Adaptive and Flexible SDN-based Multicast for Efficient Data Center Networking

Prithviraj Patil*, Akram Hakiri† and Aniruddha Gokhale*
*ISIS, Dept of EECS, Vanderbilt University, Nashville, TN, USA.
† Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France.
Email: {prithviraj.p.patil,a.gokhale}@vanderbilt.edu, hakiri@laas.fr

***Abstract*—Data center networks (DCNs) rely heavily on the use of group communications for various tasks. For example, data center management utilities (e.g., software update/upgrade, log management, resource monitoring, scaling up or down of resources, access control etc.), collaborative applications (e.g., social media, project management tools, version control systems, etc.), multimedia applications and multi-player on-line games are few examples of tasks that require efficient group communications. Although multicast is useful for efficient group communications, IP multicast has seen very low deployment in the data center networks due to its shortcomings, such as inefficient switch-memory, inability to adapt to network load, and initial receiver latency due to the creation of multicast routing tree. Even with the advent of Software-defined Networking (SDN), which makes it easier to implement multicast in the SDN controller, this approach too faces many challenges.**

In this paper we propose a novel approach to using SDN-based multicast (SDMC) for flexible, network-load aware, and switch-memory efficient group communication, suitable particularly to data center networks. Our SDN-based multicast adaptively uses a combination of unicast and software-defined multicast by switching between the two at run-time while ensuring that the application remains agnostic to this adaptation and to find a better trade-off to retain the benefits of group communication while avoiding its disadvantages. We describe the design and implementation of SDN-enabled Multicast using SDN controllers and OpenFlow-enabled switches, and evaluate it along a number of metrics, such as initial receiver latency, network load-awareness, and switch-memory utilization efficiency.

I. Introduction

Group communication is used when one or more participants need to communicate with multiple other participants. A participant could be any entity, such as an application-level abstraction, a class object, a process, a host machine or IP address, a person, etc. Group communication is heavily used in today's data center networks (DCNs) because a number of infrastructure-level tasks that are routinely performed in data center networks inherently follow the one-to-many or many-to-many communication semantics [1]. Typical examples of group communication in DCNs include software update/upgrade, active/passive data replication, access control policies, security enforcement policies, and elastic tenant-based services like Amazon Elastic Compute Cloud (EC2).

All these tasks are routinely and heavily used in data centers and hence even the slightest inefficiency (network in this case) is detrimental to the overall performance of data centers networks. Additionally, DCNs also host client applications which rely on group communication. For example, in distributed interactive simulation, the state information should be exchanged among multiple players when they move and interact in a highly interactive virtual environments. Likewise, collaborative tools such as Exchange, Lync, and SharePoint use group communication in a cloud environment to allow users to efficiently communicate and collaborate with one another and with their team through web conferencing, documents and calendars. Moreover, group communication is present in distributed database applications such as Hadoop HDFS and Apache Spark to execute resilient distributed datasets and provide a consistent view of stored data to multiple users. These use cases drive the need for efficient group communication support in data center networks.

Traditionally, IP multicast (IPMC) has been prudently and carefully used for group communication requirements of different applications [2]. However, IPMC is very inefficient in adapting to dynamically changing network loads and the network switch memory utilization. These challenges are difficult to overcome in legacy networks where IPMC is implemented in the switches and is totally distributed. Due to these reasons, IPMC has not seen large-scale deployments in DCNs.

In recent years, Software Defined Networking (SDN) [3] has emerged as a new way for managing networks. SDN decouples the control plane of networking devices from its data plane. SDN brings several benefits to cloud computing and data centers since it provides new capabilities like network virtualization, automating resource provisioning, and creating new services on top of the provisioned network resources [4]. In particular, SDN was used in unicast communication to allow retrieving the network topology, monitoring the network conditions in case of failure, and initiating and adjusting the network connectivity such as tunneling and traffic engineering.

However, SDN does not clearly specify how multicast communication should be implemented in cloud-based environment. OpenFlow, the dominant SDN technology, floods both the Layer 2 unicast and multicast packets to all VLANs irrespective of whether the ports are protected or not. Consequently, when OpenFlow tries to setup and tear down multicast sessions, the OpenFlow multicast control traffic will hit all the OpenFlow connected interfaces. As data centers are typically composed of highly structured topologies and exercise single window control of all the networking infrastructure, their is a real need to design and implement a novel SDN-enabled

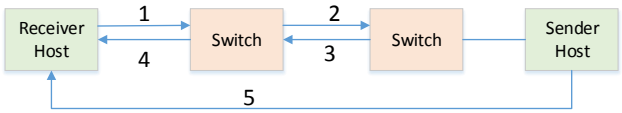
978-1-4799-7899-1/15/$31.00

approach that fulfills the requirements of data center communication. Such an approach should be able to offer better flexibility and scalability by enabling a only a group authorized use of SDN-enabled multicast communication.

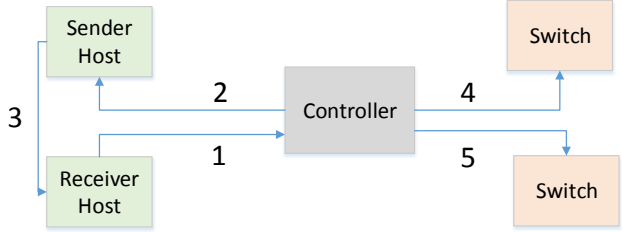
To address all these challenges yet leverage the benefits stemming from SDN-based network softwarization, we propose a novel way of using SDN-based multicast for flexible, network load-aware, switch-memory efficient group communication specifically for data center networks.

Our approach efficiently uses a combination of unicast and software defined multicast, and switches between them at runtime while ensuring that applications remain agnostic to these adaptive changes, and without any additional packet loss. It finds a better trade-off to retain the benefits of group communication while avoiding its disadvantages. Our SDN-enabled multicast protocol is more lightweight, dynamic, and adaptive to networking resources, such as link utilization and switch memory, when compared to traditional IP multicast.

