Journal of Network and Computer Applications 95 (2017) 105–117

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

A congestion-aware and robust multicast protocol in SDN-based data center networks

Tingwei Zhu[a], Dan Feng[a,*], Fang Wang[a], Yu Hua[a], Qingyu Shi[a], Yanwen Xie[a], Yong Wan[b]

[a] Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System (School of Computer Science and Technology, Huazhong University of Science and Technology), Ministry of Education of China, China
[b] Computer Engineering College, Jingchu University of Technology, China

ARTICLE INFO

Keywords:
Reliable multicast protocol
Software-defined networking
Data center networks
Distributed file systems
Data replication

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 (Hadoop Distributed File System) compared with the original and TCP-SMO (an alternative reliable multicast scheme) based ones, e.g., achieves up to 1.5× and 1.1× improvements in terms of throughput, respectively.

1. 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 Shvachko et al. (2010) in Hadoop, Ceph Weil et al. (2006) in Red Hat and GFS Ghemawat et al. (2003) 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 Distributed Cache in Hadoop Map Reduce (Hadoop, 2016) and Broadcast variables in Apache Spark (2016). 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 Xia et al. (2015).
- **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,

* Corresponding author.
E-mail addresses: twzh@hust.edu.cn (T. Zhu), dfeng@hust.edu.cn (D. Feng), wangfang@hust.edu.cn (F. Wang), csyhua@hust.edu.cn (Y. Hua), qingyushi@hust.edu.cn (Q. Shi), ywxie@hust.edu.cn (Y. Xie), wanabc@hust.edu.cn (Y. Wan).

http://dx.doi.org/10.1016/j.jnca.2017.07.013
Received 22 January 2017; Received in revised form 19 June 2017; Accepted 26 July 2017
Available online 27 July 2017

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 Benson et al. (2010) 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. In this paper, we define the "sender-initiated" to be "the sender specifies the receivers in a multicast session, and the receivers do not need to know the multicast group address and join the group in advance". On the contrary, the "receiver-initiated" is "the receivers need to obtain the multicast group address and join the group in advance, and the sender does not need to know about the receivers' information". Second, traditional IP multicast routing algorithms, such as PIM-SM Estrin et al. (1997), 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 Benson et al. (2010). Third, traditional multicast group management protocols, such as Internet Group Management Protocol (IGMP) Cain et al. (2015), 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) McKeown et al. (2008), 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 Alizadeh et al. (2010) and the data mining Greenberg et al. (2009), 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 Liang and Cheriton (2002), called HDFS-M and HDFS-T, respectively to improve performance of data replication. Compared with HDFS-O (the original pipeline-based HDFS version) and HDFS-T, HDFS-M decreases the per-packet latency by ~50% – 72% and ~10% – 45%, and improves the throughput by ~50% – 1.6× and ~20% – 1.3×, respectively under web search background traffic.

Some preliminary results of this paper were published in the Proceedings of the IEEE/ACM International Symposium on Quality of Service (IWQoS, 2016) Zhu et al. (2016). In this paper, we improve the MST calculation and adjustment algorithm, i.e., Shortest Widest Path (SWP) based algorithm, and extend the evaluations in a new data center network topology, i.e., Leaf-Spine topology.

The rest of paper is organized as follows. Section 2 presents the motivation and related work of this paper. Section 3 presents the design of MCTCP. Section 4 describes the implementation details. Section 5 describes our experimental evaluation of MCTCP and the performance comparisons among the state-of-the-art approaches. Finally, we draw conclusion in Section 6.

2. Motivations and related work

2.1. 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 Benson et al. (2010). Since existing reliable multicast schemes are based on distributed routing algorithms, which cannot make full use of network resources to achieve 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 Greenberg et al. (2009); Li et al. (2011). Traditional multicast management protocols are not aware of link failures, which generally consume 10–60 s (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.

2.2. 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 Speakman et al. (2001); openpgm (2016); Rizzo (2000), ARM Lehman et al. (1998), NORM Adamson et al. (2009), TCP-SMO Liang and Cheriton (2002), SRM Floyd et al. (1997), RMTP Paul et al. (1997), TMTP Yavatkar et al. (1995), RDCM Li et al. (2011) etc. The receiver-initiated mode is more suitable for large group scenes. The sender does not maintain information for any receivers, and thus throughput 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 Visoottiviseth et al. (2001), SCE Talpade and Ammar (1995), TCP-XM Jeacle and Crowcroft (2005) 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 Li et al. (2011) 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 Xia et al. (2015) focuses on accelerating high-performance data analytics applications by optical multicast. Kim et al. (2014) focus on scheduling multicast traffic with deadlines. As far as we are aware, all the existing reliable multicast schemes are not congestion-aware. Table 1 summarizes reliable multicast approaches similar to MCTCP.

SDN-based multicast: SDN technology provides a logically centralized approach to achieve IP multicast. Avalanche Iyer et al. (2014) and OFM Yang et al. (2012) propose SDN-based multicast system, using the SDN-controller for multicast routing and management to improve efficiency and security. Similarly, CastFlow Marcondes et al. (2012) calculates all possible routes from sources to group members in advance to speed up the processing of events in multicast groups. Ge et al. (2013) propose 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 et al. (2015) propose 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.

Table 1
Comparison of reliable multicast approaches. 'CA' represents Congestion-Awareness.

Approaches	Initial Model	Layer	CA	Robustness
PGM Speakman et al. (2001)	Receiver	Application	No	Low
NORM Adamson et al. (2009)	Receiver	Application	No	Low
TCP-SMO Liang and Cheriton (2002)	Receiver	Transport	No	Low
RDCM Li et al. (2011)	Receiver	Application	No	High
SCE Talpade and Ammar (1995)	Sender	Transport	No	Low
TCP-XM Jeacle and Crowcroft (2005)	Sender	Application	No	Low
MCTCP	Sender	Transport	Yes	High

3. 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.

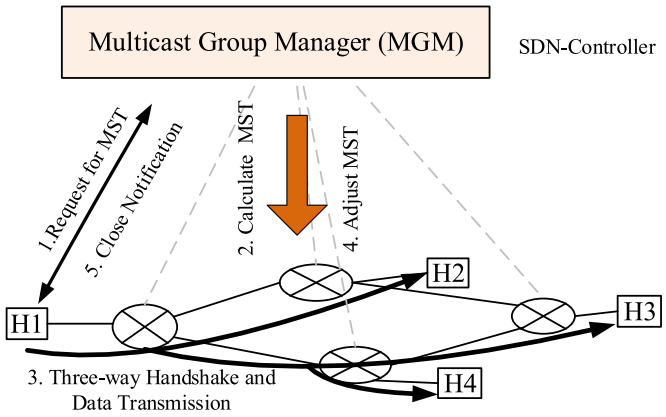


Fig. 1. Illustration of MCTCP.

