing will not change unless link fails.

B. Application Integration

HDFS is one of the most widely deployed distributed file system in data centers, which acts as the default file system in Hadoop. Its data replication process is a typical one-to-many data transmission, during which the client gets the list of DNs (Data Nodes) from an NN (Name Node), and then delivers the data chunks to them. By default, the replication factor in HDFS is three, so we assume the replication factor is three.

As shown in Fig. 4(a), the original HDFS employs a pipeline-based replication method. The data transmission unit is a packet (usually 64KB). For each packet, the client first transfers it to DN0; then the DN0 stores and passes it to DN1;



(a) Pipeline-based data replication

(b) Multicast-based data replication

Fig. 4: Illustration of Pipeline-based and Multicast-based data replication.

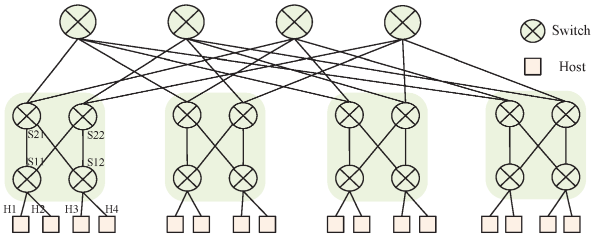


Fig. 5: Network topology used in evaluation.

finally the DN1 stores and transfers it to DN2. After the DN2 receives the packet, it returns an acknowledgment to DN1; then the DN1 returns an acknowledgment to DN0; finally the DN0 returns an acknowledgment to the Client. Therefore, the whole process can be regarded as a six-stage pipeline. We denote the original HDFS as HDFS-O. HDFS-O has $2 \cdot n$ stages when configured as n replicas, resulting in long delay in packet transmission. In addition, HDFS-O delivers data in unicast, which will generate a large number of duplicated packets into the network and reduce the overall transmission performance.

We implement multicast-based data replication on HDFS using MCTCP, which is denoted as HDFS-M. As shown in Fig. 4(b), the client divides the data into packets, and then delivers them to three data nodes DN0, DN1, DN2 in multicast. For each packet, the client transfers it to DN0, DN1, DN2 simultaneously using MCTCP, and then all the data nodes return acknowledgements to the client directly. Therefore, HDFS-M's data replication procedure can be regarded as a two-stage pipeline. Compared with HDFS-O, HDFS-M has shorter stages (two stages to six stages), so that results in lower latency. Meanwhile, since HDFS-M delivers data in multicast, the redundant packets in network are reduced greatly.

In the similar way, we implement another multicast-based HDFS using TCP-SMO, which is the state-of-the-art transport-layer reliable multicast scheme, called HDFS-T. Since TCP-SMO is a receiver-initiated solution, we have to use additional mechanism to inform a DN to subscribe a specific multicast group before writing data to the DN.

V. Evaluation

We build a test platform on Mininet [17]. The hardware consists of one server running Ubuntu 12.04.5 LTS operating system, with Intel (R) Xeon (R) E5-2620 @ 2.00GHz CPU, 32GB RAM. We install Mininet 2.2.0 and Openvswitch 2.1.0, RYU 3.17 on it. The network consists of 16 hosts interconnected using a fat-tree of twenty 4-port switches, as shown in Fig. 5.

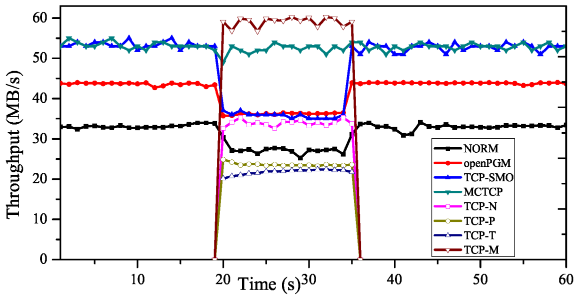


Fig. 6: Throughput comparison among NORM, openPGM, TCP-SMO and MCTCP. From 20s to 35s, a TCP flow is injected using iperf.

We perform three evaluations, including the basic evaluation, the real-word workload evaluation and the application-based evaluation, and we also discuss the complexity of the Controller.

