# Scalable and Adaptive Software Defined Network Management for Cloud-hosted Group Communication Applications

Prithviraj Patil, Akram Hakiri, Shashank Shekhar, Aniruddha Gokhale

▶ **To cite this version:**

Prithviraj Patil, Akram Hakiri, Shashank Shekhar, Aniruddha Gokhale. Scalable and Adaptive Software Defined Network Management for Cloud-hosted Group Communication Applications. 10th IEEE/ACM International Conference on Utility and Cloud Computing UCC 2017, Dec 2017, Austin, United States. 10.1145/3147213.3147220 . hal-01633339

**HAL Id: hal-01633339**

**https://hal.science/hal-01633339v1**

Submitted on 14 Nov 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Scalable and Adaptive Software Defined Network Management for Cloud-hosted Group Communication Applications

Prithviraj Patil*
The MathWorks Inc
Natick, Massachusetts, USA
prithviraj6116@gmail.com

Akram Hakiri
Univ de Carthage, ISSAT
Mateur, Bizerte, Tunisia
akram.hakiri@gmail.com

Shashank Shekhar and Aniruddha Gokhale
Dept of EECS, Vanderbilt University
Nashville, Tennessee, USA
{shashank.shekhar,a.gokhale}@vanderbilt.edu

## ABSTRACT

Group communications form the primary communication pattern for many cloud-hosted applications and cloud infrastructure management services, such as system health monitoring, multimedia distribution, collaborative applications and distributed databases. Although IP multicast has been used to support group communication semantics in diverse Internet-based distributed applications, its deployment in cloud Data Center Networks (DCNs) has been limited due to its higher resource consumption, scalability, and stability issues, which in turn degrades the utility of the cloud. Software Defined Networking (SDN) has enabled the re-engineering of multicast capabilities to overcome these limitations. To that end, this paper presents an autonomous, dynamic and flexible middleware solution called SDN-based Multicast (SDMC), which provides both network load-aware and switch memory-efficient group communication semantics in DCNs. Thus, SDMC improves DCN resource utilization while allowing applications to remain agnostic to the underlying group communication semantics by efficiently toggling between unicast and multicast in accordance with changing network bandwidth and switch memory usage. Empirical studies comparing SDMC with traditional IP multicast shows up to 60% better latency performance for different DCNs topologies, and up to 50% better performance in the switch memory utilization for multicast groups exceeding size 30.

## CCS CONCEPTS

• **Networks** → **Network resources allocation**; **Cloud computing**; **Data center networks**; • **Computer systems organization** → *Cloud computing*; *Fault-tolerant network topologies*;

## KEYWORDS

Cloud Computing; Adaptive Multicast; Software Defined Network; OpenFlow; Data Center Networks.

## 1 INTRODUCTION

Many cloud infrastructure management tasks as well as cloud-hosted applications require group communication semantics in cloud Data Center Networks (DCNs) [3]. For example, DCNs must offer diverse elasticity techniques to provide commodity and tenant-based services for scaling up or down of computing, storage and network resources, e.g., Amazon Elastic Compute Cloud (Amazon EC2) and services like Twitter and Facebook, which use multicast-centric architectures. Additionally, several fault management strategies, such as passive and active replication, failover, state synchronization of cluster servers and quorum management in active replication require group communications to tolerate and mitigate faults [37].

Group communication is also used to enforce security solutions in DCNs, such as prevention, detection and removal of malware and viruses, as well as in performing bulk software installment, update, and upgrade [28]. For example, software management tools for cloud infrastructure like Chef send the same commands to multiple (i.e., 100s of) virtual machines during software installation and update. Likewise, access control policies in DCNs use group communication to manage users, groups, passwords, and user privileges [30].

Two key considerations dictate the performance of group communications in DCNs. First, the router/switch's available network capacity (i.e., available bandwidth) is required to estimate the bandwidth available for data traffic and control messages that can be forwarded along their multiple ports concurrently [32]. Hence, the ability to instrument the router's (or switch's) network capacity is crucial to estimating the network load in terms of the overall bandwidth that a router/switch is able to support. Second, since the memory of the routers/switches (including their queuing buffers) holds packets and connection state information of the traffic transiting across the network devices, it is important to maintain the router/switch memory utilization under a given threshold to avoid buffer overflows [12].

Group communication semantics for contemporary DCN-hosted applications are often realized using multicast, e.g., IP multicast (IPMC), which is a commonly used group communication protocol in the Internet to support multi-point communication requirements so as to conserve bandwidth and reduce the load on servers [23]. IPMC, however, incurs substantial security, performance and scalability degradation in DCNs, e.g., IPMC can be exploited for distributed Denial-of-Service attack [1]. Moreover, existing multicast routing algorithms, such as Protocol-Independent Multicast (PIM) (e.g., PIM Sparse Mode (PIM-SM) and PIM Dense Mode (PIM-DM)) and Multicast Open Shortest Path First (MOSPF), require substantial manual deployment and management efforts by the cloud operators [19]. Furthermore, the Internet Group Management Protocol (IGMP) [7], which is used by IPMC for dynamic membership registration, sends multiple messages to all the routers to notify the occurrence of group events, which requires significant CPU resources, and substantial memory resources to hold the table size for IP multicast [26].

A further downside with IPMC concerns its limited state space in commodity routers which impedes router functionality since they

*Work conducted as part of doctoral studies at Vanderbilt University.

must maintain routing states and perform a costly and wasteful per-group translation [31]. Hence, packet filtering becomes ineffective in large multicast groups and in turn the overwhelmed receivers will begin dropping packets. Although several improvements, such as Hierarchy-Aware multicast [25], multi-path routing [18], and Bloom filter-based group management [15], address these issues, they cannot be readily adopted in DCNs due to their complexity, security issues, and the manual efforts required in managing them.

Being cognizant of the switch memory utilization as well as the network load to avoid drastic problems is a critical requirement for any group-based communication semantics in DCNs. Complicating this requirement is the fact that DCNs often comprise a very large number of distributed network equipment, which makes it harder to instrument, coordinate and maintain a consistent and global view of the system without developing additional and complex solutions. It is in this context that emerging approaches in the form of Network Function Virtualization (NFV) or network softwarization and Software-Defined Networking (SDN) [13] hold promise [27]. NFV, for instance, aims to virtualize a set of network functions by moving network function into software packages, meaning that building a service chain no longer requires acquiring hardware. Complementing NFV is SDN which separates the control plane from the data or forwarding plane. NFV can enable multicast routing on SDN by constructing a traffic forwarding topology, deploying the required multicast functions and steering traffic through the constructed multicast trees [35]. Recent efforts [5, 9] have used SDN for efficient bandwidth management during the creation of multicast trees by using topology information in DCNs. Despite this promise, current approaches cannot autonomously adapt to the network load and router (or SDN switch) memory utilization, which is critical to the scalability of group communications, and also for monitoring and evaluating network performance in DCNs.