To illustrate the benefits of our approach, consider traditional IP multicast as shown in Figure 1a. When a new receiver joins a multicast group (line 1), it waits till the multicast routing tree is created (lines 2, 3 and 4) before it starts receiving packets (line 5). In contrast, for SDN-enabled multicast (SDMC) as shown in Figure 1b, when a new receiver joins multicast group (line 1), it is added as a unicast destination by the SDN controller (line 2) and hence starts to receive packets immediately (line 3). At a later time, SDN-enabled multicast converts this unicast receiver to multicast (or vice versa) based on the fluctuations in network load and switch memory utilization (lines 4 and 5). This makes our approach more flexible, scalable and adaptive compared to both traditional IPMC and/or simple unicast.



(a) Traditional Multicast Topology



(b) Architecture of the SDN-enabled Multicast Approach

Fig. 1. Comparison between the SDN-enabled Multicast and Traditional IP Multicast

The key contributions made in this paper are:

- We describe the design of a network-load-adaptive and network switch-memory-adaptive multicast for data center networks.
- We present implementation details of our SDN-enabled multicast as a SDN network application running on a SDN controller and OpenFlow-enabled switches.
- We evaluate the effectiveness of our approach in data center networks along a number of metrics, such as load variations and switch-memory utilization.

A. *Organization of the paper*

The rest of the paper is organized as follows: in Section II, we discuss related research comparing it with our approach. Section III discusses the considerations that drive the SDN-enabled multicast design and its architecture. It then describes the design and implementation details of our approach illustrating how the design meets the various considerations. This section also illustrates the behavior of our SDN-enabled multicast during a variety of events like sender join, receiver join etc. In Section IV, we provide an empirical evaluation of our approach for multiple data center network topologies and settings using metrics such as latency, network load variations, switch-memory utilization etc, and compare it with unicast and traditional multicast. Finally, in Section V, we provide concluding remarks summarizing the work, and directions for future work.

II. Related Work

Several approaches have been proposed to improve the efficiency of multicast in data center networks. Authors in [5] describe an approach to improve multicast in DCNs by exploiting the diversity of multi paths available in large and highly connected DCNs. This approach creates backup overlays for every multicast routing tree and switches among them as per network load fluctuations. In [1], the authors describe a technique to improve multicast latency by compressing the tree using multi-class bloom filters. Such an approach promises to make multicast more scalable to support large number of multicast groups by removing distributed routes computation in multicast routers. Similarly, authors in [6] present an optimization multicast routing tree creation by using the Steiner tree approach. Al those approaches do not use SDN at all. In contrast to those papers, our SDN-enabled solution uses the SDN paradigm to build multicast routing approach in DCNs.

For related work on SDN-based multicast, the authors in [7] propose an OpenFlow-based multicast mechanism that shifts the multicast management to a remote centralized controller. Although this work is focused on managing multicast, its focus is more on evaluating the feasibility of multicast-based OpenFlow in general and not particularly on group communication in data centers. Authors in [8] propose the Avalanche framework, which is a SDN-enabled multicast in commodity switches used in data centers. Avalanche proposes a new multicast routing algorithm called Avalanche Routing Algorithm (AvRA) to minimize the size of the routing tree. In

contrast to Avalanche which is designed for particular Tree-like topologies in data center, our approach in SDN-enabled Multicast is a more general framework that can address different topologies. Our approach fits well for tree topologies, mesh topologies and jellyfish topologies as well.

Similarly, the authors in [9] describe a SDN-enabled efficient multicast scheme especially for IP-over-OBS networks. In [10], the authors propose another SDN-enabled multicast scheme which uses knowledge of the anticipated processing time for each route based on historical data to use an optimal routing tree. Despite their novel approaches, all of the above SDN-based approaches suffer from the same issues suffered by traditional multicast, i.e., scalability and latency, since none of these adopt to changing network load and switch-memory.

III. Design and Implementation of SDMC

This section describes the design and implementation of SDMC. Before delving into the design details, we first outline the key design considerations for SDMC.

A. Design Considerations

Before presenting the detailed design of SDMC, we first outline the key requirements of a solution that provides efficient group communication in data center networks.

1. Flexible/Dynamic: Existing multicast protocols used in group communications follow all or none semantics for multicast users [11]. This forces users the choice of using either multicast for all the senders and receivers or not using it at all. It does not allow users to use selective multicast for a few receivers or few senders while using unicast for remaining ones. Thus, a solution like SDMC should be capable of allowing applications to use multicast communication selectively as per application needs.

2. Initial Latency and lazy initialization: In existing multicast protocols like IPMC, creation/destruction of multicast senders/receivers immediately triggers the creation/update/deletion of multicast routing trees. This incurs initial latency for the receivers especially for highly dynamic subscriptionpublications and for larger sized multicast groups. Thus, SDMC should be capable of reducing the overhead and delay in the initial multicast routing tree creation without compromising receiver performance. This is achieved by delaying the creation of multicast routing tree (lazy initialization) to a later time when network and switch conditions are suitable for it.

3. Reuse of overlapping multicast routing trees: Existing multicast protocols make it impossible to reuse partial or complete multicast routing trees due to the flat nature of its multicast IDs. This approach makes scaling of multicast very difficult due to the limited switch memory resources. Consequently, SDMC should reuse the partially or completely overlapping multicast routing trees. For example, if two (or more) multicast IDs are having same (or almost the same) receivers (or switches to which receivers are connected), they should be able reuse the same multicast routing tree and hence save valuable switch memory.

4. Adapt to network load: SDMC should be able to adapt to changing network traffic by switching between unicast and multicast. For example, if one (or more) link is under high load due to unicast traffic, SDMC should be able to switch that traffic to use multicast (if that traffic is part of group communication).