Basic Evaluation: We compare the performance of NORM [18], openPGM [19], TCP-SMO [13] and MCTCP, and verify the congestion-awareness, robustness and TCP-friendliness of MCTCP. NORM and openPGM (an implementation of PGM) are two popular open-source application-layer reliable multicast schemes, and TCP-SMO is a transport-layer reliable multicast scheme. Here we only make a comparison among the comprehensive reliable multicast schemes (all the previous SDN-based schemes are focused on routing strategies).

Real-world Workload Evaluation: We evaluate the performance of MCTCP and TCP-SMO under background traffic in the patterns of two realistic workloads, the web search workload [11] and the data mining workload [12] from production data centers.

Application-based Evaluation: We evaluate the performance of HDFS-M, HDFS-T and HDFS-O under the background traffic in the patterns of web search workload.

Complexity of the Controller: We discuss the capability of the SDN controller, such as the running time of the algorithm, computation and network overhead of the controller.

A. Basic Evaluation

The purpose of the basic evaluation is to evaluate the performance of MCTCP, and to see the behavior in case of link congestion and link failures, how MCTCP performs when co-existing with standard TCP.

First, we evaluate the throughput of NORM, openPGM, TCP-SMO and MCTCP, with the congestion control enabled, transmission rate at 500Mbps. In this test, we use the first four nodes in Fig. 5, the node $H1$ sends data to the rest three nodes. At time 20s, we start a TCP flow using iperf and make it conflict with the MST of the multicast group to simulate congestion. The iperf lasts 15s. Fig. 6 depicts the throughput of the four schemes. The TCP-N, TCP-P, TCP-T and TCP-M indicate the injected TCP flow in NORM, openPGM, TCP-SMO and MCTCP test, respectively. We have the following observations:

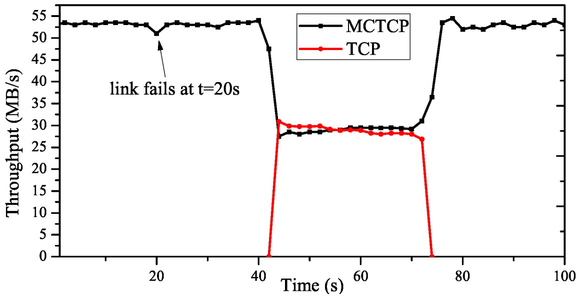


Fig. 7: Throughput of MCTCP when a link failure occurs and co-existing with TCP. The link $S11 \to S21$ fails at t=20s. A TCP flow is started from time 44s to 74s.

When no link congestion occurs (during time 0-20s and 35-60s), MCTCP achieves 60% and 22% better performance than NORM and openPGM, and is analogous to TCP-SMO. When link congestion occurs (during time 20-35s), MCTCP exhibits nearly no throughput degradation, while NORM, openPGM and TCP-SMO suffer from throughput degradation by 17%, 17% and 33%, respectively. That means MCTCP achieves up to 90%, 44% and 45% better performance than NORM, openPGM and TCP-SMO, respectively when congestion happens.

MCTCP outperforms the alternative schemes mainly because of two reasons. First, MCTCP is a transport-layer protocol, which can process data transmission, packet acknowledgement and data re-transmission more efficiently than the application-layer protocol. Therefore, MCTCP outperforms NORM and openPGM even in no link congestion scenes. Second, MCTCP can detect link congestion in real-time, and adjust the MST to bypass congested links immediately once the link congestion is detected. Therefore, when link congestion occurs, MCTCP updates the MST to minimize performance loss. In the alternative schemes, however, once established, the MSTs are scarcely changed, leading to significant performance degradation when congestion occurs.

Second, we evaluate the results of MCTCP when dealing with link failures and sharing links with TCP. The three alternative schemes are based on IGMP, so they leverage the mechanisms of IGMP to deal with link failures. For IGMP, a querier is responsible for sending out IGMP group membership queries on a timed interval to retrieve IGMP membership reports from active members, and to allow updating of the group membership tables. Hence, the MSTs will not be updated during the query interval even if a link failure occurs. The link failure recovery time of NORM, openPGM and TCP-SMO depends on the query interval, which is typically 10s-60s. Therefore, we do not evaluate the results of the three alternative schemes in dealing with link failures.