To leverage the benefits of NFV/SDN while addressing unresolved challenges and limitations in recent SDN-based multicast efforts, we propose an autonomic Cloud resource management architecture called SDN-based Multicast (SDMC) to achieve utility in clouds by supporting dynamic and flexible, network load-aware and switch memory-efficient group communications in DCNs. SDMC is not a new multicast protocol; rather it is a SDN-enabled distributed middleware framework for improving utility in DCNs that provides dynamic resource management by intelligently and dynamically switching different group communications flows between unicast and multicast thereby balancing between network bandwidth and switch memory utilization. SDMC is fully decoupled from any specific multicast protocol, such as IPMC. Moreover, the middleware aspect enables a large number of DCN-based group communication applications and services to avail of our adaptive resource management solution.

The key contributions of these paper are:

- We describe the architectural innovations and algorithms in SDMC that provide an autonomous, adaptive and flexible network link and switch memory load-aware multicast approach for data center networks.
- We present details of our distributed SDMC middleware, which is manifested in the form of (i) a suite of SDN network applications hosted on a SDN controller and OpenFlow-enabled switches [24], and (ii) a SDN middleware layer on the network hosts where applications reside.
- We evaluate the effectiveness of our approach in different data center network topologies along a number of metrics, such as load variations and switch-memory utilization.

The rest of the paper is organized as follows: Section 2 describes the design and implementation details of SDMC. Section 3 provides an empirical evaluation of SDMC. Section 4 discusses related work and compares them to SDMC. Finally, Section 5 presents concluding remarks alluding to lessons learned and future work.

## 2 DESIGN AND IMPLEMENTATION OF SDMC

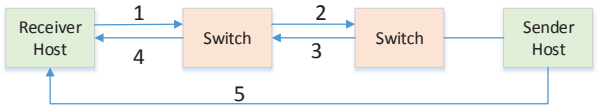
This section delves into the details of SDMC. We first present three key contributions in SDMC required to support application-agnostic dynamic and adaptive resource management, and then present the architectural details that enable us to realize utility in the cloud. Subsequently we present the algorithms that use the framework to perform an adaptive, network link and switch memory load-aware multicast for data center networks.

### 2.1 Contribution 1: Two-level SDMC-ID Structure

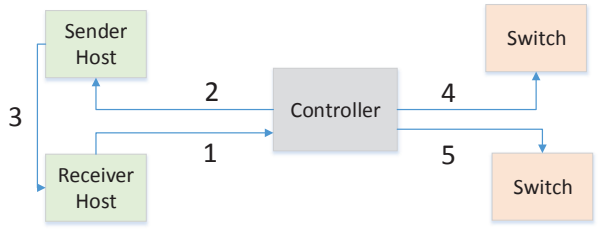
A key requirement for SDMC is to support application-agnostic adaptive behavior and promote the reuse of multicast routing trees. To that end, SDMC defines a new multicast identifier space called SDMC-ID.[1] The SDMC-ID space is divided into two regions: application-level (or external, i.e., $ID_e$) and network-level (or internal, i.e., $ID_i$). Applications using SDMC interact using only the external SDMC-IDs while the network data path deals only with internal SDMC-IDs. To keep applications agnostic of internal IDs, SDMC supports a translation layer in the form of a middleware on each host (see Section 2.4). This two-level SDMC-ID structure allows SDMC to use the same multicast routing tree (with the same internal SDMC-ID) with overlapping receivers belonging to two different external SDMC IDs.

SDMC maintains the mapping between the two types of IDs by encoding the participants' external multicast channel IDs and the node-specific internal IDs, and storing it in each immediate SDN router/switch without requiring packet header modification. As shown in Figure 1, traditional multicast uses the same multicast ID for all the communication needs. In contrast, SDMC uses the external SDMC multicast ID for communication between application endpoints (i.e., sender and receiver) and network endpoints (i.e., switches or controller). For the communication between the controller and switches, SDMC uses the corresponding internal multicast ID. Hence, the communication over lines 1, 2 and 3 uses external (or application-level) multicast ID while the communication over lines 4 and 5 uses internal (or network-level) multicast ID.

[1] We use SDMC-ID to denote both the identifier space for SDN-enabled multicast as well as the actual external or internal IDs assigned to a flow. Its semantics should be evident from the context.




(a) Traditional Multicast Topology




(b) Architecture of the SDN-enabled Multicast Approach

**Figure 1: Comparison between the SDN-enabled Multicast and Traditional IP Multicast**

## 2.2 Contribution 2: Lazy Initialization Strategy

Our second contribution lies in the initialization process for the SDMC senders/receivers and the SDMC routing tree creation, which uses a lazy approach to reduce the initial latency that is otherwise incurred by the receivers in traditional multicast and also to allow flexibility in adapting dynamically to the network load and switch memory. SDMC exhibits this lazy approach in its operation while switching to multicast communication from the default unicast. Specifically, the immediate SDN router/switch does not rely on multicast routing information right away; rather it can be stateless for forwarding since it uses existing unicast routing information.

This lazy approach manifests itself in three different situations. First, when a new receiver requests to listen on an external SDMC-ID, it is not immediately added to the SDMC-ID as a multicast receiver but rather as a unicast destination for all the existing senders of that SDMC-ID, if any. Second, the SDMC multicast routing tree for a new receiver is created in the controller but is not installed (e.g., in the form of OpenFlow rules [4]) in the switches right away. Third, when a receiver (or sender) leaves the SDMC-ID group, the multicast tree is not updated immediately. All these decisions (i.e., when to add a receiver as a multicast destination; when to install the multicast routing tree in switches; and when to update the multicast routing tree after a receiver leaves) are taken by SDMC holistically based on all other SDMC sender/receiver statuses and depending on the network load and switch-memory utilizations (see the algorithms in Section 2.5).

Figure 2 shows the lazy initialization of a sender. It shows the setup when two receivers subscribe to this sender. Even though the sender has a multicast ID attached to it, receivers are added as a unicast destination to reduce the start-up latency. This enables the receiver to receive packets immediately since there is no need to create any multicast routing tree in the switches. The corresponding behavior on the receiver side appears in Figure 3. As seen from

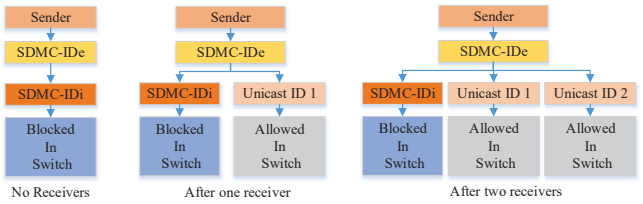



**Figure 2: Initial SDMC Sender Setup showing External and Internal SDMC-IDs**

these figures, both the sender and receiver use only the external SDMC-ID, i.e., $ID_e$ in Figures 2 and 3, which is mapped either to an internal SDMC-ID (multicast), i.e. $ID_i$ in Figures 2 and 3 or a unicast, transparently to the application layer.