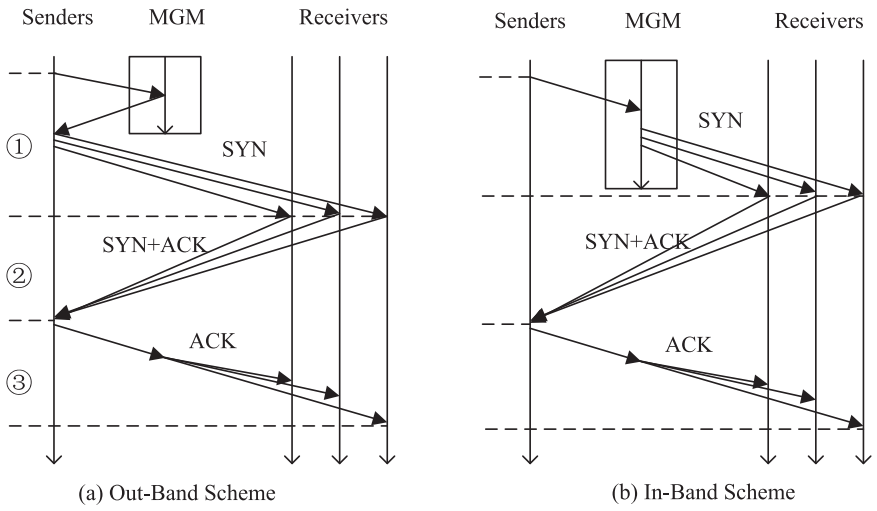


Fig. 2. The Procedure of MCTCP Session Establishment with three receivers.

3.1. Host-side protocol

3.1.1. 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.

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.

3.1.2. 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 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.

3.1.3. Session close

After data transmission is completed, the sender closes the multicast session initiatively, and then notifies the MGM.

3.2. 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.

3.2.1. 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.

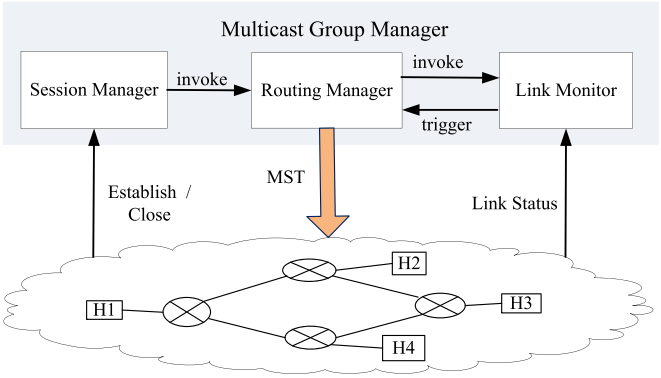


Fig. 3. The Multicast Group Manager.

3.2.2. 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 the 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. We observe 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.

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

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. By default, the $F = 0.4{\cdot}B$.

3.2.3. 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 updated quickly.

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. The routing calculation consists of two steps: all-pair shortest paths calculation and MST calculation. We first calculate the all-pair shortest paths, which are called GSP (Global Shortest Paths) and then calculate the MSTs using the GSP.

(1) Shortest Path (SP) based algorithm. By default, we use the Shortest Path (SP) algorithms (such as Dijkstra, 1959 and Floyd Warshall Floyd, 1962 algorithm) for GSP calculation. The shortest path means the least cost path whose sum weight is the smallest. After calculating the GSP, we calculate the minimum cost MST using the minimum-cost path heuristic algorithm (MPH) Takahashi and Matsuyama (1980). The MPH algorithm inputs a set of sender/receiver nodes and all-pair shortest paths, and outputs a minimum cost MST.

(2) Shortest Widest Path (SWP) based algorithm. However, in many cases, the applications of MCTCP are bandwidth hungry. The SP-based algorithm for MST calculation is not the best solution, as it does not consider the effect of the bottleneck paths. In MCTCP, the throughput of a multicast group is decided by the slowest receiver. Therefore, any bottleneck link in the MST can reduce the throughput of the group significantly. To avoid or mitigate the impact of the bottleneck links, we use the Shortest Widest Path (SWP) Wang and Crowcroft (1996) based algorithm to calculate the MSTs.

First, we use the SWP algorithm for GSP calculation. Given any two nodes i and j, and two constraints B_a and D_a, the goal of the SWP algorithm is to find a path between i and j whose available bandwidth is no less than B_a and the path length is no more than D_a. The basic idea of SWP algorithm is to eliminate any links with an available bandwidth less than B_a and then find the shortest path of the rest of the network.

To maximize the bandwidth of each multicast group, the optimal solution is to find the widest paths for each MST. However, it is costly to calculate the widest MST for each group. To reduce the overhead of routing calculation, we calculate the GSP only once in each adjustment cycle, during which multiple MSTs of different multicast groups may be calculated using the same GSP. Therefore, we do not try to find the widest paths for each group. Instead, our goal is to avoid the bottleneck links and find the wider paths for each pairs in best-effort way, so that we can find the wider MSTs to improve the throughput of the corresponding multicast groups.

To use SWP algorithm for GSP calculation, the primary challenge is to determine the minimum available bandwidth B_a. Since our goal is to improve the throughput of the multicast groups, the bottleneck links are determined by the current throughput of the existing groups. First, the B_a should be no less than a fixed threshold $B_a = \eta{\cdot}B$ (e.g. $\eta = 0.1$), so that the heavy load links will not be selected. Second, the B_a should be no less than the average transmission bandwidth of all the current active multicast groups, thus ensuring sufficient space to adjust MSTs for throughput enhancement. As a result, the $B_a = \max(\eta{\cdot}B, \overline{B_i})$, where B is the link capacity and B_i is the throughput of group i.

Different with the original SWP algorithm, we try to decrease the possibility of choosing the bottleneck links in the GSP calculation, but not to meet the available bandwidth constraints. Therefore, we do not eliminate the links with an available bandwidth less than B_a. Instead, the weight of the links whose available bandwidth is less then B_a will be multiplied by 10. Then we use the shortest path algorithm to calculate the GSP. With a weight far larger than the others, the bottleneck links will not be chosen preferentially. The MPH algorithm preferentially chooses the minimum distance path in each iteration, which fails to find the widest path. We modify the MPH algorithm to choose the widest path in each iteration. If there are multiple paths with the same width (minimum residual bandwidth of all links on the path), the shortest path will be chosen. The modified MPH algorithm is called Widest MPH (WMPH).

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, and the relevant MSTs will be recalculated using the GSP.

(1) Shortest Path (SP) based algorithm. For the SP based algorithm, we compare the total cost of the newly calculated MST with the current one to decide whether to install the new MST. Suppose we have updated the group G_1 to the new MST. Then G_1 will generate new load on the new links in the new MST. Therefore, we should update the weight of the new links. We add the load generated by G_1 to the new links, i.e., an addition of B_1/B to each link, where B_1 is the throughput of G_1. 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 weights 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 $C_{new} = \sum^{l \in L_{new} \cap L_{cur}} w(l) + \sum^{l \in L_{new} - L_{cur}} (w(l) + B_1/B)$. If $C_{cur} - C_{new} \geq C_{thr}$, the new MST will be installed. By default, we let the $C_{thr} = B_1/B$.

(2) Shortest Widest Path (SWP) based algorithm. For the SWP based algorithm, we do not use the total weight of a MST to determine whether to update the MST. If the available bandwidth of the new MST B_n is larger than the current throughput of G_1 B_1, i.e., $B_n > B_1 \cdot (1 + \alpha)(\alpha > 0)$, the new MST will be installed. This ensures that the new MST has enough free bandwidth for throughput improvement of the group G_1. By default, $\alpha = 0.3$, and we can adjust the α according to the network states.