5. Adapt to switch-Memory: In data centers, switches come in various sizes and shapes. Depending on the specifications of a particular switch, its switch memory, switching speed, etc, may vary. Large data centers, especially those that are in operation for long time, are likely to have heterogeneous switches. Thus, SDMC should be able to adapt to the switch-memory limitation scenario of the data center. **6. Consistent and application-agnostic behavior:** This consideration arises due to the flexible and adaptive nature of our new multicast protocol SDMC. Since, we allow the multicast protocol to dynamically adapt between total multicast, partial multicast or total unicast on network load, switch memory and application requirements, a receiver may be switched from using unicast to multicast and back to unicast during its life-cycle. Hence, while switching between these configurations, SDMC is required to provide a consistent performance to all senders and receivers such that application remains unaware of these switching.

B. SDMC Architecture

In this section, we describe the overall design of SDMC describing how it meets the solution requirements outlined above, and provide its implementation details. Figure 2 depicts the SDMC architecture that supports SDN-based multicasting. It contains SDN-enabled switches connected to form a SDN network with the control plane managed by the SDN controller (either centralized or distributed) and connected to a number of host machines (physical or virtual) with the SDN middleware (SDN Middleware) installed on them. The primary artifacts of this architecture include:

OpenFlow enabled Switches: In a typical data center network, 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. Moreover, each switch has a fixed memory in the form of a ternary content-addressable memory (TCAM), which is used to store the OpenFlow rules for forwarding data packets.

SDN controller: The data center network is managed by the SDN controller. This SDN controller can either be centralized or distributed. To avoid dealing with additional complexities arising from the use of distributed controllers, for this paper, we have used a centralized controller. We assume the SDN controller is placed on a dedicated machine with dedicated out-of-band connections to all the OpenFlow-enabled switches. For the SDN controller, we have various choices like NOX, POX, Floodlight, RYU, OpenDayLight etc.

SDMC as SDN-NetApp: The core logic of SDMC is implemented as a SDN network application (SDN-NetApp). As described previously, a network application forms the top layer

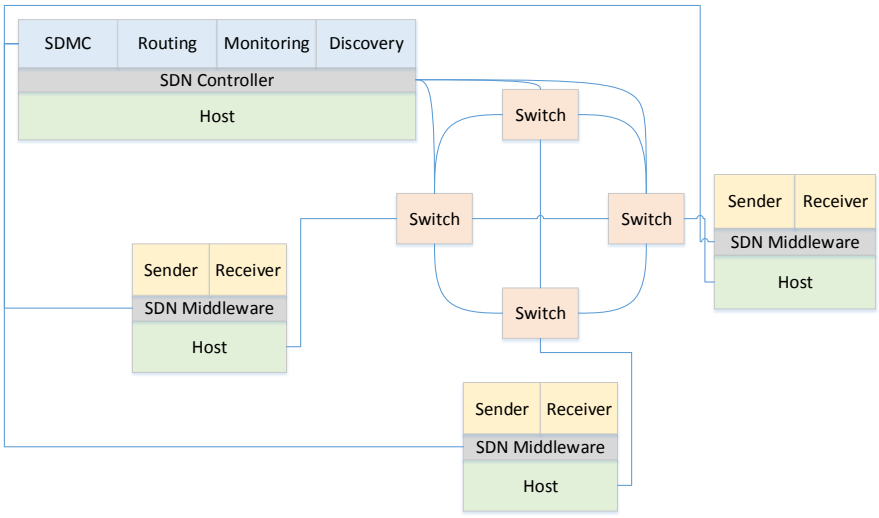


Fig. 2. SDMC Architecture

of the SDN architecture and runs in the context of the SDN controller.

Other SDN Applications: As described above a major part of SDMC is implemented as a SDN-NetApp in the control plane. Beyond this, we also need other SDN applications (apart from SDMC itself) for the execution of SDMC as described below. Some of these applications are general-purpose (like routing) but others (like host manager) are needed to be specifically built for the SDMC type applications.

- *Discovery:* This SDN-NetApp provides the service of automatic discovery of joining and leaving switches and hosts. It can assist SDMC for finding out joining or leaving recipients or senders of the multicast group.
- *Topology:* This SDN-NetApp provides the service of topology creation out of the SDN network. It can be asked to create different topologies involving the switches and hosts. It does that by activating some links and de-activating other links.
- *Monitoring:* This SDN-NetApp is used to monitor and report various network properties like link bandwidth utilization, switch memory utilization etc.
- *Routing:* The routing SDN-NetApp is used to find (unicast) routes between two hosts using algorithms like OSPF. SDMC uses the services of this SDN-NetApp to build a dynamic multicast routing tree at run-time.
- *Network Virtualizer:* If the operational environment involves a shared and multi-tenant SDN network, then we need a network virtualizer to slice the single physical SDN network into multiple SDN networks. For this work though, we focus only on non-virtualized SDN networks. Hence, we will not need a Network Virtualizer.
- *Host Manager:* This SDN-NetApp is used to keep track of hosts connected to switches and to communicate with them. This application is used by SDMC to communicate with the SDN middleware of the host machines (explained next). This application is required since we build a hybrid multicast protocol with the combination of application-level multicast (or overlay multicast) and native network-level-multicast.

Host machines with SDN Middleware: The SDMC functionality which deals with application-level multicast (or overlay multicast) is implemented with the help of a middleware (SDN Middleware) which runs on top of the host machine connected to the SDN network. This SDN-NetApp can control the host network by communicating with this middleware using the Host-Manager service application of SDN. This SDN middleware, which is installed on the host machine, is capable of performing several tasks like translating an endpoint listening on a multicast id into multiple unicast ids, switching an endpoint from unicast to multicast or vice-versa without application intervention, etc.