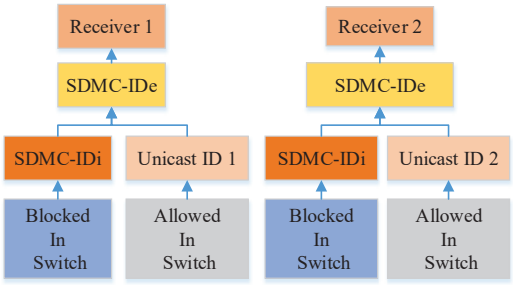



**Figure 3: Initial SDMC Receiver Setup**

## 2.3 Contribution 3: Network Link and Switch Memory Monitoring

To make dynamic resource management decisions, SDMC must periodically keep track of the utilizations of network links and switches. SDN switches use counters for tracking the number of packets that are sent through a given link and record link forwarding success and failure. This is used to track the bandwidth utilization and detect if certain packets are consuming more bandwidth than anticipated. Using these counters, the controller can make a decision about the link utilization. To that end, SDMC provides a special capability in the SDN controller via a network management application (see Section 2.4). SDMC then populates the network link utilization information against the SDMC-IDs which are using that link for unicast for a given receiver. The network topology corresponding to this scenario is shown in Figure 4.

SDMC also keeps track of the memory utilization of the network switches with the help of the controller. Since the controller installs rules in the switches, it knows exactly how many OpenFlow rules exist on each of the switches. Each switch comes with a maximum number of OpenFlow rules that it can accommodate. So we measure the switch-memory utilization as the number of actual OpenFlow rules installed in the switch against the maximum number of OpenFlow rules allowed.

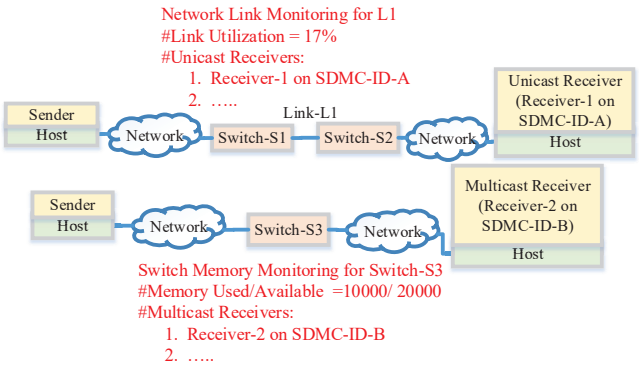



**Figure 4: Network Link and Switch Memory Monitoring**

## 2.4 Putting it Together: SDMC Architecture and Middleware Design

To support the key capabilities of SDMC outlined above, SDMC is realized as a distributed SDN-based middleware framework shown in Figure 5, which comprises the following: the core intelligence of SDMC is made available as a suite of SDN applications (called SDN NetApps in SDN parlance) that execute inside a (logically) centralized controller as shown in the figure; and as a SDN middleware layer available on each host where the senders and receivers use the APIs provided by SDMC. SDMC uses the traditional OpenFlow channel for communication between the controller and SDN routers, where these routers are connected to host machines (physical or virtual). The use of SDN principles allows SDMC to be flexible and dynamic. The remainder of this section explains each component.

**SDMC SDN controller:** The DCN is managed by a (logically) centralized SDN controller, which is placed on a dedicated machine(s) with dedicated out-of-band connections to all the OpenFlow-enabled routers. The SDMC-specific SDN NetApps hosted on the controller provide the core intelligence of SDMC as follows:

- *Topology Discovery:* This NetApp provides autonomous discovery of joining and leaving routers and hosts. It uses the link layer discovery protocol (LLDP) to assist the controller in identifying the joining or leaving multicast group participants.
- *Topology Management:* This NetApp creates a graph topology of all nodes connected to the controller. By manipulating the connected graph, the controller can activate or de-activate the connected routers, hosts and the links connecting them.
- *Monitoring:* This NetApp provides real-time reports on various network parameters, such as bandwidth utilization, latency, packet error rates, resource utilization, etc to the controller.
- *Routing:* This NetApp implements the multicast intelligence (explained in Section 2.5.3) to build the dynamic multicast routing tree at run-time.
- *Resource Manager:* This NetApp manages the SDMC ID space and the mapping between the external and internal IDs (explained in Sections 2.5.1 and 2.5.2), and determines and enforces the resource management decisions (explained in Section 2.5.4).

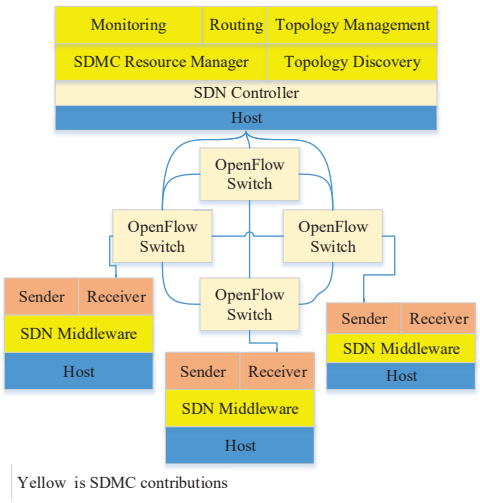



**Figure 5: SDMC Architecture**

**OpenFlow-enabled switches:** In a typical DCN, a number of OpenFlow-enabled switches are connected to form a network with topologies like mesh, tree or jellyfish. Each switch contains an OpenFlow client to connect to the SDN controller. The OpenFlow client conforms to the OpenFlow protocol by providing capabilities to add, remove, update, and delete packets inside the network devices. The OpenFlow logic is designed in terms of flow processing controlled by a group of flow tables. Flow tables are composed of a set of entries to process packets whose headers match predefined patterns in the header field. The OpenFlow protocol specifies the formats of the messages exchanged between controllers and remote switches through a secure channel so that commands and packets can be exchanged securely. The mapping of the data plane into forwarding tables provides a simple abstraction to describe the common requirements of network equipment to store, process, and forward packets hop-by-hop. Thus, the abstraction provided by the OpenFlow protocol enables the use of open and standardized interfaces that can be used to program network devices in a vendor-agnostic manner.

**Host machines and SDMC middleware:** Multicast functionalities, which deal with overlay multicast, are implemented using SDMC's SDN middleware that runs in each of the host machines. The SDN middleware controls the communication using a host manager service to translate endpoints listening on multicast IDs into multiple unicast IDs, and switch endpoints from unicast to multicast and vice-versa transparently to the applications. Those endpoints can decide whether senders or receivers are able to send or listen only on SDMC IDs or not.

**Host manager:** The host manager NetApp (not shown) keeps track of all machines connected to the network. This application is used by SDMC to communicate with the SDMC middleware on the host machines. This application is required since we build a hybrid multicast protocol by combining application-level multicast (or overlay multicast) and native network-level-multicast.

**SDMC participants:** The SDMC participants (senders and receivers) run on top of the SDMC middleware on the hosts.

## 2.5 SDMC Behavior and its Dynamic Resource Management Algorithms

We now describe the runtime operation of SDMC and explain its dynamic resource management algorithms. We illustrate these operations in the form of sequence of activities executed by SDMC in response to various events like sender and receiver join, and sender and receiver leave.

*2.5.1 Algorithm for Sender Join.* When a participant (sender) wants to send data on an application-level SDMC-ID, $M^e$, the following sequence of steps described in lines 4–10 of Algorithm 1 are carried out.