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. 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 $w(l_2) = w(l_2) - B_1/B$, and $w(l_4) = w(l_4) + B_1/B$, 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.

To improve the overall performance, when multiple groups need to be adjusted, the group with heavier traffic load is adjusted first. Specifically, we first sort the multicast groups by their transmission rates within the adjustment interval, and then adjust the multicast group with lower transmission rate first. The routing adjustment algorithm does not guarantee that the placement is optimal, but it performs relatively well in practice as shown in Section 5. The pseudocode of Routing Adjustment is shown in Algorithm 1 and 2.

Algorithm 1. Multicast routing adjustment.

```
1:  // Update the link weight, and find out the overloaded links
2:  for l in L_all do
3:    l. weight = l. rate < B? l. rate/B: l. flownum·F/B;
4:    if l. weight > W_thr then
5:      C. append(l); // Store the overloaded links in C
6:    end if
7:  end for
8:  Sort the groups according their transmission rates in
    ascending order, get G;
9:  for g in G do
10:   if g. link in C then
11:     n = g. newMst(); // Calculate the new MST
12:     // Check whether to update
13:     if checkMst(n. L, g. L) then
14:       // B_g is the throughput of g
15:       for l in {n. L - g. L} do
16:         l. weight += B_g/B;
17:       end for
18:       for l in {g. L - n. L} do
19:         l. weight − = B_g/B;
20:       end for
21:       g. update(); // Update the MST
22:     end if
23:   end if
24: end for
```

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

```
1:  // Calculate the L_cur
2:  for l in L_cur do
3:    C_cur += l. weight;
4:  end for
5:  // Calculate the L_new
6:  for l in {L_cur ∩ L_new} do
7:    C_new += l. weight;
8:  end for
9:  for l in {L_new − L_cur} do
10:   C_new += l. weight + B_g/B;
11: end for
12: if C_cur − C_new ≥ C_thr then
13:   return TRUE;
14: end if
15: return FALSE;
```

Routing examples. The following example compares the results of two routing algorithms (SP and SWP) in MST calculation. As shown in the Fig. 4, the load of each link is marked on the link, and the capacity of each is 10 *Mbps*. When using the SP algorithm, the path with minimum sum load will be chosen. Therefore, the path $S1 \rightarrow S5 \rightarrow S2$, $S1 \rightarrow S5 \rightarrow S3$, $S1 \rightarrow S5 \rightarrow S4$ are the shortest paths from $S1$ to $S2$, $S3$, $S4$, respectively. The MST for a group $G1$: $H1 \rightarrow \{H2, H3, H4\}$ will be $\{S1 \rightarrow S5, S5 \rightarrow S2, S5 \rightarrow S3, S5 \rightarrow S4\}$ and the available bandwidth for $G1$ is 2*Mbps*. When using the SWP algorithm, the load of the bottleneck links (with load larger than 6) will multiply by 10, and then the path with minimum sum load will be chosen. As a result, the path $S1 \rightarrow S6 \rightarrow S2$, $S1 \rightarrow S6 \rightarrow S3$, $S1 \rightarrow S6 \rightarrow S4$ are the shortest paths from $S1$ to $S2$, $S3$, $S4$, respectively. The MST for a group $G1$ will be $\{S1 \rightarrow S6, S6 \rightarrow S2, S6 \rightarrow S3, S6 \rightarrow S4\}$ and the available bandwidth for $G1$ is 5 *Mbps*.

Another example compares the MPH algorithm with the WMPH algorithm as shown in the Fig. 5. Assume the capacity of each link is 10 *Mbps* and the minimum available bandwidth $B_a = 4$ *Mbps* in SWP algorithm. The SWP algorithm calculates the paths from $S1$ to $S2$, $S3$, $S4$ to be $S1 \rightarrow S2$, $S1 \rightarrow S3$, $S1 \rightarrow S2 \rightarrow S4$, with distance and minimum available bandwidth (3, 7), (5, 5) and (6, 7), respectively. After calculating the GSP using the SWP algorithm, the MPH calculates the minimum cost MST based on the shortest distance. As a result, MPH calculates the MST for $G2$: $H1 \rightarrow \{H2, H3\}$ to be $MST1$: $\{S1 \rightarrow S3, S3 \rightarrow S4\}$. However, WMPH will preferentially select the widest path in each iteration, so that the MST for $G2$ will be $MST2$: $\{S1 \rightarrow S2 \rightarrow S4, S4 \rightarrow S3\}$. Compare to $MST1$, the minimum available bandwidth of $MST2$ is larger, i.e., 7 *Mbps* to 5 *Mbps*.

4. Implementation details

We implement MCTCP on a Linux platform, the HSP as a kernel-level module, and the MGM as an application on Ryu (2016), 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.

4.1. 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.

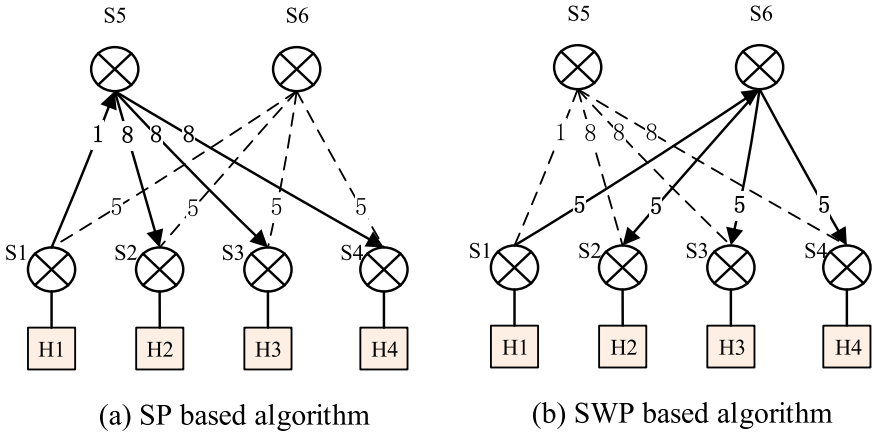


Fig. 4. An example of the comparison between SP-based algorithm and SWP-based algorithm.

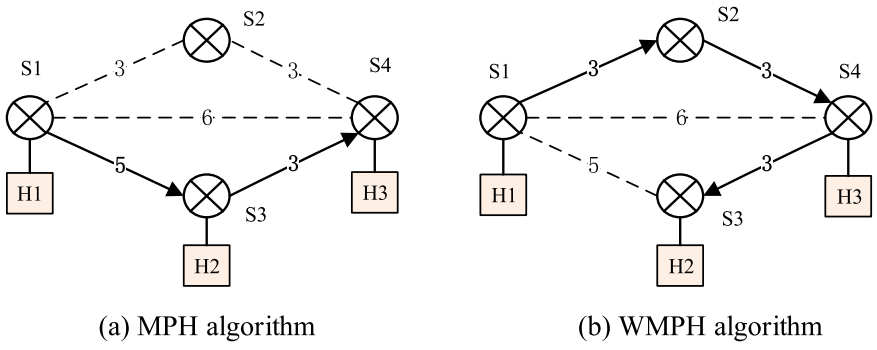


Fig. 5. An example of the comparison between MPH algorithm and WMPH algorithm.