Like the previous experiment, we observe that the MST is $\{S11 \to S21, S21 \to S12\}$ after the transmission begins. At time 20s, we shutdown the link of $S11 \to S21$, and then start a TCP flow between $H2$ and $H3$ using iperf during time 44s-74s. TCP and MCTCP both run "reno" congestion control algorithm. Fig. 7 depicts the results in this experiment. We have the following observations from this figure. First, the link failure has only a slight impact on MCTCP throughput. This is because the MST will be updated to bypass the failing link when a link failure is detected, and the lost multicast packets will be retransmitted quickly. Second, during the period from time 44s to 74s, as no alternative links for adjustment, MCTCP has to share the link with TCP. The congestion control mechanism on the sender makes MCTCP TCP-friendly.

B. *Real-world Workload Evaluation*

In this experiment, we try to find out how MCTCP performs in practice network environment. We assume MCTCP shares the network with the background traffic, which are according to the patterns of two real-world workloads, the web search and the data mining. We generate the background traffic using a client/server model. All the hosts open socket sinks for incoming traffic and all the hosts send data to a server based on exponential distribution. Each client randomly selects a receiver each time. The flow sizes are in according with the CDFs of realistic workloads mentioned above, which are similar to [20].

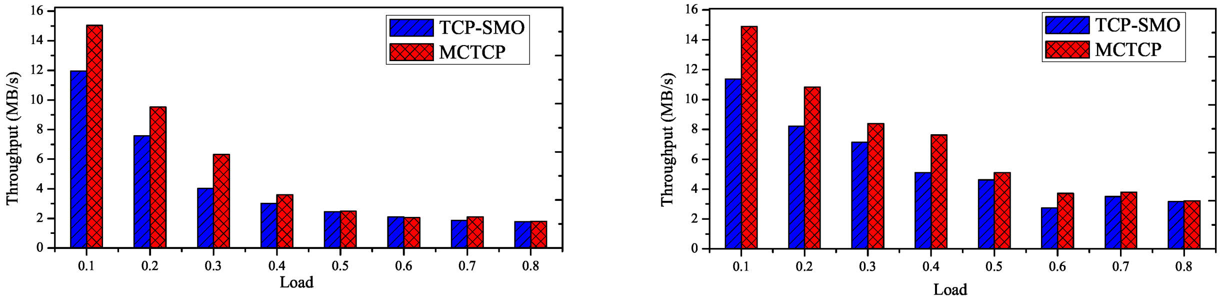
We evaluate the performance of MCTCP and TCP-SMO under the background traffic in the two workload patterns, while varying the network loads from 0.1 to 0.8. We start four multicast groups, each with a sender and three receivers. For each group, the members at least span 3 racks, and the racks and group members are randomly chosen. Different multicast groups do not share common group members. We run ECMP during this evaluation. The MGM monitors the network at 2 seconds polling rate.

Fig. 8 shows the bandwidth of MCTCP and TCP-SMO under background traffic of web search and data mining workloads respectively. We make the following two observations. First, when the load is less than 0.5, under both workloads, MCTCP outperforms TCP-SMO. Specifically, MCTCP achieves bandwidth improvements by ∼20%-57% under the web search workload and ∼17%-49% under the data mining workload. Second, when the load is no less than 0.5, MCTCP is comparable with TCP-SMO and has a narrow win in most cases under both workloads.

MCTCP is able to find the less congested links during runtime, and adjusts the MSTs to improve the performance of multicast groups. When the load is less than 0.5, the traffic does not saturate all links. Due to the unload-balanced nature of the two workloads, MCTCP can always find the lower cost MST to keep the multicast group in the less congested state. Therefore, MCTCP can outperform TCP-SMO. When the load is no less than 0.5, all the links are almost saturated by the background traffic, resulting in no obvious space for MST adjustment. Given the average traffic load in DCNs is moderate, which generally at 30% [7], MCTCP can perform well in realistic data centers environment.