SDMC participants: The SDMC participants (senders and receivers) run on top of the SDN middleware. The SDN middleware hides the various complexities arising out of dynamic, lazy and flexible SDMC (described below) from these participants. Senders and receivers will send/listen only on a SDMC ID, and not deal with (and even know of) whether the underlying layers are using unicast or multicast or both.

C. Lazy Initialization

The initialization process of SDMC senders-receivers and the SDMC routing tree creation uses a lazy approach to reduce the initial latency incurred by the receivers in traditional multicast and to allow flexibility in adapting dynamically to the network load and switch memory (described later in this section). SDMC exhibits this lazy approach in its operation while switching to multicast communication from the default unicast. This lazy approach manifests for three different scenarios.

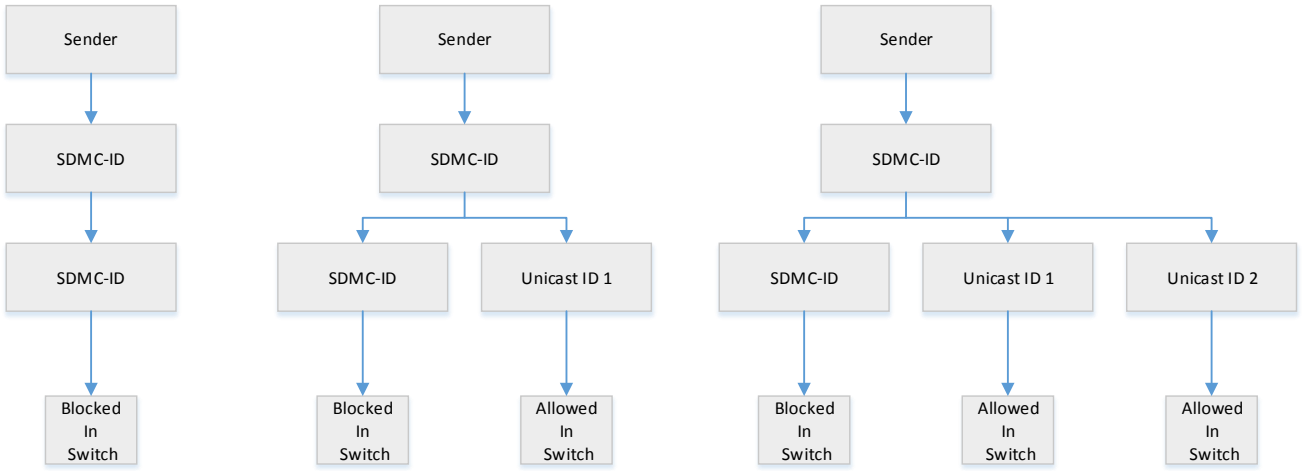


Fig. 3. Initial SDMC Sender Setup

First, when a new receiver requests to listen on SDMC-ID, it is not immediately added to the SDMC-ID as a multicast receiver but is added as a unicast destination for all the existing senders of that SDMC-ID, if any. Secondly, the SDMC multicast routing tree for a new receiver is created in the controller but is not installed (in the form of OpenFlow rules) in the switches immediately. Finally, when a receiver (or sender) leaves the SDMC-ID group, multicast tree is not updated immediately. All these above decisions (i.e., 1. when to add a receiver as a multicast destination; 2, when to install multicast routing tree in switches; and 3, when to update the multicast routing tree after a receiver leaves) are taken by the SDMC holistically based on all other SDMC sender-receiver status and on the network load and switch-memory utilization instead of triggering them immediately.

Figure 3 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 for reducing the start-up latency. This makes the receiver receive packets immediately since there is no need to create any multicast routing tree in the switches. The corresponding behavior on the receiver side apears in Figure 4.

D. Two-level SDMC-ID

To allow the reuse of multicast routing trees, the SDMC-ID space is divided into two regions: application-level (or external) and network-level (or internal). The SDMC participants will interact with only the external SDMC-IDs while the network data path will deal with internal SDMC IDs. The SDN middleware will be responsible for translating external SDMC IDs to the appropriate internal SDMC IDs (and vice-versa). The SDMC (SDN-NetApp) will direct the SDN middleware about the use of appropriate translation and when to switch between different SDMC IDs as described below. This two level SDMC-ID structure allows SDMC to use the same multicast routing tree (with the internal SDMC ID) for overlapping receivers of two different external SDMC IDs.

The decision about how to divide the address space into external and internal SDMC IPs is left to the administrator of the SDN network and can be configured at network set up time via the SDN controller configuration parameters. Figure 1 shows the communication between different entities during basic multicast and SDMC. The traditional multicast uses the same multicast ID for all the communication. SDMC however uses the external multicast for communication between application endpoints (sender and receiver) and network endpoints (switches or controller). However, for communication between the controller and switches, respective internal multicast id is used. Hence, the communication over line 1, 2 and 3 uses external (or application-level) multicast id while over line 4 and 5 uses internal (or network-level) multicast id.

E. Network Link and Switch-memory monitoring

SDMC periodically keeps track of network links and their utilization with the help of the monitoring network application. It then populates the link utilization information against the SDMC IDs which are using that link for the unicast for a receiver as shown in Table I. Network link monitoring and switch memory monitoring is also shown pictorially in Figure 5.

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 are on each of the switches. Each switch comes with the 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.