4.2. 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. 6(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; 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.

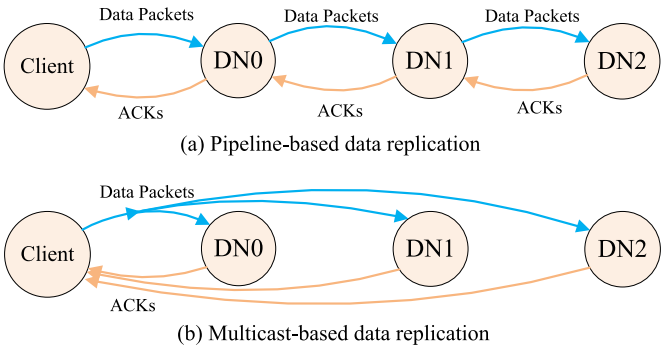


Fig. 6. Illustration of Pipeline-based and Multicast-based data replication.

We implement multicast-based data replication on HDFS using MCTCP, which is denoted as HDFS-M. As shown in Fig. 6(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.

5. Evaluation

We build a test platform on Mininet Handigol et al. (2012). The hardware consists of one server running Ubuntu 12.04.5 LTS operating system, with Intel (R) Xeon (R) E5-2620 @ 2.00 GHz CPU, 32 GB RAM. We install Mininet 2.2.0 and Openvswitch 2.1.0, RYU 3.17 on it. We use two representative data center network topologies (i.e. Fat-Tree Al-Fares et al., 2008 and Leaf-Spine topology Alizadeh and Edsall, 2013) in our evaluation. For the Fat-Tree topology, there are 4 pods (k = 4), containing twenty 4-port switches and 16 hosts, as shown in Fig. 7. For the Leaf-Spine topology, there are 4 spine switches and 8 leaf switches, and each leaf switch contains 4 hosts, as shown in Fig. 8. Two MCTCP versions are implemented in our evaluation, including MCTCP-S and MCTCP-W which are based on SP (Shortest Path) algorithm and SWP (Shortest Widest Path) algorithm respectively.

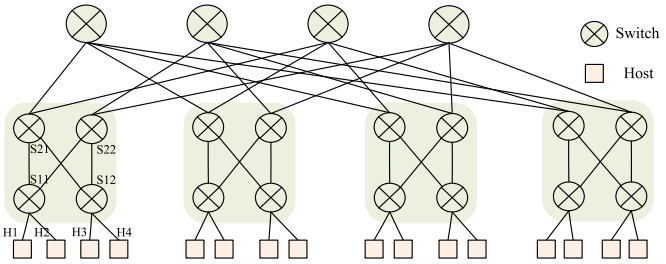


Fig. 7. Fat-Tree topology used in evaluation.

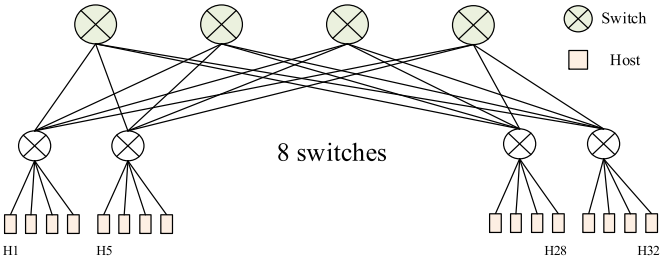


Fig. 8. Leaf-Spine topology used in evaluation.

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 Adamson et al. (2009), openPGM openpgm (2016), TCP-SMO Liang and Cheriton (2002) 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 TCP-SMO, MCTCP-S and MCTCP-W under background traffic in the patterns of two realistic workloads, the web search workload Alizadeh et al. (2010) and the data mining workload Greenberg et al. (2009) from production data centers.

Application-based evaluation: We evaluate the performance of HDFS-O, HDFS-T and HDFS-M 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.

5.1. 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 500 Mbps. In this test, we use the first four nodes in Fig. 7, the node $H1$ sends data to the rest three nodes. At time 20 s, we start a TCP flow using iperf and make it conflict with the MST of the multicast group to simulate congestion. The iperf lasts 15 s Fig. 9 depicts the throughput of the four schemes. We have the following observations:

When no link congestion occurs (during time 0–20 s and 35–60 s), MCTCP achieves 60% and 22% better performance than NORM and openPGM, and is analogous to TCP-SMO. When link congestion occurs (during time 20–35 s), 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

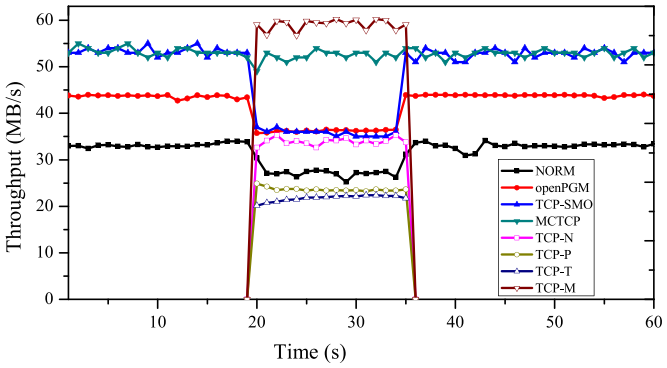


Fig. 9. Throughput comparison among NORM, openPGM, TCP-SMO and MCTCP. From 20–35 s, a TCP flow is injected using iperf. The TCP-N, TCP-P, TCP-T and TCP-M indicate the injected TCP flow in NORM, openPGM, TCP-SMO and MCTCP test, respectively.

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 10–60 s. 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 $\{S11 \rightarrow S21, S21 \rightarrow S12\}$ after the transmission begins. At time 20 s, we shutdown the link of $S11 \rightarrow S21$, and then start a TCP flow between $H2$ and $H3$ using iperf during time 44–74 s. TCP and MCTCP both run "reno" congestion control algorithm. Fig. 10 depicts the results in this experiment. We have the following observations from this figure. First, the link failure has only a slight impact on

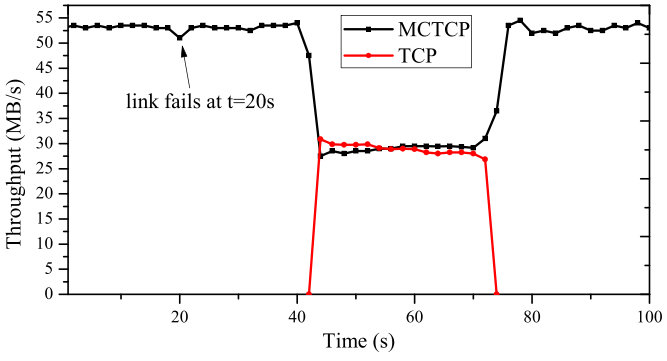
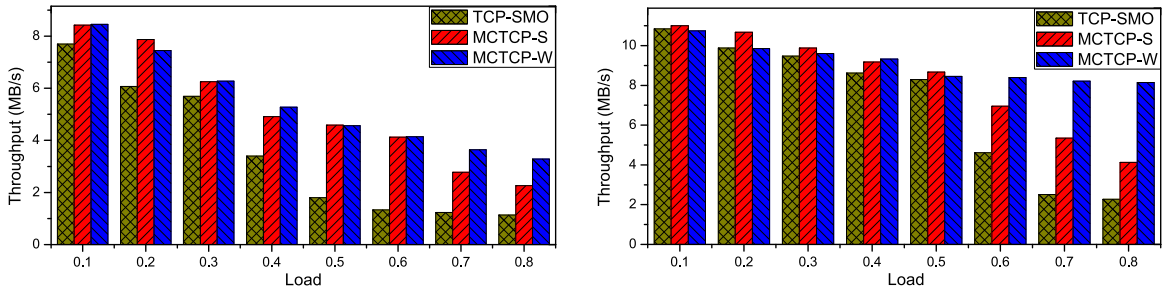


Fig. 10. Throughput of MCTCP when a link failure occurs and co-existing with TCP. The link $S11 \rightarrow S21$ fails at t = 20 s. A TCP flow is started from time 44–74 s.