C. *Application-based Evaluation*

We carry out performance comparison among HDFS-O, HDFS-T and HDFS-M under background traffic in patterns



(a) With background traffic in web search patterns

(b) With background traffic in data mining patterns

Fig. 8: Average multicast throughput of MCTCP and TCP-SMO under two background traffic in patterns of web search and data mining.

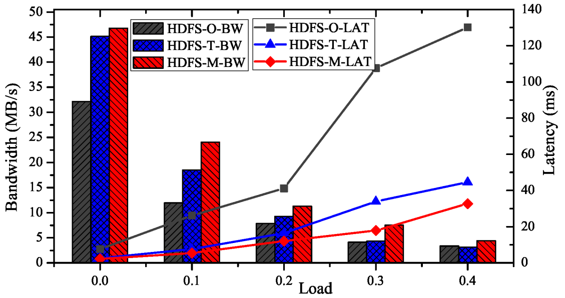


Fig. 9: Performance comparison among HDFS-M, HDFS-T and HDFS-O.

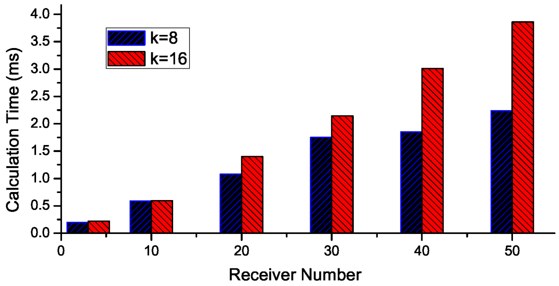


Fig. 10: Calculation times for different receiver numbers.

of the web search workload. In HDFS, for the common case, the replication factor is default three, with one replica on a node in the local rack, another on a node in a remote rack, and the last on a different node in the same remote rack. However, in many cases, three copies could be distributed in different racks for the sake of load balance or data protection. In order to fully demonstrate the impact of the network on HDFS data replication, we configure all replicas on remote nodes. That is, we start a name node and three data nodes on four different hosts, and perform tests on the name node host. All data nodes store data on Ramdisk. We compare HDFS-O, HDFS-T and HDFS-M under different additional network loads, varying from 0 to 0.4. When load is 0, there is no background traffic.

We measure two metrics, the per-packet latency and the overall throughput, with the packet size 64KB, data size 500MB. Fig. 9 depicts the results, from which we have three clear observations. First, the multicast-based data replication schemes outperform the pipeline-based scheme, especially in per-packet latency. The latency of pipeline-based scheme is typically about 3 times the multicast-based schemes. Second, HDFS-M is almost analogous to HDFS-T without background traffic, and improves the bandwidth by ∼20%-50%, and decreases the per-packet latency by ∼13%-47% under web search background traffic. Third, compared with HDFS-O, HDFS-M achieves ∼30%-101% better bandwidth and ∼70%-83% lower per-packet latency.

HDFS-M outperforms HDFS-O and HDFS-T mainly due to two reasons. First, compared with HDFS-O, HDFS-M has shorter transmission paths and generates much fewer redundant network packets. So the latency can be significantly reduced and the probability of being affected by the background traffic is smaller. In HDFS-O, each packet is processed serially by all the three data nodes. For each packet, the completion time includes multiple processing and transmission time. Moreover, when the network is congested, multiple redundant unicast traffics will incur heavier congestion in network, leading to significant performance degradation. Second, HDFS-M can adjust the MSTs to the most efficient one timely based on the network utilization. Therefore, HDFS-M can choose the least congested links to minimize performance loss. Although HDFS-T has the same short transmission path as HDFS-M, it is unable to bypass the congested links, thus results in more performance degradation due to background traffic.