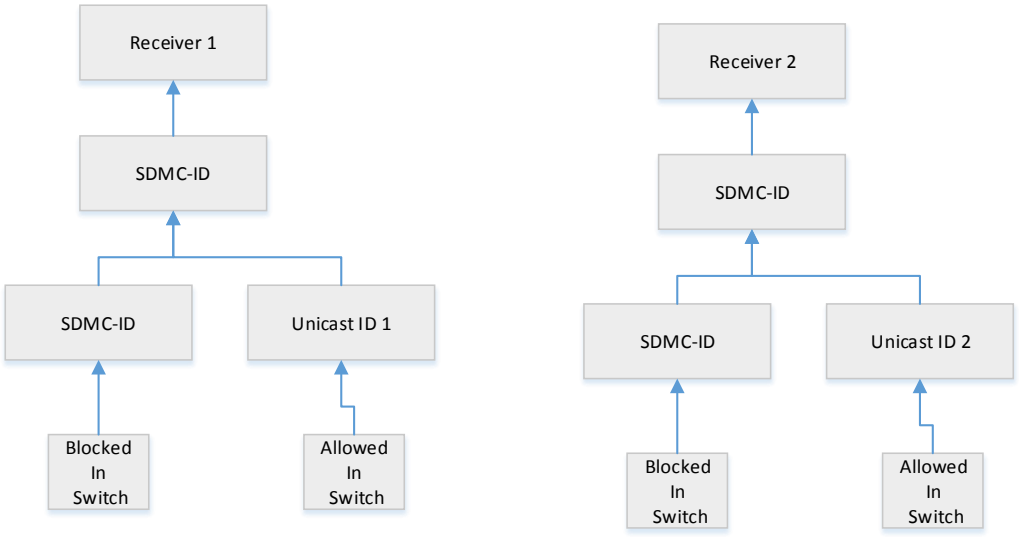


Fig. 4. Initial SDMC Receiver Setup

TABLE I
NETWORK LINK MONITORING

Link	Switch No.	Switch No.	Utilization	SDMC-ID	Receiver (Unicast)
L1	Switch-03	Switch-05	17%	SDMC-ID-01 SDMC-ID-87	Rec-33 Recv-54
L2	Switch-20	Switch-11	32%	SDMC-ID-11 SDMC-ID-42 SDMC-ID-45	Rec-12 Recv-23 Recv-54
L3	Switch-31	Switch-21	31%	SDMC-ID-03 SDMC-ID-85	Rec-13 Recv-53
L4	Switch-13	Switch-03	69%	SDMC-ID-21 SDMC-ID-52 SDMC-ID-74 SDMC-ID-24	Rec-21 Recv-67 Recv-42 Recv-45
L5	Switch-3	Switch-9	81%	SDMC-ID-1	Rec-1
L6	Switch-15	Switch-21	70%	SDMC-ID-11 SDMC-ID-32	Rec-11 Recv-28

For this work, we deal with OpenFlow version 1.0 rules for all switches. However, as newer versions of the OpenFlow standard are proposed, the controller should keep track of different versions of the OpenFlow rule space for different switches. This table (Table II) also keeps track of multicast receivers and associated SDMC ids for each switch. Network link monitoring and switch memory monitoring has been shown pictorially in Figure 5.

F. SDMC Operation

We now describe the runtime operation of SDMC and explain the sequence of activities executed by SDMC in response to various events like sender join, receiver join, sender leave, receive leave, etc.

Sender Join: When a participant (sender) wants to send data on an application-level SDMC ID, M^e,

- The sender sends a request to its SDN middleware. The SDN middleware sends the request to the SDN NetApp.
- The SDN NetApp assigns an appropriate internal SDMC ID M^i to correspond to the requested application-level SDMC ID, M^e.
- It also installs an OpenFlow rule in the edge switch of the sender to block all the traffic with the destination id M^i.
- Thereafter, the SDN middleware on the sender node installs a translation rule for $M^e <-> M^i$ in the host 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

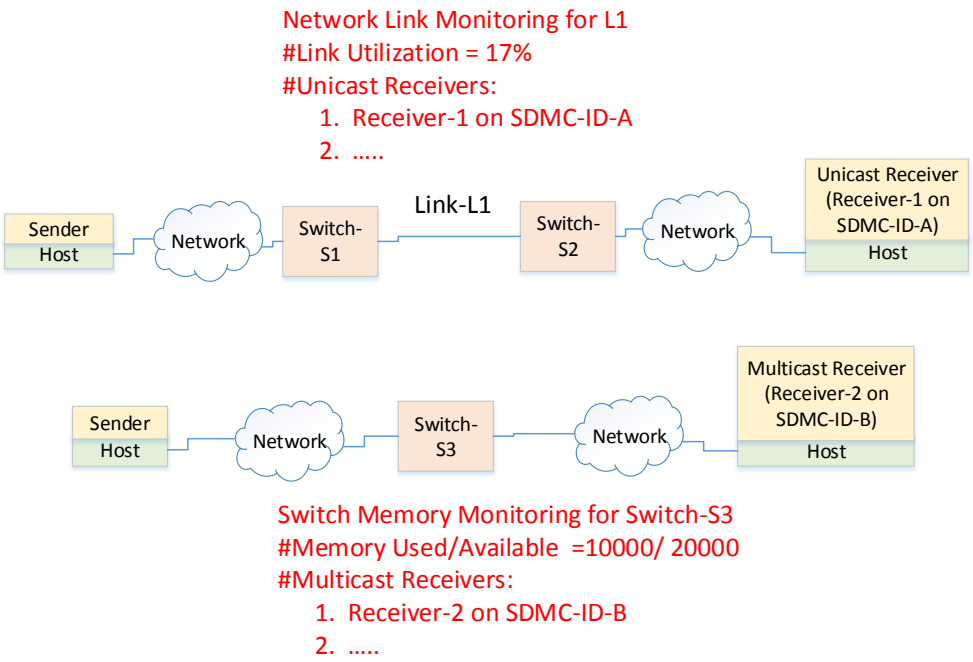


Fig. 5. Network Link and Switch Memory Monitoring

TABLE II
NETWORK SWITCH MEMORY MONITORING

Switch No.	Available Memory	Used Memory	SDMC-ID	Receiver (Multicast)
Switch-00	10000	9132	SDMC-ID-01 SDMC-ID-19	Rec-65 Recv-65
Switch-10	20000	1313	SDMC-ID-41 SDMC-ID-64 SDMC-ID-08	Rec-43 Recv-84 Recv-63
Switch-32	20000	14434	SDMC-ID-13 SDMC-ID-43	Rec-15 Recv-14
Switch-12	30000	15151	SDMC-ID-01 SDMC-ID-18 SDMC-ID-34 SDMC-ID-75	Rec-64 Recv-24 Recv-47 Recv-51
Switch-32	10000	3234	SDMC-ID-14 SDMC-ID-65	Rec-76 Recv-54

external SDMC ID M_e.