**Algorithm 1:** Joining Multicast Group

```
   Data: M^e, M^i, U_r1, U_r2
   Result: sender or receiver joined SDMC
1  senderJoinRequest(M^e);
2  receiverJoinRequest(M^e, U_r1);
3  while Listening do
4      if sender then
5          if firstsender then
6              create new M^i for M^e; installOFTransRule(M^e → M^i) in this sender
                 and in all the existing receivers;
7          else
8              retrieve M^i for M^e; installOFTransRule(M^e → M^i) in this sender;
9          end
10         installOFTransRule(M^e → U_r1, U_r2, ...) in all the senders;
11     else
12         addUnicastDestToSenders(U_r1);
13         nonBlocking_AddToMulticastTree(U_r1);
14         waitForTriggerToToggleToMulticast(U_r1);
15     end
16 end
```

- The sender uses its SDN middleware to forward the request to the SDMC Routing NetApp. The latter assigns an appropriate internal SDMC-ID $M^i$ to correspond to the requested application-level SDMC ID, $M^e$. The SDMC Routing NetApp at the controller side installs an OpenFlow rule in the edge switch of the sender to block all the traffic with destination ID $M^i$.
- Thereafter, if this sender is the first sender for $M^e$, then the SDN middleware on the sender node (and for all the existing receivers) installs a translation rule for $M^e \leftrightarrow M^i$ in the host middleware so that (1) when sender sends packets on external SDMC-ID $M^e$, it gets translated to internal SDMC ID $M^i$ and also (2) when any receiver receives the packet with SDMC-ID $M^i$, it gets translated to application-level external SDMC-ID $M^e$.
- Additionally, the SDN middleware of the joining sender installs the translation rule $M^e \rightarrow (M^i, U_{r1}, U_{r2})$, to convert $M^e$ into destinations of the existing unicast receivers, i.e. $U_{r1}$ and $U_{r2}$. This step is performed only if at least one receiver is listening on $M^e$ as a unicast destination. This allows the sender to start sending packets using unicast immediately instead of incurring the delay in traditional multicast.

*2.5.2 Algorithm for Receiver Join.* Similar to the joining of a sender, when a joining receiver wants to listen on an application-level SDMC-ID $M^e$, it requests its SDN middleware to retrieve the internal network-level SDMC-ID $M^i$ from the SDMC SDN NetApp. Then, the SDN middleware installs two translation rules $M^e \leftrightarrow M^i$ and $M^e \leftrightarrow U_r$ (where $U_r$ is the unicast ID of this receiver), while giving preference to the $M^e \leftrightarrow U_r$ rule over $M^e \leftrightarrow M^i$ so that the first rule gets matched. This forces the receiver to listen on the unicast ID instead of multicast ID to begin with. This is part of the lazy initialization of SDMC receivers.

Lines 11–15 of Algorithm 1 illustrate the joining operation for a SDMC receiver. The following sequence of steps are followed:

- When a new receiver (r1) sends a request to listen on $M^e$, the receiver is not immediately added to $M^e$ as a multicast destination. Instead it is added as a unicast destination (as $U_r1$) for all the existing senders of $M^e$.
- Next, SDMC concurrently creates a multicast routing tree for a new receiver as explained in Section 2.5.3. Even though every existing sender of $M^e$ has a multicast ID (in the form of $M^i$) attached to it, new receivers are added as unicast destinations for reducing the startup latency. This allows the joining receiver to receive packets immediately.
- The receiver is transparently switched to multicast only if the network link utilization crosses a threshold limit, as explained in Section 2.5.4.

As part of these steps, the SDMC SDN NetApp searches for all the senders of $M^i$ and adds the unicast destination of $U_r$ in their SDN middleware translation rules. Then, it requests the unicast routing paths for this receiver to every sender of $M^i$ from the Routing module at the SDN controller. Based on these routing paths, it then updates the controller's routing table by adding a sender-receiver pair against each appropriate network link and switch.

*2.5.3 Multicast Tree Calculation.* In the current IP multicast, the number of multicast forwarding states is proportional to the number of multicast groups where the number of multicast groups grow proportionally to the number of forwarding states. Additionally, the number of control messages required to maintain the forwarding states will grow in the same manner. This scalability issue has to be solved before multicast can be deployed over the Internet. To address this issue, SDMC allows optimally aggregating multicast groups so that the controller can aggregate local groups into virtual meta-groups to perform routing tree aggregation and addressing.

To that end, the SDMC Routing NetApp in the SDN controller implements Algorithm 2 to compute the multicast tree. Based on inputs from the topology discovery NetApp, it keeps track of all links in the network and adds them to the link list (L), and creates a list of nodes (N) joining those links as shown in lines 1 to 6. Moreover, since there could be multiple multicast groups in the DCN, SDMC finds the address of the group from the list of all available multicast groups and returns a list of routers in the same group, i.e., the multicast tree (lines 7–9). Subsequently, SDMC calculates the shortest path in the multicast tree and installs the OpenFlow rules for that multicast tree (lines 10 and 11).

Since the DCN environment can carry multiple concurrent multicast sessions, SDMC allows accommodating all the multicast groups while allowing the optimization of the network resource among multiple co-existing multicast trees. As shown in line 8 in Algorithm 2, SDMC allows multicast tree packing based on the least cost tree (Steiner tree) to perform adaptive resource management. In random graphs, such as distributed routers in the Internet, the Steiner tree is known to be NP-Complete. Nonetheless, the routers

**Algorithm 2:** Minimum Steiner Multicast Tree and Route Calculation

```
   Data: M^e, N
   Result: Minimum Steiner Multicast Tree
1  foreach M^e in N do
2      if M^e ! visited then
3          visited.add(M^e);
4          tree.append(M^e);
5      end
6  end
7  for i ← tree[0] to len(tree) do
8      multicastTree= Steiner(tree,M^i, M^e);
9  end
10 installMulticastRules(multicastTree,M^i, M^e);
11 return multicastTree;
```

**Algorithm 3:** Adapting to network link and switch memory utilization

```
   Data: switch(s1), Link(l1), receiver(r1, U_r1), SDMC(M^i)
   Result: Receiver toggled between multicast and unicast & resource utilization kept under
           threshold
1  if ThresholdViolationDetect(switch s1) then
2      addUnicastDestToSender(U_r1);
3      discardDuplicates(r1);
4      listenOnUnicast(r1);
5      removeFromMulticast(U_r1,M^i);
6      stopListenOnMulticast(r1,M^i);
7  end
8  if ThresholdViolationDetect(link l1) then
9      addToMulticast(U_r1,M^i);
10     discardDuplicates(r1);
11     listenOnMulticast(r1,M^i);
12     RemoveUnicastDestFromSender(U_r1);
13     stopListenOnUnicast(r1);
14 end
```

and switches in DCNs are organized as structured graphs so that it is possible to build optimal or near-optimal routing trees using a Steiner tree.

Furthermore, as SDMC can dynamically perform the multicast-to-unicast translation, a source group can send an aggregation packet to the edge switch using the unicast ID, typically IP unicast address, so that the controller installs the required OpenFlow rule to perform multicast group registration. This strategy offers bandwidth efficiency for a large number of OpenFlow rules because the translation happens likely close to the receiver. This way, SDMC drastically reduces the required network states and concentrates them in a single switch at the network edge close to the receiver. Furthermore, the application remains agnostic to the rest of the network topology changes because transparent to the application, SDMC makes trade-offs to retain the benefits of group communication by dynamically switching between unicast and multicast, and effectively limiting the explosion of the network states during the communication.