D. *Complexity of the Controller*

The MST can be calculated immediately when a group arrives, as the GSP which needed in MST calculation will always be calculated in advance, i.e., when the controller initiation or at the beginning of each adjustment period. Therefore, the calculation time of a group is independent of the topology scale and the total number of active groups, but only related to the receiver numbers. Fig. 10 depicts the results of MST calculation time for various receivers in $k = 8$ and $k = 16$ Fat-tree topology. There are 32 edge switches in $k = 8$ Fat-tree topology. When the receiver number is larger than 32, the MST will span all pods, thus leading to saturation on calculation time. It is worth noting that when generating

TABLE I: Calculation times in different topologies.

Topology	Link Stats(ms)	GSP(ms)	Group Generation	
			Calculation(ms)	Install(ms)
k=8	38.41	1.789	0.615	13.929
k=16	154.254	25.544	0.722	13.53

TABLE II: Adjustment times for various groups g in different topologies.

Topology	g=10(ms)	g=100(ms)	g=1000(ms)	g=10000(ms)
k=8	12.458	75.695	491.822	5189.126
k=16	62.951	111.259	635.594	5453.479

TABLE III: CPU usage and network overhead in SDN controller for different groups g.

Topology	CPU		Netowork	
	g=100(%)	g=1000(%)	g=100(KB/s)	g=1000(KB/s)
k=8	1.20	4.70	54.8	55.2
k=16	3.10	5.80	292	293

TABLE IV: Comparison of reliable multicast approaches. 'CA' represents Congestion-Awareness.

Approaches	Initial Model	Layer	CA	Robustness
PGM [21]	Receiver	Application	No	Low
NORM [18]	Receiver	Application	No	Low
TCP-SMO [13]	Receiver	Transport	No	Low
RDCM [14]	Receiver	Application	No	High
SCE [28]	Sender	Transport	No	Low
TCP-XM [29]	Sender	Application	No	Low
MCTCP	Sender	Transport	Yes	High

a group, we first calculate the MST, and then install it to the corresponding switches. The install time which depends on the controller platform implementation is out of the scope of this paper.

During an adjustment period, the controller first checks the current status of all links, in which we use '$OFPMP\_PORT\_STATS$' interface to get the port statistics of each switch. If link congestion is detected, we will re-calculate the GSP using Dijkstra algorithm. Table I shows the link stats, GSP calculation and a group generation time under $k = 8$ and $k = 16$ Fat-tree topology, respectively. We believe the link stats time is limited by the single-thread implementation of Ryu, and it can accelerate the process to more acceptable range by using multi-thread. By default, we will calculate the full GSP in each adjustment period, so that the controller can adjust a large number of groups in a single period. In the case of only a small number of active groups, we choose to not calculate the full GSP, but only the shortest paths for MST calculation on demand .

As can be seen from the Algorithm 1, the time complexity of multicast adjustment is $O(|L| + |G|)$, where the $|L|$ is number of links and the $|G|$ is the number of groups which need to be adjusted. Table II shows the adjustment time of various group numbers under $k = 8$ and $k = 16$ Fat-tree topology. The results turn out that the controller can adjust thousands of groups within one second.

We examine the CPU usage and network overhead of the SDN controller for different groups in $k = 8$ and $k = 16$ Fat-tree topology, respectively. As shown in Table III, the computation and network overhead are negligible.

VI. Related Work

Reliable Multicast: As a traditional network technology, reliable multicast has been studied for decades, during which a large number of reliable multicast solutions have been proposed. These proposals can be divided into two categories, the receiver-initiated and the sender-initiated.

Most of the previous reliable multicast solutions are receiver-initiated. In this mode, each receiver needs to obtain the correct multicast address in advance, and subscribes or unsubscribes it freely, like typical researches PGM [21] [19] [22], ARM [23], NORM [18], TCP-SMO [13], SRM [24], RMTP [25], TMTP [26], RDCM [14] etc. The receiver-initiated mode is more suitable for large group scenes. The sender does not maintain information for any receivers, and thus bandwidth of the reliable multicast will not reduce badly as the number of receivers increases. On the contrary, the sender-initiated mode is mainly designed for medium or small group scenes, typical including M/TCP [27], SCE [28], TCP-XM [29] and so on.