(a) With background traffic in web search patterns (b) With background traffic in data mining patterns

Fig. 11. Average throughput of TCP-SMO, MCTCP-S and MCTCP-W under two background traffic in patterns of web search and data mining in Fat-Tree topology in separated setup.

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 44–74 s, 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.

5.2. 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.

The evaluations are performed in two setups, including the separated and shared setups.

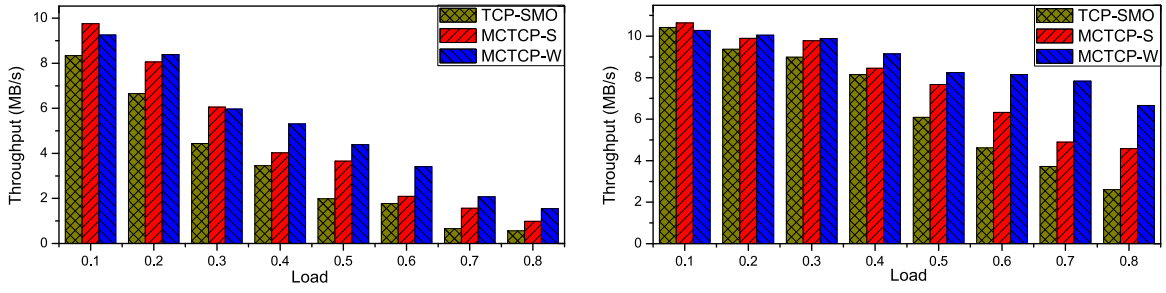
- **Separated setup**. We divide the hosts into two groups, one for generating background traffic (*GroupA*) and the other for performing evaluation (*GroupB*). Specifically, we put all the even numbered hosts into *GroupA* and the odd numbered hosts into *GroupB*. For the *GroupA*, all the hosts open socket sink for incoming traffic and the host with index i sends data to the host with index $(i + 4)mod(numhosts)$. The flow sizes are in according with the CDFs of realistic workloads mentioned above, which are similar to Bai et al. (2015).
- **Shared setup**. The background traffic shares the same hosts with the multicast group members. For the background traffic, 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. To make the background traffic more complicated and ensure that the host side is not the bottleneck of the multicast group transmission, we also inject several additional flows randomly among the nodes which do not run multicast testing.

First, we evaluate the throughput of TCP-SMO, MCTCP-S and MCTCP-W under the background traffic in the two workload patterns in separated setup, while varying the network loads from 0.1 to 0.8, in both Fat-Tree and Leaf-Spine topologies. We start two and four multicast groups in Fat-Tree and Leaf-Spine topology, respectively. Each group consists of a sender and three receivers. The members of each group span at least 3 racks and the racks and group members are randomly chosen from *GroupB*. Different multicast groups do not share common group members. We run ECMP during this evaluation. The MGM monitors the network at 2 s polling rate.

Figs. 11 and 12 show the throughput of TCP-SMO, MCTCP-S and MCTCP-W under background traffic in patterns of web search and data mining workloads in Fat-Tree and Leaf-Spine topology respectively. We make the following two observations. First, MCTCP (both the MCTCP-S and MCTCP-W) outperforms TCP-SMO in throughput. Specifically, MCTCP-S (MCTCP-W) achieves throughput improvements over TCP-SMO by ∼10% – 2.1× (∼10% – 2.1×) and ∼17% – 1.3× (∼10% – 2.1×) under the web search workload in Fat-Tree and Leaf-Spine topology and achieves ∼1% – 1.1× (∼1% – 2.5×) and ∼2% – 76% (∼7% – 1.5×) improvements under the data mining workload in Fat-Tree and Leaf-Spine topology, respectively. Second, MCTCP-W outperforms MCTCP-S in most cases, especially when load is larger than 0.5.

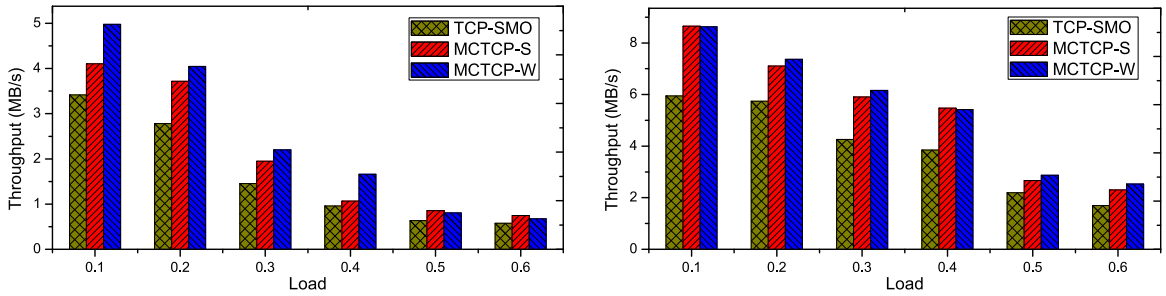
MCTCP is able to find the less congested links during runtime, and adjusts the MSTs to improve the performance of multicast groups. As only half of the hosts in the network generate background traffic, the network will not be saturated by the background traffic. Therefore, there are always more idle links exist during the multicast group transmission. MCTCP-S can find the lower cost MST to keep the multicast group in the less congested state. MCTCP-W will find the wider MST to maximize the multicast group throughput. When the load is low (such as less than 0.5), there are little bottleneck links in the network, the SP based algorithm is comparable with the SWP based algorithm. When the load is heavy, a large number of bottleneck links will significantly reduce the throughput of the multicast groups, even if the sum weight of the MST is low. As a result, MCTCP-W performs much better than MCTCP-S when the load is larger than 0.5.