*2.5.4 Adaptive Resource Management.* SDMC keeps track of network links and their utilization with the help of the SDMC Monitoring NetApp. Then, it populates the link utilization information against the SDMC-IDs which are using that link for the unicast for a receiver. Additionally, SDMC also keeps track of the memory utilization of the network switches with the help of the controller. SDMC reduces switch memory utilization by dynamically switching some multicast receivers to unicast destinations. This is due to the removal from the switch of OpenFlow rules used for multicast routing of those toggled receivers. Similarly, SDMC also reduces link utilization by switching some unicast receivers to multicast destinations, which reduces packet duplication and hence decreases network link utilization. In this way, SDMC trades off link and switch memory utilization keeping both of them below the threshold limits. In current work we do not consider both metrics at once. Moreover, SDMC also does not handle the case when both metrics are high, nor does it handle oscillations in adaptations.

Based on insights from [2], we maintain the threshold limits under 70% of the resource utilization to avoid possible SLA violation while decreasing the energy consumption as soon as possible. To trigger the dynamic adaptations, SDMC registers listener events in the SDMC Monitoring NetApp to get notifications about link and switch memory threshold violations. Algorithm 3 describes the procedure for this dynamic and adaptive resource management.

SDMC continually monitors link and switch memory utilization. As shown in Lines 1–6 of Algorithm 3, when a switch memory reaches the threshold limit, first, SDMC updates the translation rule in the SDN middleware of all the senders by adding in its mapping the unicast address of $r1$, i.e., $M^e \leftrightarrow (M^i, U_{r2})$ would become $M^e \leftrightarrow (M^i, U_{r1}, U_{r2})$. Then, SDMC instructs the receiver to listen on its own unicast ID ($U_{r1}$) along with multicast ID $M^i$. At this point the SDN middleware of $r1$ starts receiving duplicate packets: one each from unicast and multicast destinations. SDMC, however, instructs the SDN middleware of $r1$ to discard any duplicate packets. Hence, the application-level receiver (r1) receives only a single copy of each packet. Now, since $r1$ does not need multicast destination support, SDMC can safely remove OpenFlow entries in the router that were installed before to reach $r1$. This step reduces the switch memory utilization. At this point, $r1$'s SDN middleware again starts to receive only a single copy of each packet but through unicast destination. Now, SDMC instructs $r1$ to stop listening on multicast ID $M_i$ altogether. In this way, $r1$ is toggled from multicast to unicast to reduce the switch memory utilization.

In the same way, when a link utilization crosses threshold limit, SDMC reduces link utilization (Lines 8–13 in Algorithm 3) as follows. First, SDMC updates the multicast routing tree for multicast ID of $r1$ ($M^i$) inside the SDN routers by adding flow entries that allow $r1$ to be reached by all the senders of $M^i$. Then, SDMC instructs the receiver to listen on multicast ID $M^i$ along with its own unicast ID ($U_{r1}$). At this point, the SDN middleware of $r1$ starts receiving duplicate packets: one each from unicast and multicast destination. SDMC, however, instructs the middleware of $r1$ to discard the duplicate packets. Hence, the application level receiver (r1) receives only a single copy of each packet. Then, SDMC updates the translation rule in the SDN middleware of all the senders by removing in its mapping the unicast address of the $r1$, e.g., $M^e \leftrightarrow (M^i, U_{r1}, U_{r2})$ would become $M^e \leftrightarrow (M^i, U_{r2})$. This step reduces the network link utilization as each sender now sends one less copy of each packet. At this point, $r1$'s SDN middleware again starts to receive only a single copy of each packet but through the multicast destination. Now, SDMC instructs $r1$ to stop listening on unicast ID $U_{r1}$ altogether. In this way, $r1$ is switched from unicast to multicast to reduce the network link utilization.

## 3 EXPERIMENTAL EVALUATION

We evaluated SDMC using an emulated DCN testbed comprising Mininet [14] as the network emulator with OpenFlow virtual switches used for creating different DCN topologies. We implemented SDMC using the Python-based POX SDN controller. For senders and receivers, we developed a representative publish-subscribe application which runs as a SDMC participant (sender or receiver) and is hosted on 200 virtual hosts. We also created up to 500 multicast groups for these participants. We arranged these 200 virtual hosts in four different topologies, viz., Jelly-fish, Tree, Fat-tree and Random, which are the most used network topologies in DCNs. We argue that these topologies have diverse dissemination paths that highlight the advantages of SDMC. We evaluate SDMC in terms of initial setup latency, network overhead, switch-memory utilization, and packet loss and compare it against traditional IP multicast.[2]

### 3.1 Average Initialization Setup Time

Multicast service mode needs an initialization setup time to enable receivers to join the group and perform their membership. The initial setup time is measured as the time for processing a receiver's joining-request by the controller. It does not concern actually sending or receiving packets. So the time between some receiver A contacting the controller that it wants to join and the controller replying back with a message of "membership successful" is the initial setup time. In IPMC this time involves updating all sender-to-receiver mcast trees but in SDMC it only involves updating the sender switches. Thus, SDMC's receiver-initiated group membership allows better management of the leaf nodes.

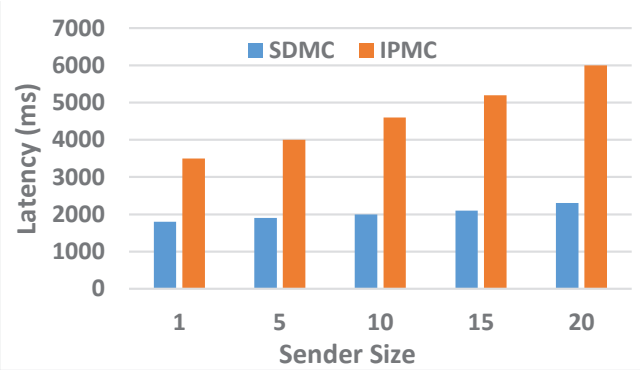
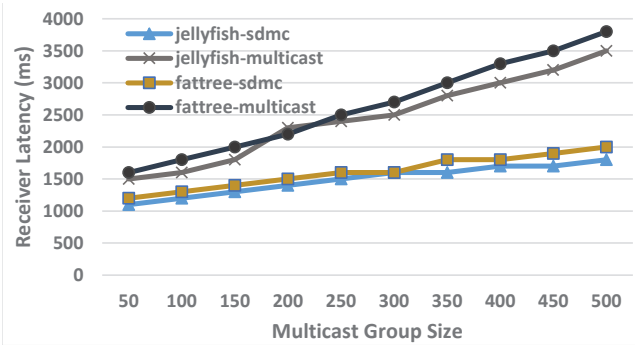
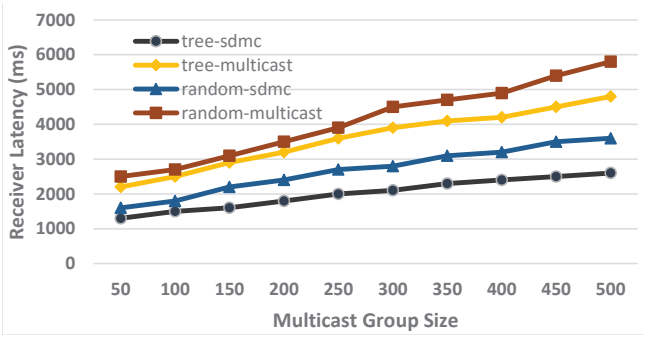