Most of the existing reliable multicast solutions are application-layer protocols, like PGM and NORM, which suffer from high software overhead on end hosts in the high bandwidth, low latency network environment of data centers. RDCM [14] focuses on reliable multicast in data centers, leveraging the rich path diversity available in data center networks to build backup overlays, and recovers lost packets in a peer-to-peer way among receivers. TCP-SMO and SCE are transport-layer protocols, which have high performance on end hosts like MCTCP. But they are based on traditional multicast management protocols and routing algorithms like IGMP and PIM-SM, which are not aware of link failures and congestion, leading to low routing efficiency and robustness. M/TCP is network-equipment protocol, which requires assistance from network devices. Blast [6] focuses on accelerating high-performance data analytics applications by optical multicast. Kim [30] focuses on scheduling multicast traffic with deadlines. As far as we are aware, all the existing reliable multicast schemes are not congestion-aware. Table IV summarizes reliable multicast approaches similar to MCTCP.

SDN-based Multicast: SDN technology provides a logically centralized approach to achieve IP multicast. Avalanche [31] and OFM [32] propose SDN-based multicast system, using the SDN-controller for multicast routing and management to improve efficiency and security. Similarly, CastFlow [33] calculates all possible routes from sources to group members in advance to speed up the processing of events in multicast groups. Ge [34] proposes an OpenFlow-based dynamic MST algorithm to optimize the performance of multicast transmissions, enabling adjustable multicast routing when source and group members are unchanged. Shen [35] proposes an approximate algorithm, called Recover Aware Edge Reduction Algorithm (RAERA) to achieve a new reliable multicast tree for SDN, named Recover-aware Steiner Tree (RST).

However, all of the SDN-based multicast researches are focused on multicast routing only, but not concerning about the multicast protocol design. Moreover, they are not congestion-aware.

VII. Conclusion

In order to meet the requirements of data center multicast, we propose MCTCP, an SDN-based reliable multicast data transmission solution, mainly for small groups in data centers. It is a sender-initiated transport-layer solution which extends TCP as the host-side portocol. The MSTs are maintained by the MGM in a centralized way. Therefore, the MGM can leverage real-time network states to reactively multicast flows to active and low-utilized links. For each group, the MST is calculated during session establishment and adjusted dynamically in case of link congestion or failures to achieve optimal routing efficiency and robustness. Taken together, MCTCP is efficient in both end hosts and multicast routing. In our experiments, MCTCP outperforms the existing reliable multicast solutions, especially in the case of co-existing with background traffic. Moreover, we implement multicast-based data replication on HDFS using MCTCP. Experimental results show that the multicast-based data replication has better performance than the original pipeline-based HDFS, and the multicast-based scheme built with MCTCP performs better.

Acknowledgment

This work is supported in part by the National High Technology Research and Development Program (863 Program) of China under Grant No.2013AA013203; National Basic Research 973 Program of China under Grant 2011CB302301; Key Laboratory of Information Storage System, Ministry of Education, China. This work is also supported by NSFC 61173043 and State Key Laboratory of Computer Architecture, No.CARCH201505.

References

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *MSST*, May 2010, pp. 1–10.

[2] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," ser. OSDI. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[3] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, oct 2003.

[4] "Hadoop," https://hadoop.apache.org.

[5] "Spark," http://spark.apache.org.

[6] Y. Xia, T. E. Ng, and X. S. Sun, "Blast: Accelerating high-performance data analytics applications by optical multicast," in *INFOCOM*, Hong Kong, China, 2015.

[7] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *IMC*. New York, USA: ACM, 2010, pp. 267–280.

[8] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol independent multicast-sparse mode (pim-sm): Protocol specification," June 1997, rFC2117.

[9] B. Cain, D. S. E. Deering, B. Fenner, I. Kouvelas, and A. Thyagarajan, "Internet Group Management Protocol, Version 3," IETF RFC 3376, Oct. 2015.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM*, vol. 38, no. 2, pp. 69–74, 2008.

[11] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *SIGCOMM*. New York, USA: ACM, 2010, pp. 63–74.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *SIGCOMM*. New York, USA: ACM, 2009, pp. 51–62.