- Additionally, it installs the following translation rule $M^e -> (M^i, U_1, U_2)$, where U_1 and U_2 are the unicast destinations of the receivers of the application-level multicast id M^e. This allows sender to start sending packets using unicast.

As seen from the above sequence of events, the joining of a sender does not trigger the creation or update of the multicast routing tree. This is possible because initially every sender is made to use unicast only. Subsequently, depending on the conditions of the network and the switch, the sender is asked to switch between multicast and unicast. This is part of the lazy initialization process of SDMC.

Receiver Join: When a participant (receiver) wants to listen on an application-level SDMC ID, M^e, the following steps are performed:

- The receiver sends a request to its own SDN middleware which forwards the request to the SDMC NetApp.
- The SDMC NetApp retrieves the respective internal network-level SDMC-ID ,M^i, if available otherwise create a new one, and sends it to the receiver SDN-Middleware.
- SDMC then installs an OpenFlow rule in the edge switch of the receiver to block all the traffic with the destination

id M^i.

- The SDN middleware on the receiver installs the two translation rules $M^e <->M^i$ and $M^e <-> U^1$, where the U^1 is the unicast id of this receiver.
- The SDN middleware also gives preference to the $M^e <-> U^1$ rule over $M^e <-> M^i$ so that the first rule gets matched. This makes the receiver to listen on the unicast ID instead of multicast ID. This is part of the lazy initialization of SDMC receivers.
- Meanwhile, the SDMC NetApp searches for all the senders of M^i and adds the unicast destination of U^1 in their SDN middleware translation rules.
- It then requests the unicast routing paths for this receiver to every sender of M^i from the Routing NetApp. Based on these routing paths, it then updates Table I and Table II by adding a sender-receiver pair against each appropriate network link and switch.

Similar to the joining of a sender, joining of a receiver also does not trigger the creation or update of the multicast routing tree. This is possible because initially every sender is made to use unicast only. Later on, depending on the conditions of the network and the switch, the receivers are asked to switch between multicast and unicast. This is part of the lazy initialization process of SDMC.

Adapting to network load: As discussed above, whenever one or more links in the SDN network crosses a predefined threshold load, the SDMC NetApp is notified by the Monitoring NetApp. This is done because the SDMC NetApp registers a listener event on the Monitoring NetApp to notify it in the case of the link crossing a particular threshold. The threshold is specified as a percentage of bandwidth utilization for a specific amount of time, e.g., 90% bandwidth utilization for a link for more than 30 consecutive seconds. The SDMC NetApp logic periodically (e.g., once in 60 seconds etc.) attempts to reduce the network load by switching relevant unicast receivers to multicast receivers. It searches for the unicast receivers in Table I against the overloaded link. The SDMC NetApp then switches these receivers from unicast to multicast either one by one or simultaneously. In the next subsection, we describe the process to switch from unicast to multicast without losing any packet or impacting the receiver's performance.

Switching from unicast to multicast: This event is triggered by the fluctuations in network link utilization as discussed above. Figure 6 shows the sequence of events that take place when a receiver is switched from unicast to multicast by the SDMC. Thus, when SDMC wants to add receiver r1 to the multicast for a particular sender s1, it will execute following steps.

- SDMC will instruct the receiver r1 to listen on its unicast id U_1 along with multicast id M^i. At this point, receiver will not receive anything on its multicast id from sender s1, since the core switches and edge switches of sender/receiver are blocking the packets for multicast destination M^i.
- Then, it updates the multicast routing tree by adding appropriate OpenFlow rules to reach receiver r1 from the existing multicast routing tree of M^i in the core switches. At this point too, the receiver r1 will only receive packets on unicast id since the edge switches of sender/receiver are blocking the packets for multicast destination M^i.
- Update multicast routing tree on the edge switch of sender s1 by adding/enabling the OpenFlow rule for multicast ID M^i for receiver r1. This last step is not required if one or more receiver, apart from r1, of sender s1 are using multicast and are reached by the same switch from the sender s1. This step is always required initially when there are no existing multicast receivers for the sender s1). At this point too, the receiver r1 will only receive packets on unicast id since the edge switches of receiver are blocking the packets for multicast destination M^i. However, now there will be duplicate packets sent by the sender s1 in the network but without any receiver.
- SDMC now executes the following two tasks atomically: (1) add an OpenFlow rule which blocks packets to the unicast destination of the receiver r1 on the edge switch of the sender s1; (2) update the multicast routing tree on the edge switch of the receiver r1 by adding/enabling the OpenFlow rule for multicast ID M^i to reach the receiver r1.
- The atomicity of the above two steps guarantees that no packet will be lost and no packet will be received more than once in the migration from unicast to multicast. This, however, does not prevent the senders from sending packets on both unicast and multicast IDs during the time step 3 is started till the last time step in this sequence is finished.
- Later, when the receiver starts to receive packets on its multicast id, it will stop listening on its unicast id.
- The receiver then notifies SDMC that it is now listening on only multicast ID, SDMC will then update a translation rule in the SDN middleware of s1 by removing the unicast address of receiver r1 (u1) from its mapping, i.e., $M^e -> (M^i, U_1, U_2)$ becomes $M^e -> M^i, U_2$. At this point, the sender will stop sending packets to unicast destination of receiver r1.