**Figure 6: Setup Latency Induced by Increasing Multicast Senders**

Figure 6 shows the receiver-initiated setup time in terms of number of senders in the group. For this experiment, the sender would send packets with max UDP size (65K) at some constant rate. The transmission rate was same for all the senders (1 Mbps), which was set by the mininet virtual switches and communication was through UDP channel. The figure shows that SDMC incurs less setup latency than IPMC. In particular, in all cases the average initialization latency incurred is at least 50% less than IP multicast. This is due to the fact that SDMC defers creating the routing tree in the controller to a later time. Since the controller has global knowledge of all the joining/leaving nodes, it programs the switches by injecting OpenFlow rules, which allows receivers to join the multicast tree at a later time. This approach is different from traditional IP multicast, which involves IGMP snooping to filter the unwanted multicast packets. These results validate our claims about better latency performance compared to traditional IP multicast.




**(a) JellyFish and FatTree Topologies**




**(b) Tree and Random Topologies**

**Figure 7: Initial Receiver Latency for SDMC and IP Multicast**

Additionally, we measured the initialization setup time for different network topologies in DCNs. Figure 7 shows the setup delay for the most common data center network topologies, viz. jellyfish, tree, flat-tree, and random topologies. It shows that SDMC incurs less setup time compared to traditional multicast in all the four DCN topologies. Again, the evaluation results confirm the efficiency of our SDN-enabled multicast compared to traditional IP multicast.

### 3.2 Adaptiveness to Network Load

To evaluate the awareness and adaptation capabilities of SDMC to router capacity, we performed network load measurements in four of the data center network topologies, i.e., jellyfish, mesh, tree, and random topologies. Figure 8 shows the network load for SDMC and IP multicast. SDMC efficiently switches between unicast and multicast based on network load as described in Algorithm 3. When network load is less, SDMC uses unicast while it switches to multicast when network load increases.

[2]SDMC is decoupled from IPMC. Moreover, comparing SDMC performance with other multicast protocols is part of future work.

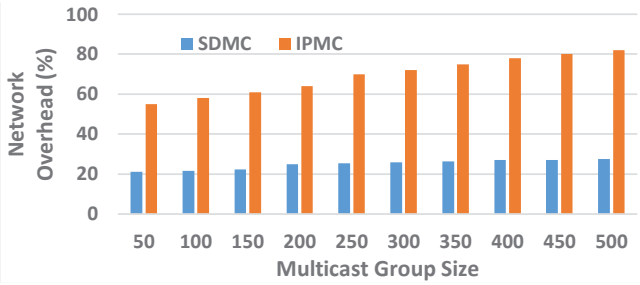



**Figure 8: Network Overhead for SDMC and IP Multicast**

As seen from Figure 8, SDMC adapts to network load by toggling between unicast and multicast. In particular, SDMC incurs up to 25% network overhead when increasing the number of multicast groups up to 500. Conversely, IP multicast incurs up to 80% of network overhead.

### 3.3 Adaptiveness to Switch-Memory Utilization

Figure 9 shows the switch memory utilization for IPMC and SDMC to evaluate the effectiveness of our approach in avoiding the switch buffer overflow. SDMC requires 50% less memory utilization as measured in terms of the number of OpenFlow rules that are stored in switch buffers compared to IPMC once the number of multicast groups exceeds size 30. This is due mainly to the fact that OpenFlow rules are injected by the SDN controller only when missing packets or unrecognized packets transit through the switch. Thereafter, the switch will send a request to the controller to install new rules.

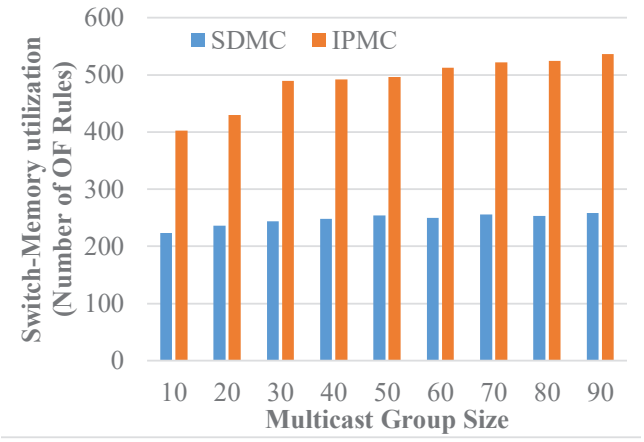



**Figure 9: Switch-Memory Utilization for SDMC and IP Multicast**

Moreover, as described in Algorithm 3, when the switch memory utilization crosses a threshold limit, i.e. 70% of the available resources as described in Section 2.5.4, SDMC toggles to unicast. This helps to reduce load on switch memory buffers resulting in less switch buffer overflows. This approach is different from traditional multicast, which performs stateful packet forwarding, because it needs to maintain the same state across all the transit switches. Our results, show that our approach succeeds in improving switch memory utilization compared to IP multicast.

### 3.4 Evaluating the Packet Loss

We evaluated the packet loss and studied its impact in affecting the network's application behavior. Packet loss can occur when the traffic transmitted to the receivers across a particular link exceeds the capacity of that link. Additionally, another source of packet loss is that short-lived bursts of traffic may occur and deteriorate the network performance for a short time. By characterizing the link utilization through the packet loss we can better investigate the bottlenecks, if any, of our SDN-enabled multicast approach. Note that SDMC ensures that the sender does not send duplicate packets. Hence, during dynamic adaptation, the receiver will not receive any duplicates but can received packets out of order, which we assume will be handled by the application layer since the underlying protocol we use is UDP.

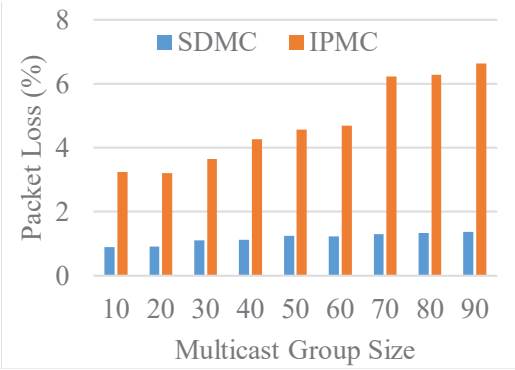



**Figure 10: Packet Loss for SDMC and IP Multicast**

Figure 10 illustrates the packet loss for the IP multicast and compare it to our approach in SDMC. This figure shows that our approach presents a less than 2% of packet loss, which is better than the traditional multicast that experiences more than 6% of packet loss, when the number of multicast groups exceeds size 70. These results demonstrate that our SDN-enabled multicast performs better data transmission compared to IP multicast.

## 4 RELATED WORK

This section describes related work on resource management solutions for group communications in data center networks and compares it to the SDMC solution.