Second, we perform the same evaluations in shared setup. Figs. 13 and 14 show the results in shared setup. Similar to the separated setup, MCTCP (both the MCTCP-S and MCTCP-W) outperforms TCP-SMO in throughput. MCTCP-S (MCTCP-W) achieves throughput improvements over TCP-SMO by ∼11% – 34% (∼17% – 51%) and ∼10% – 33%



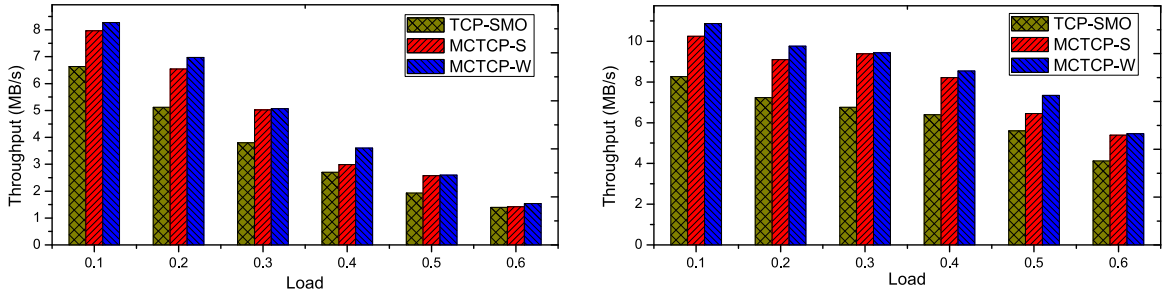
(a) With background traffic in web search patterns (b) With background traffic in data mining patterns

Fig. 12. Average throughput of TCP-SMO, MCTCP-S and MCTCP-W under two background traffic in patterns of web search and data mining in Leaf-Spine topology in separated setup.



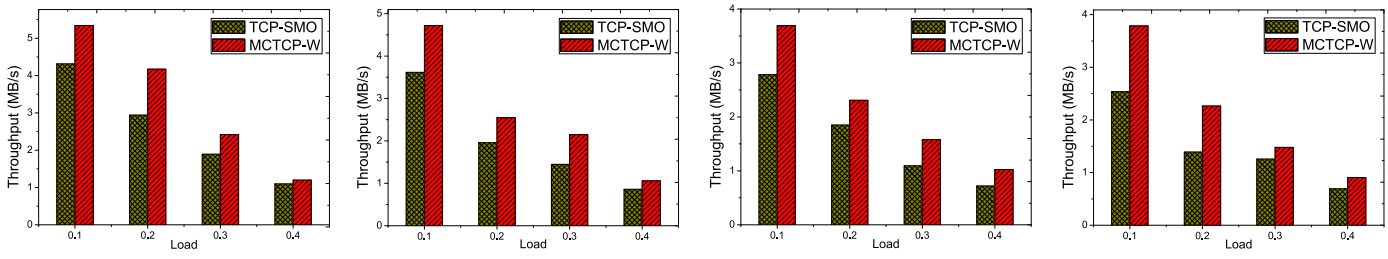
(a) With background traffic in web search patterns (b) With background traffic in data mining patterns

Fig. 13. Average throughput of TCP-SMO, MCTCP-S and MCTCP-W under two background traffic in patterns of web search and data mining in Fat-Tree topology in shared setup.



(a) With background traffic in web search patterns (b) With background traffic in data mining patterns

Fig. 14. Average throughput of TCP-SMO, MCTCP-S and MCTCP-W under two background traffic in patterns of web search and data mining in Leaf-Spine topology in shared setup.



(a) 3 receivers (b) 4 receivers (c) 5 receivers (d) 6 receivers

Fig. 15. Average throughput of TCP-SMO, MCTCP-S and MCTCP-W under background traffic in patterns of web search in different receivers.

(~10% – 36%) under the web search workload in Fat-Tree and Leaf-Spine topology and achieves ~21% – 45% (~28% – 49%) and ~15% – 38% (~30% – 39%) improvements under the data mining workload in Fat-Tree and Leaf-Spine topology, respectively. Compared to the separated setup, the throughput improvements achieved in shared setup is much smaller. This is because that, in shared setup, more background traffic is generated in the network and the traffic is much more balanced compared with that in separated setup, so that leaving much less free space for MST adjustment.

Third, we evaluate the throughput of TCP-SMO and MCTCP-W varying the receiver numbers. The evaluations are performed under the background traffic in web search patterns in shared setup, while varying the network loads from 0.1 to 0.4, in Fat-Tree topology. We start two multicast groups, and the two groups share one same receiver. Fig. 15 shows the results varying the receiver numbers. MCTCP achieves ~20% – 45% throughput improvements in different receiver numbers. From the results we can see that, the throughput improvements remain as the receiver number increases.

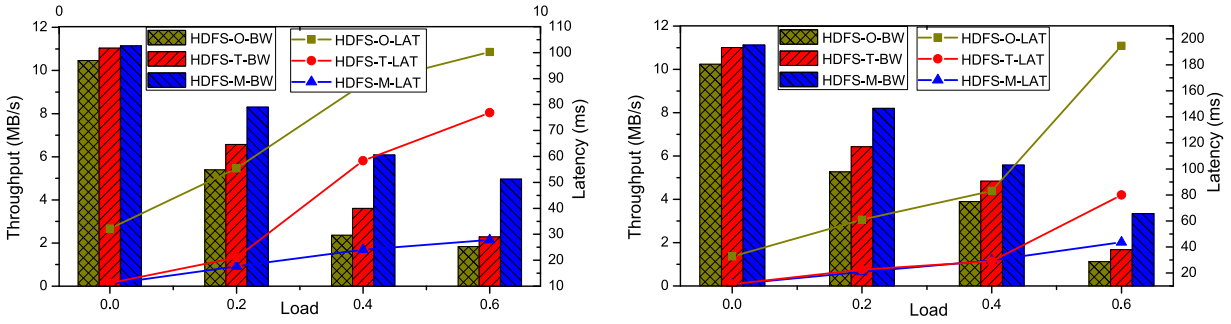
5.3. Application-based evaluation

We carry out performance comparison among HDFS-O, HDFS-T and HDFS-M under background traffic in patterns of the web search workload in both the Fat-Tree and Leaf-Spine topologies. 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 one name node and multiple data nodes on different hosts, and perform tests on the name node host. All data nodes store data on Ramdisk.

We conduct the evaluations in both the separated and shared setups, which are generated in the same way as in Section 5.2. **First,** we compare HDFS-O, HDFS-T and HDFS-M under separated setup with three replicas, varying the load from 0 to 0.6. When load is 0, there is no background traffic. **Second,** we compare HDFS-O, HDFS-T and HDFS-M under shared setup with load 0.2, varying the replica size from 3 to 6.

We measure two metrics, the per-packet latency and the overall throughput, with the packet size 64 KB, data size 300 MB. The per-packet latency is a microbenchmark, which reflects the write latency when the request size is small. Both the throughput and the latency are important for HDFS.

Fig. 16 depicts the results under separated setup, from which we



(a) Fat-Tree topology (b) Leaf-Spine topology

Fig. 16. Performance comparison among HDFS-O, HDFS-T and HDFS-M in Fat-Tree and Leaf-Spine topologies, in separated setup.

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, compared with HDFS-O, HDFS-M achieves ~50% – 1.6× better throughput and ~50% – 72% lower per-packet latency in both Fat-Tree and Leaf-Spine topologies under web search background traffic. Third, HDFS-M is almost analogous to HDFS-T without background traffic, and improves the throughput by ~20% – 1.1×, and decreases the per-packet latency by ~10% – 45% in both Fat-Tree and Leaf-Spine topologies under web search background traffic.

Fig. 17 depicts the results in shared setup with different replica sizes. Similar to the former evaluation, HDFS-M outperforms the alternative schemes and the performance improvements remain as the replica size increases.