However, for efficiency, SDMC should not switch single receiver-sender pairs from unicast to multicast but perform bulk switching periodically.

Adapting to switch-memory utilization: Similar to network load monitoring, the Monitoring NetApp monitors the network switch memory utilization too. Hence, whenever a memory utilization of a SDN switch in the SDN network crosses the threshold limit, the SDMC NetApp is notified by the Monitoring NetApp. This is done after SDMC-NetApp registers a listener event on the Monitoring NetApp to notify it in the case of switch memory utilization crosses a particular threshold. This threshold is specified either in the percentage of memory utilization of a switch or number of OpenFlow rules installed on the switch for the SDMC. In this work, we take the latter approach of counting the switch-memory in the

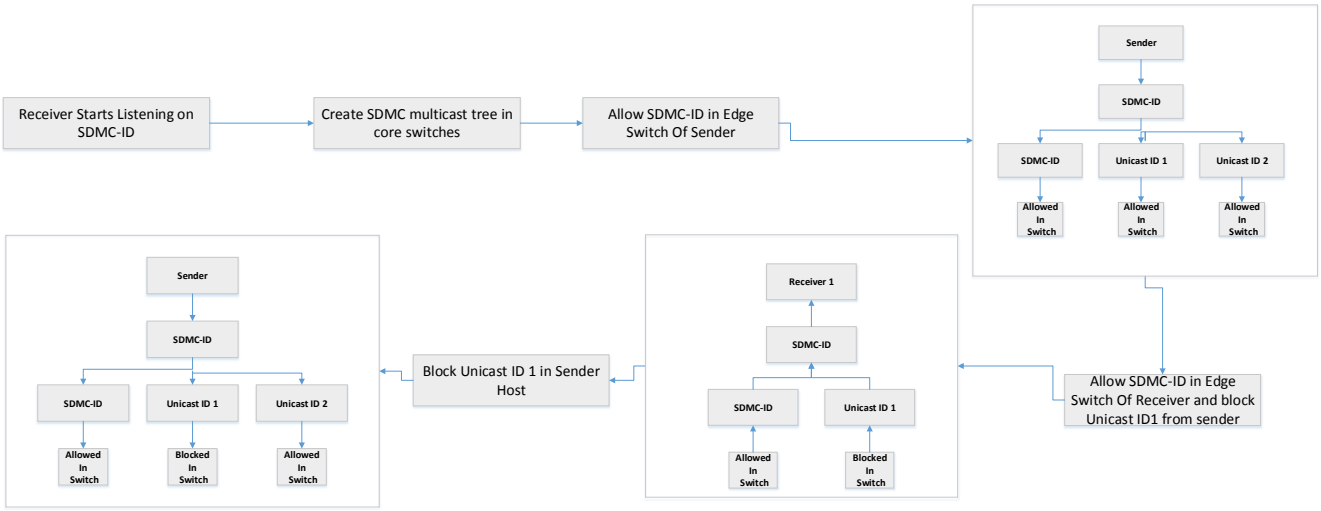


Fig. 6. Adapting to Network Load

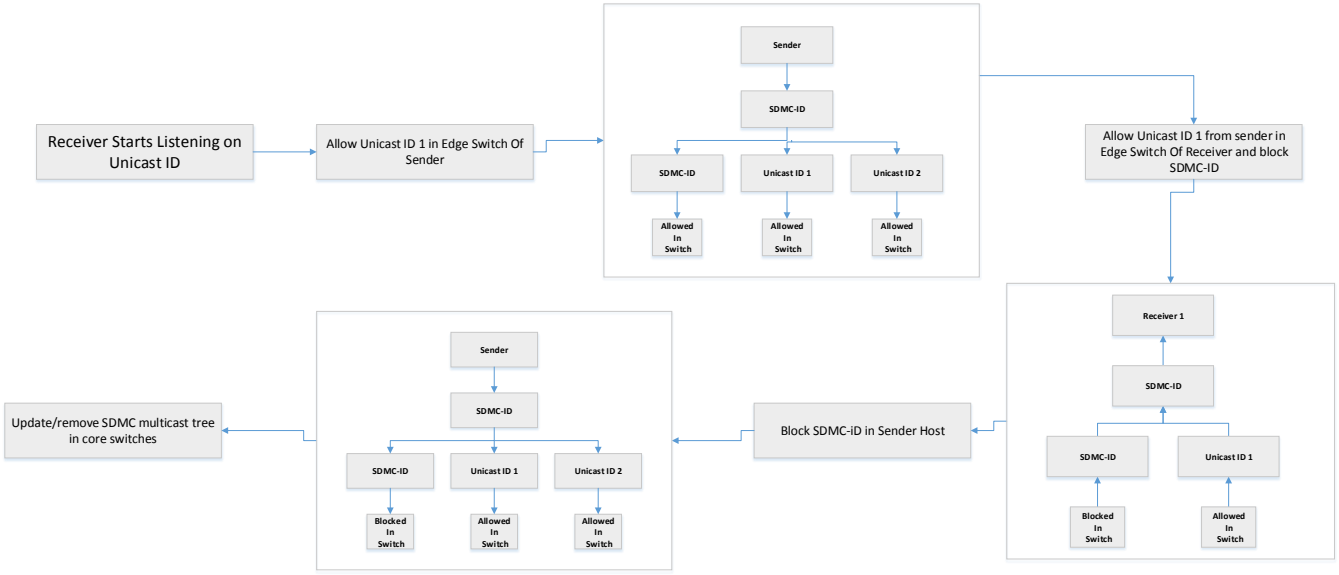


Fig. 7. Adapting to Switch Memory Utilization

form of number of OpenFlow rules. The reasoning behind this approach is that it specifically measures the switch memory utilized for the SDMC and not other SDN network applications like unicast routing or load balancing, etc. The SDMC NetApp logic periodically (e.g. once in 60 seconds) tries to decrease the memory utilization of the switch by switching relevant multicast receivers to use unicast communication. To do that, it searches for the multicast receivers in Table II against the overloaded switch. SDMC-NetApp then switches these receivers from multicast to unicast either one by one or simultaneously. In the next subsection, we describe the process to switch from multicast to unicast without losing any packet or impacting the receiver's performance.