### 4.1 Scalable Multicast Routing in DCNs

Due to the many different challenges incurred in deploying group communications in DCNs, scalable multicast routing has been an important topic of research. In [16], Bloom filter-based group management was introduced to compress multicast routing entries by removing computations of distributed routes from multicast routers. Similarly, BUFFALO [33] distributes IP forwarding tables on each router interface and requires one Bloom filter on each router interface. A similar idea was discussed in [6] to encode the multicast

tree information into in-packet Bloom filters. Despite the advantages of Bloom filter-based multicast in reducing flow entries, such approaches require installing Bloom filters in each switch port to perform the compression actions. Moreover, in typical high-density Ethernet port routers installed in DCNs, it is difficult to maintain efficient memory utilization since dealing with false positives in the lookup incur additional router overhead.

The Hierarchy-Aware multicast framework [25] relies on Distributed Hash Tables (DHTs) to create multicast forwarding trees for dynamic membership management in overlay DCNs. Nonetheless, DHT-like tree creation pays less attention to the links at the bottom of the tree, which results in suboptimal routing when top-down data forwarding is performed. Dual-structure Multicast (DuSM) [5] classifies multicast groups based on their flow granularity. First, large (i.e., elephant) flows are detected and classified in the same multicast trees to which higher bandwidth is allocated. Then, multicast-to-unicast translation rules are performed for all the other short (i.e., mice) flows. Although such an approach allows control over traffic congestion in DCNs, it is difficult to adopt it in latency-sensitive applications associated with bursty mice traffic, which represent most of the traffic transiting the DCNs.

Recent efforts [17, 18] introduced Reliable Data Center Multicast (RDCM) by exploiting the multi-path diversity available in highly connected DCNs. RDCM creates backup overlays for every multicast routing tree, and switches among them as per network load fluctuations. Similarly, a unified unicast-multicast framework was introduced in the Code-Oriented eXplicit multicast (COXcast) project [10], where a unicast path is created to the underlying routers and then multicast trees are created by encoding the output port bitmap of nodes on the path between the source and destination. Our approach is similar to RDCM and DuSM in that SDMC achieves fault tolerant and network-efficient communication by bypassing initial multicast tree creation and switching to unicast instead. Similar to COXcast that uses both a common identifier and a node-specific key to route packets to multiple receivers, SDMC introduces two-level identifiers to interact with multiple end hosts and routers, respectively.

Unlike these approaches, SDMC moves the multicast management logic from routers to a centralized SDN controller, which holds all the intelligence required to manage multicast groups. The SDN controller leverages high-density links in DCNs to provide a global view of the entire network and installs multicast flow entries in the routers.

## 4.2 SDN-enabled Multicast in DCNs

Prior research on multicast issues in SDN-enabled group communications exist. For instance, [26] extends IP multicast using OpenFlow for dynamic group management rather than employing the traditional IGMP protocol [7]. The authors introduce a VXLAN [21] controller to perform Ethernet-over-IP encapsulation over a logical "tunnel." Similarly, in [8], the authors introduce IP-over-Labeled Optical Burst Switching (LOBS) network to encapsulate multicast messages over a Layer 2 tunneling protocol. OpenFlow is used to build and delete multicast trees using a unified SDN control plane.

CastFlow [22] moves IP multicast management to a centralized SDN controller, and IP multicast trees are calculated using OpenFlow. The controller implements the IGMP messaging layer to handle joining/leaving messages when they are received by the first router. Inspired by CastFlow, Software-Defined Multicast (SDM) [29] was introduced to manage live streaming traffic in the ISP's networks. It combines IP multicast functionality with unicast data delivery using OpenFlow to achieve efficient traffic delivery. SMARTFlow [20] uses an OpenFlow controller to calculate multicast trees based on the Spanning Tree algorithm and encapsulates multicast messages in smart grids over Ethernet MAC addresses. The authors in [11] present Scalar-pair Vectors Routing and Forwarding (SVRF), which encodes multicast group addresses into scalar-pair vectors to identify outgoing ports for delivery of unicast and multicast packets.

Using a remote SDN controller to perform multicast management was also introduced in OpenFlow Multicast (OFM) [34]. OFM maintains a multicast rule database to store all flow entries for all routers managed by the controller. OFM holds a state database to store multicast group members and their state information. However, in practice it is hard to forecast all possible flow entries that can match against future arriving multicast group members. The Avalanche Routing Algorithm (AvRA) [9] was introduced to minimize the size of the routing tree. It enables efficient bandwidth management by using a centralized SDN controller to gather topology information in DCNs that do not have traditional IP multicast. Likewise, the authors in [36] propose building multicast trees for video conferencing by using a SDN controller, where source-based multicast trees are constructed for end-to-end packet delivery.

Our approach in SDMC allows managing multicast communications in overlay data center networks using OpenFlow. Compared to CastFlow, SDMC eliminates periodic join/leave messages generated by IGMP. Rather, by using the topology discovery module, the controller keeps track of all joining and leaving nodes in real-time thereby addressing the hardware resource problem. Furthermore, compared to Avalanche, which achieves data rates up to 12% better than IP multicast, in applications deployed in the Portland Fat Tree topology, SDMC demonstrates a more general framework that can address different topologies like tree, random, mesh, and jellyfish. For instance, SDMC's jellyfish topology, which supports more servers than an equal-cost Avalanche fat-tree topology, achieves often up to 60% better average latency. Moreover, Avalanche gets up to 35% reduction, compared to IP multicast, in the number of links that are less than 5% utilized, once the number of multicast groups exceeds 1,000. In contrast, the performance of SDMC is up to 50% better, compared to IP multicast, in the switch memory utilization, once the number of multicast groups exceeds just 50. Additionally, SDMC provides a monitoring module for traffic monitoring and merging trees to detect the switch between unicast and multicast.

# 5 CONCLUSIONS

This paper presented the design, implementation and evaluation of a SDN-enabled multicast solution called SDMC for autonomous, adaptive and flexible network load-aware and switch memory-efficient group communications in data center networks. SDMC, which is provided as a middleware suite, uses a combination of unicast and software-defined multicast, and dynamically switches between them while ensuring that end applications remain agnostic to the

adaptation yet provide superior performance over either only unicast or multicast cases, which ultimately improves the utility of the clouds. Experimental evaluation of our solution shows that SDMC performs up to 50% better compared to IP multicast alone in terms of network load-awareness, and switch-memory utilization efficiency, and up to 60% better for latency performance. SDMC is available in open source at https://github.com/prithviraj6116/sdmc.

Presently, SDMC makes adaptation decisions considering only one criteria at a time, i.e., utilization of network link or switch memory, and does not handle potential oscillations in the adaptation. Our future work will focus on handling them together with weights assigned to each depending on the needs of the group communication traffic. We will also support distributed SDMC controllers using blockchains, which supports novel consensus algorithms to improve controller coordination, consistency, and reliability for holding persistent data sharing and enabling selective privacy.

## ACKNOWLEDGMENTS

This work was funded partially by the Fulbright Visiting Scholars Program and NSF CNS US Ignite 1531079. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF, DGA, CNES or the Fulbright program.

## REFERENCES

[1] Dmitry Basin, Ken Birman, Idit Keidar, and Ymir Vigfusson. 2010. Sources of Instability in Data Center Multicast. In *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware (LADIS '10)*. 32–37.

[2] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. 2012. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems* 28, 5 (2012), 755 – 768.