HDFS-M outperforms HDFS-O and HDFS-T in both latency and throughput 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 probability of being affected by the background traffic is smaller, which can result in significant latency reduction and throughput improvement in HDFS-M. 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 (both latency and throughput) 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 the background traffic. As a result, HDFS-M can achieve better throughput than the alternative schemes.

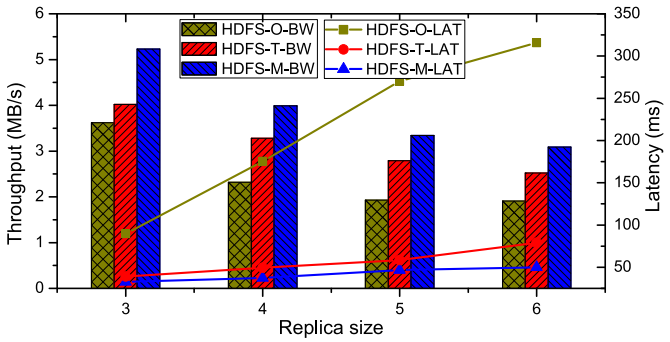


Fig. 17. Performance comparison among HDFS-O, HDFS-T and HDFS-M in different replica sizes.

5.4. 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. 18 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 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 2 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

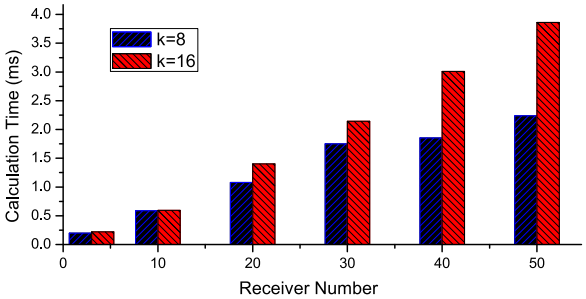


Fig. 18. Calculation times for different receiver numbers.

Table 2
Calculation times in different topologies with three receivers.

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

Table 3
Adjustment times for various groups g in different topologies with three receivers.

Topology	g = 10 (ms)	g = 100 (ms)	g = 1000 (ms)	g = 10,000 (ms)
k = 8	12.458	75.695	491.822	5189.126
k = 16	62.951	111.259	635.594	5453.479

Table 4
CPU usage and network overhead in SDN controller for different groups g with three receivers.

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

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 $O(|L| + |G|\cdot O(MST))$, where the $|L|$ is number of links, the $|G|$ is the number of groups which need to be adjusted, and the $O(MST)$ is the complexity of calculating a single MST. For the MPH algorithm, the time complexity of calculating an MST is $O(mn^2)$, where m is the receiver number of a multicast group, and n is the total number of the network nodes. Table 3 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 4, the computation and network overhead are negligible.

6. 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. This work is also supported by NSFC No. 61232004, No. 61502190, No. 61173043 and State Key Laboratory of Computer Architecture, No. CARCH201505. Dan Feng is the corresponding author.

References

Adamson B., Bormann, C., Handley, M., Macker, J., 2009. Nack-oriented reliable multicast (norm) transport protocol. rfc5740 (November).
Al-Fares, M., Loukissas, A., Vahdat, A., 2008. A scalable, commodity data center network architecture. In: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM'08, ACM, New York, NY, USA, pp. 63–74. http://dx.doi.org/10.1145/1402958.1402967.
Alizadeh, M., Edsall, T., 2013. On the data path performance of leaf-spine datacenter fabrics. In: 2013 IEEE Proceedings of the 21st Annual Symposium on High-Performance Interconnects, HOTI'13, IEEE Computer Society, Washington, DC, USA, pp. 71–74. http://dx.doi.org/10.1109/HOTI.2013.23.
Alizadeh, M., Greenberg, A., Maltz, D.A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., Sridharan, M., 2010. Data center tcp (dctcp). In: SIGCOMM, ACM, New York, USA, pp. 63–74.
Bai, W., Chen, K., Wang, H., Chen, L., Han, D., Tian, C., 2015. Information-agnostic flow scheduling for commodity data centers. In: NSDI, USENIX Association, Oakland, CA, pp. 455–468.
Benson, T., Akella, A., Maltz, D.A., 2010. Network traffic characteristics of data centers in the wild. In: IMC, ACM, New York, USA, pp. 267–280.
Cain, B., Deering, D.S.E., Fenner, B., Kouvelas, I., Thyagarajan, A., 2015. Internet Group Management Protocol, Version 3, IETF RFC 3376 (Oct.). http://dx.doi.org/10.17487/rfc3376.
Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numer. Math. 1 (1), 269–271. http://dx.doi.org/10.1007/BF01386390.
Estrin, D., Farinacci, D., Helmy, A., Thaler, D., Deering, S., Handley, M., Jacobson, V., Liu, C., Sharma, P., Wei, L., 1997. Protocol independent multicast-sparse mode (pim-sm): Protocol specification, rFC2117 (June).
Floyd, S., Jacobson, V., Liu, C.-G., McCanne, S., Zhang, L., 1997. A reliable multicast framework for light-weight sessions and application level framing. IEEE/ACM Trans. Netw. 5 (6), 784–803. http://dx.doi.org/10.1109/90.650139.
Floyd, R.W., 1962. Algorithm 97: shortest path. Commun. ACM 5 (6), 345. http://dx.doi.org/10.1145/367766.368168.
Ge, J., Shen, H., Yuepeng, E., Wu, Y., You, J., 2013. An openflow-based dynamic path adjustment algorithm for multicast spanning trees. Secur. Priv. Comput. Commun. (TrustCom), 1478–1483. http://dx.doi.org/10.1109/TrustCom.2013.179.
Ghemawat, S., Gobioff, H., Leung, S.-T., 2003. The google file system. SIGOPS Oper. Syst. Rev. 37 (5), 29–43. http://dx.doi.org/10.1145/1165389.945450.
Greenberg, A., Hamilton, J.R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D.A., Patel, P., Sengupta, S., 2009. Vl2: A scalable and flexible data center network. In: SIGCOMM, ACM, New York, USA, pp. 51–62. http://dx.doi.org/10.1145/1592568.1592576.
Hadoop, 2016. ⟨https://hadoop.apache.org⟩.
Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., McKeown, N., 2012. Reproducible network experiments using container-based emulation, In: CoNEXT, ACM, New York, NY, USA, pp. 253–264. http://dx.doi.org/10.1145/2413176.2413206.
Iyer, A., Kumar, P., Mann, V., 2014. Avalanche: Data center multicast using software defined networking. In: COMSNETS, pp. 1–8. http://dx.doi.org/10.1109/COMSNETS.2014.6734903.
Jeacle, K., Crowcroft, J., 2005. Tcp-xm: unicast-enabled reliable multicast. In: ICCCN, pp. 145–150. http://dx.doi.org/10.1109/ICCCN.2005.1523829.
Kim, K.S., ping Li, C., Modiano, E., 2014. Scheduling multicast traffic with deadlines in wireless networks. In: INFOCOM, pp. 2193–2201. http://dx.doi.org/10.1109/INFOCOM.2014.6848162.
Lehman, L., Garland, S., Tennenhouse, D., 1998. Active reliable multicast. In: INFOCOM, 2, pp. 581–589. http://dx.doi.org/10.1109/INFCOM.1998.665078.
Li, D., Xu, M., chen Zhao, M., Guo, C., Zhang, Y., Wu, M.-Y., 2011. Rdcm: Reliable data center multicast. In: INFOCOM, pp. 56–60. http://dx.doi.org/10.1109/INFCOM.2011.5935228.
Liang, S., Cheriton, D., 2002. Tcp-smo: extending tcp to support medium-scale multicast applications. In: INFOCOM, pp. 1356–1365. http://dx.doi.org/10.1109/INFCOM.2002.1019386.
Marcondes, C.A.C., Santos, T., Godoy, A.P., Viel, C.C., Teixeira, C.A.C., 2012. Castflow: Clean-slate multicast approach using in-advance path processing in programmable networks. In: ISCC, IEEE, pp. 94–101.
McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., 2008. Enabling innovation in campus networks. SIGCOMM 38 (2), 69–74. http://dx.doi.org/10.1145/1355734.1355746.
openpgm, 2016. ⟨http://code.google.com/p/openpgm⟩.
Paul, S., Sabnani, K.K., Lin, J.C.H., Bhattacharyya, S., 1997. Reliable multicast transport protocol (rmtp). IEEE J. Sel. Areas Commun. 15 (3), 407–421. http://dx.doi.org/10.1109/49.564138.
Rizzo, L., 2000. Pgmcc: A tcp-friendly single-rate multicast congestion control scheme. SIGCOMM, ACM, New York, NY, USA, pp. 17–28. http://dx.doi.org/10.1145/347059.347390.
Ryu, 2016. ⟨http://osrg.github.io/ryu⟩.
Shen, S.H., Huang, L.H., Yang, D.N., Chen, W.T., 2015. Reliable multicast routing for software-defined networks. In: 2015 IEEE Conference on Computer Communications (INFOCOM), pp. 181–189. http://dx.doi.org/10.1109/INFOCOM.2015.7218381.
Shvachko K., Kuang, H., Radia, S., Chansler, R., 2010. The hadoop distributed file system. In: MSST, pp. 1–10. http://dx.doi.org/10.1109/MSST.2010.5496972.
Spark, 2016. ⟨http://spark.apache.org⟩.
Speakman, T., Crowcroft, J., Gemmell, J., Farinacci, D., Lin, S., 2001. Pgm reliable transport protocol specification. rFC3208 (December).