[13] S. Liang and D. Cheriton, "Tcp-smo: extending tcp to support medium-scale multicast applications," in *INFOCOM*, 2002, pp. 1356–1365.

[14] D. Li, M. Xu, M. chen Zhao, C. Guo, Y. Zhang, and M.-Y. Wu, "Rdcm: Reliable data center multicast," in *INFOCOM*, April 2011, pp. 56–60.

[15] H. Takahashi and A. Matsuyama, "An approximate solution for the steiner problem in graphs," *Math.Japonica*, vol. 24, no. 6, pp. 573–577, 1980.

[16] "Ryu," http://osrg.github.io/ryu.

[17] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *CoNEXT*. New York, NY, USA: ACM, 2012, pp. 253–264.

[18] B. Adamson, C. Bormann, M. Handley, and J. Macker, "Nack-oriented reliable multicast (norm) transport protocol," November 2009, rfc5740.

[19] "openpgm," http://code.google.com/p/openpgm.

[20] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, and C. Tian, "Information-agnostic flow scheduling for commodity data centers," in *NSDI*. Oakland, CA: USENIX Association, May 2015, pp. 455–468.

[21] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, and S. Lin, "Pgm reliable transport protocol specification," December 2001, rFC3208.

[22] L. Rizzo, "Pgmcc: A tcp-friendly single-rate multicast congestion control scheme," ser. SIGCOMM. New York, NY, USA: ACM, 2000, pp. 17–28.

[23] L. Lehman, S. Garland, and D. Tennenhouse, "Active reliable multicast," in *INFOCOM*, vol. 2, Mar 1998, pp. 581–589.

[24] 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.

[25] S. Paul, K. K. Sabnani, J. C. H. Lin, and S. Bhattacharyya, "Reliable multicast transport protocol (rmtp)," *Selected Areas in Communications, IEEE Journal on*, vol. 15, no. 3, pp. 407–421, 1997.

[26] R. Yavatkar, J. Griffoen, and M. Sudan, "A reliable dissemination protocol for interactive collaborative applications," in *MULTIMEDIA*. New York, USA: ACM, 1995, pp. 333–344.

[27] V. Visoottiviseth, T. Mogami, N. Demizu, Y. Kadobayashi, and S. Yamaguchi, "M/tcp: The multicast-extension to transmission control protocol," in *ICACT*, Muju, Korea, Feb. 2001.

[28] R. Talpade and M. Ammar, "Single connection emulation (sce): an architecture for providing a reliable multicast transport service," in *Distributed Computing Systems*, May 1995, pp. 144–151.

[29] K. Jeacle and J. Crowcroft, "Tcp-xm: unicast-enabled reliable multicast," in *ICCCN*, Oct 2005, pp. 145–150.

[30] K. S. Kim, C. ping Li, and E. Modiano, "Scheduling multicast traffic with deadlines in wireless networks," in *INFOCOM*, April 2014, pp. 2193–2201.

[31] A. Iyer, P. Kumar, and V. Mann, "Avalanche: Data center multicast using software defined networking," in *COMSNETS*, Jan 2014, pp. 1–8.

[32] Y. Yang, Z. Qin, X. Li, and S. Chen, "Ofm: A novel multicast mechanism based on openflow," *Advances in Information Sciences and Service Sciences*, vol. 4, no. 9, pp. 278–286, 2012.

[33] C. A. C. Marcondes, T. Santos, A. P. Godoy, C. C. Viel, and C. A. C. Teixeira, "Castflow: Clean-slate multicast approach using in-advance path processing in programmable networks." in *ISCC*. IEEE, 2012, pp. 94–101.

[34] J. Ge, H. Shen, E. Yuepeng, Y. Wu, and J. You, "An openflow-based dynamic path adjustment algorithm for multicast spanning trees," in *Security and Privacy in Computing and Communications (TrustCom)*, July 2013, pp. 1478–1483.

[35] S.-H. Shen, L.-H. Huang, D.-N. Yang, and W.-T. Chen, "Reliable multicast routing for software-defined networks," in *INFOCOM*, Hong Kong, China, 2015.