[3] M. Chen, H. Jin, Y. Wen, and V. C. M. Leung. 2013. Enabling technologies for future data center networking: a primer. *IEEE Network* 27, 4 (2013), 8–15.

[4] OpenFlow Consortium et al. [n. d.]. OpenFlow Switch specification v1. 0. ([n. d.]).

[5] W. Cui and C. Qian. 2015. Scalable and Load-Balanced Data Center Multicast. In *2015 IEEE Global Communications Conference (GLOBECOM)*. 1–6.

[6] Z. Guo and Y. Yang. 2015. Exploring Server Redundancy in Nonblocking Multicast Data Center Networks. *IEEE Trans. Comput.* 64, 7 (2015), 1912–1926.

[7] Hugh Holbrook, Storigen Systems, and Brian Haberman. 2015. Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast. RFC 4604. (1 Oct. 2015).

[8] Linfeng Hong, Dongxu Zhang, Hongxiang Guo, Xiaobin Hong, and Jian Wu. 2013. OpenFlow-based multicast in IP-over-LOBS networks: A proof-of-concept demonstration. In *2012 17th Opto-Electronics and Communications Conference*.

[9] A. Iyer, P. Kumar, and V. Mann. 2014. Avalanche: Data center Multicast using software defined networking. In *2014 Sixth International Conference on Communication Systems and Networks (COMSNETS)*. 1–8.

[10] W. K. Jia. 2014. A Scalable Multicast Source Routing Architecture for Data Center Networks. *IEEE Journal on Selected Areas in Communications* 32, 1 (January 2014), 116–123.

[11] W. K. Jia and L. C. Wang. 2013. A Unified Unicast and Multicast Routing and Forwarding Algorithm for Software-Defined Datacenter Networks. *IEEE Journal on Selected Areas in Communications* 31, 12 (2013), 2646–2657.

[12] M. A. Khoshkholghi, M. N. Derahman, A. Abdullah, S. Subramaniam, and M. Othman. 2017. Energy-Efficient Algorithms for Dynamic Virtual Machine Consolidation in Cloud Data Centers. *IEEE Access* 5 (2017), 10709–10722.

[13] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. 2015. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* 103, 1 (2015), 14–76.

[14] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 19.

[15] Dan Li, Henggang Cui, Yan Hu, Yong Xia, and Xin Wang. 2011. Scalable data center multicast using multi-class bloom filter. In *Network Protocols (ICNP), 2011 19th IEEE International Conference on*. IEEE, 266–275.

[16] D. Li, Y. Li, J. Wu, S. Su, and J. Yu. 2012. ESM: Efficient and Scalable Data Center Multicast Routing. *IEEE/ACM Transactions on Networking* 20, 3 (June 2012), 944–955.

[17] D. Li, M. Xu, Y. Liu, X. Xie, Y. Cui, J. Wang, and G. Chen. 2014. Reliable Multicast in Data Center Networks. *IEEE Trans. Comput.* 63, 8 (Aug 2014), 2011–2024.

[18] Dan Li, Mingwei Xu, Ming-chen Zhao, Chuanxiong Guo, Yongguang Zhang, and Min-you Wu. 2011. RDCM: Reliable data center multicast. In *INFOCOM, 2011 Proceedings IEEE*. IEEE, 56–60.

[19] Xiaozhou Li and Michael J. Freedman. 2013. Scaling IP Multicast on Datacenter Topologies. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13)*. 61–72.

[20] Yona Lopes, Natalia C. Fernandes, Carlos A. M. Bastos, and Débora C. Muchaluat-Saade. 2015. SMARTFlow: A Solution for Autonomic Management and Control of Communication Networks for Smart Grids. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*.

[21] Mallik Mahalingam, T. Sridhar, Mike Bursell, Lawrence Kreeger, Chris Wright, Kenneth Duda, Puneet Agarwal, and Dinesh Dutt. 2015. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348. (14 Oct. 2015).

[22] Cesar AC Marcondes, Tiago PC Santos, Arthur P Godoy, Caio C Viel, and Cesar AC Teixeira. 2012. CastFlow: Clean-slate multicast approach using in-advance path processing in programmable networks. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*. IEEE, 000094–000101.

[23] Mike McBride. 2013. *Multicast in the Data Center Overview.* Internet-Draft draft-ietf-mboned-dc-deploy-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-mboned-dc-deploy-01 Work in Progress.

[24] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.

[25] K. Nagaraj, H. Khandelwal, C. Killian, and R. R. Kompella. 2012. Hierarchy-aware distributed overlays in data centers using DC2. In *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*. 1–10.

[26] Yukihiro Nakagawa, Kazuki Hyoudou, and Takeshi Shimizu. 2012. A Management Method of IP Multicast in Overlay Networks Using Openflow. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*.

[27] P. Pan and T. Nadeau. 2013. *Software-Defined Network (SDN) Problem Statement and Use Cases for Data Center Applications.* Technical Report 00.

[28] N.B. Roy and D. Das. 2015. Application of MultiCast Tree concept to cloud security with optimization algorithm for node search technique. In *Electrical, Electronics, Signals, Communication and Optimization (EESCO), 2015 International Conference on*. 1–6.

[29] Julius Rückert, Jeremias Blendin, and David Hausheer. 2015. Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks. *Journal of Network and Systems Management* 23, 2 (2015), 280–308.

[30] K. Sriprasadh, Saicharansrinivasan, O. Pandithurai, and A. Saravanan. 2013. A novel method to secure cloud computing through multicast key management. In *Information Communication and Embedded Systems (ICICES), 2013 International Conference on*. 305–311.

[31] Ymir Vigfusson, Hussam Abu-Libdeh, Mahesh Balakrishnan, Ken Birman, Robert Burgess, Gregory Chockler, Haoyuan Li, and Yoav Tock. 2010. Dr. Multicast: Rx for Data Center Communication Scalability. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys '10)*. 349–362.

[32] Han Wang, Ki Suh Lee, Erluo Li, Chiun Lin Lim, Ao Tang, and Hakim Weatherspoon. 2014. Timing is Everything: Accurate, Minimum Overhead, Available Bandwidth Estimation in High-speed Wired Networks. In *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*. 407–420.

[33] Minlan Yu, Alex Fabrikant, and Jennifer Rexford. 2009. BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '09)*. 313–324.

[34] Yang Yu, Qin Zhen, Li Xin, and Chen Shanzhi. 2012. OFM: A Novel Multicast Mechanism Based on OpenFlow. *Advances in Information Sciences & Service Sciences* 4, 9 (2012).

[35] S. Q. Zhang, Q. Zhang, H. Bannazadeh, and A. Leon-Garcia. 2015. Routing Algorithms for Network Function Virtualization Enabled Multicast Topology on SDN. *IEEE Transactions on Network and Service Management* 12, 4 (2015), 580–594.

[36] M. Zhao, B. Jia, M. Wu, H. Yu, and Y. Xu. 2014. Software defined network-enabled multicast for multi-party video conferencing systems. In *2014 IEEE International Conference on Communications (ICC)*. 1729–1735.

[37] Wenbing Zhao, P.M. Melliar-Smith, and L.E. Moser. 2010. Fault Tolerance Middleware for Cloud Computing. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. 67–74.