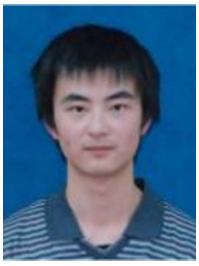
Takahashi, H., Matsuyama, A., 1980. An approximate solution for the steiner problem in graphs. Math. Jpn. 24 (6), 573–577.
Talpade, R., Ammar, M., 1995. Single connection emulation (sce): an architecture for providing a reliable multicast transport service. In: Distributed Computing Systems, pp. 144–151. http://dx.doi.org/10.1109/ICDCS.1995.500013.
Visoottiviseth, V., Mogami, T., Demizu, N., Kadobayashi, Y., Yamaguchi, S., 2001. M/tcp: The multicast-extension to transmission control protocol. In: ICACT, Muju, Korea, Feb.
Wang, Z., Crowcroft, J., 1996. Quality-of-service routing for supporting multimedia applications. IEEE J. Sel. Areas Commun. 14 (7), 1228–1234. http://dx.doi.org/10.1109/49.536364.
Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D.E., Maltzahn, C., 2006. Ceph: A scalable, high-performance distributed file system. OSDI, USENIX Association, Berkeley, CA, USA, pp. 307–320.
Xia, Y., Ng, T.S.E., Sun, X.S., 2015. Blast: Accelerating high-performance data analytics applications by optical multicast. In: 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, pp. 1930–1938. http://dx.doi.org/10.1109/INFOCOM.2015.7218576.
Yang, Y., Qin, Z., Li, X., Chen, S., 2012. Ofm: a novel multicast mechanism based on openflow. Adv. Inf. Sci. Serv. Sci. 4 (9), 278–286.
Yavatkar, R., Griffoen, J., Sudan, M., 1995. A reliable dissemination protocol for interactive collaborative applications. In: MULTIMEDIA, ACM, New York, USA, pp. 333–344.
Zhu, T., Wang, F., Hua, Y., Feng, D., Wan, Y., Shi, Q., Xie, Y., 2016. Mctcp: Congestion-aware and robust multicast tcp in software-defined networks. In: IEEE/ACM Proceedings of the 24th International Symposium on Quality of Service (IWQoS), pp. 1–10. http://dx.doi.org/10.1109/IWQoS.2016.7590433.

Tingwei Zhu He received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2012. He is currently a PhD student majoring in Computer Architecture in HUST. His current research interests include software-defined networking and distributed storage systems. He has several publications in international conferences, including IWQoS and ICPP.

Dan Feng She received the BE, ME, and PhD degrees in Computer Science and Technology in 1991, 1994, and 1997, respectively, from Huazhong University of Science and Technology (HUST), China. She is a professor and vice dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 100 publications in major journals and international conferences, including IEEE-TC, IEEE-TPDS, ACM-TOS, JCST, FAST, USENIX ATC, ICDCS, HPDC, SC, ICS, IPDPS, and ICPP. She serves on the program committees of multiple international conferences, including SC 2011, 2013 and MSST 2012. She is a member of IEEE and a member of ACM.

Fang Wang She received her BE degree and Master degree in computer science in 1994, 1997, and Ph.D. degree in computer architecture in 2001 from Huazhong University of Science and Technology (HUST), China. She is a professor of computer science and engineering at HUST. Her interests include distribute file systems, parallel I/O storage systems and graph processing systems. She has more than 50 publications in major journals and international conferences, including FGCS, ACM TACO, SCIENCE CHINA Information Sciences, Chinese Journal of Computers and HiPC, ICDCS, HPDC, ICPP.

Yu Hua He received the BE and PhD degrees in computer science from the Wuhan University, China, in 2001 and 2005, respectively. He is currently a professor at the Huazhong University of Science and Technology, China. His research interests include computer architecture, cloud computing and network storage. He has more than 80 papers to his credit in major journals and international conferences including IEEE Transactions on Computers (TC), IEEE Transactions on Parallel and Distributed Systems (TPDS), USENIX ATC, USENIX FAST, INFOCOM, SC, ICDCS, ICPP and MASCOTS. He has been on the organizing and program committees of multiple international conferences, including INFOCOM, ICDCS, ICPP, RTSS and IWQoS. He is a senior member of the IEEE and CCF, a member of ACM, and USENIX.



Qingyu Shi He received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014. He is currently a PhD student majoring in Computer Architecture in Wuhan National Laboratory for Optoelectronics (WNLO). His current research interests include software-defined networking and network storage system.

Yanwen Xie He received the BE degree in computer science and technology from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2012. He is currently a PhD student majoring in Computer Architecture in HUST. His current research interests include erasure coding, distributed storage systems and big-data parallel computing.

Yong Wan He is currently an assistant professor in the School of Computer Engineering at the JingChu University of Technology. He obtained his Ph.D. degree in Computer Science from HuaZhong University of Science and Technology, China, in 2013. His current research research interests include Computer networks and protocols, High Performance Network Cluster, Parallel and Distributed Systems.