Switching from multicast to unicast: This event is triggered by the changes in switch memory utilization as discussed above. Figure 7 shows the sequence of events that happen during receiver is switched from multicast to unicast by the SDMC. So, when SDMC wants to remove the receiver r1 from the multicast for a particular sender s1, the following steps will be followed:

- SDMC will instruct the receiver r1 to listen on its unicast id U_1 along with multicast id M^i. At this point, receiver will not receive anything on its unicast id from sender s1, since core switches and edge switches of sender/receiver are blocking the packets for multicast destination M^i and sender is not sending any packet on the unicast id of the receiver.
- SDMC will then update a translation rule in the SDN-

Middleware of sender s1 by adding the unicast address of receiver r1 (u1) in its mapping. i.e. $M^e -> (M^i, U_2)$. becomes $M^e -> M^i, U_1, U_2$. At this point, sender will start to send on both the unicast and multicast. But, at this point too, receiver will not receive anything on its unicast id from sender s1, since core switches and edge switches of sender/receiver are blocking the packets for multicast destination M^i.

- Now, SDMC updates the multicast routing tree for multicast id M^i, on the core switches by disabling/removing OF rules which to reach receiver r1 for multicast id. At this point too, receiver will not receive anything on its unicast id from sender s1, since edge switches of sender/receiver are blocking the packets for multicast destination M^i.
- SDMC now executes following two tasks atomically: (1) remove the OF rule on the edge switch of sender s1 which blocks packets to the unicast destination of receiver r1; and (2) update multicast routing tree on the edge switch of sender s1 by disabling/removing OF rule to reach receiver r1.
- The atomicity of the above step guarantees that no packet will be lost and no packet will be received more than once in the migration from multicast to unicast. This, however, does not prevent senders from sending packets on both unicast and multicast IDs during the time step 3 is started till the time previous step in this sequence is finished.
- Later, when receiver starts to receive packets on its unicast id, it will stop listening on its multicast id.
- The receiver then notifies SDMC that it is now listening on only unicast ID, SDMC will then update the multicast routing tree on the edge switch of the receiver r1 such that packets for destination M^i will not reach receiver r1.

However, for efficiency SDMC should not switch single receivers-sender pair from multicast to unicast but perform bulk switching periodically.

Receiver Leave: When a receiver wants to leave the multicast group, its SDN middleware notifies the SDMC with "Receiver_Leave" notification. SDMC first checks if the receiver is using unicast or multicast. If the receiver is on unicast mode, then SDMC only needs to remove it from the translation rule of its senders. However, if receiver is using multicast, then SDMC needs to update the routing tree. However, SDMC does not need to do this action immediately. But, it only removes that OpenFlow rule from the edge-switch of the receiver which forwards dest=M^i packets to this receivers. The remaining multicast routing tree is cleaned up during the next periodical switch-memory monitoring event. So, if the same receiver (or another receiver connected to the same switch or another receiver which can be reached via the same switch) joins again (before cleanup), SDMC takes lesser time in updating the multicast routing tree for it.

Sender Leave: When a sender receiver wants to leave the multicast group, its SDN middleware notifies the SDMC with "Sender_Leave" notification. SDMC then adds the OpenFlow rule in the edge switch of the sender to block the traffic from the sender s1. At this point, there will be no traffic from the sender in the network. Then, SDMC asks the SDN middleware of the sender to delete the sender. SDMC defers the removal of multicast routing tree of the M^i used by sender to later time.

IV. Experimental Evaluation

In this section we evaluate the performance of SDMC and compare it against that of unicast and traditional multicast using IPMC for various performance metrics like average latency for receivers, switch memory utilization efficiency, network load-awareness and packet loss. As described in the design considerations, SDMC is hybrid and flexible in the sense that it tries to combine the benefits of both unicast and multicast at run-time. Hence, we compare SDMC with unicast and multicast (IPMC) and illustrate how overall SDMC performs better than both the approaches.

A. Average Initialization Setup Time

a) Rationale: In group-base communication, multicast service mode needs an initialization setup time to enable receivers join the group and perform their membership. Therefore, we measure the performance of our approach against the generic IP multicast service in term of initialization setup time for the receivers. As the size of the multicast group increases, multicast protocols should be able to refreshes group membership periodically. The receiver-initiated group membership allows better management of the leaf nodes. As an increasing number of senders/receivers can join and/or leave the group membership, the size the multicast group becomes complex and the initialization setup time may increase. In traditional IP multicast, the update time of the routing tree may increase to allow receivers detecting transmissions from a specific sender. In our SDN-enabled multicast, the receivers can use unicast service in order to reduce the setup latency, then they can switch to multicast.

b) Analysis: Figure 8 shows the receiver-initiated setup time in terms of group membership size. The Figure compares our SDN-enabled the traditional multicast service and the unicast service as well. I also shows the initialization setup time against the increasing number of senders. i.e, this number is increased from one to 20. We also performed the measurements for three different network topologies in data center networks, i.e., tree, mesh, and jellyfish topologies as well. A close inspection to Figure 8 shows that our SDN-enabled multicast performs better latency in comparison to traditional IP multicast. In particular, in all cases the average initialization latency perform up to 50% better results then IP multicast. We claim that our approach allows creating the routing tree in the centralized controller before being populated to the receivers. As the controller have a global knowledge of all the joining/leaving nodes, it programs the switches by injecting OpenFlow rules, which allow receivers to join the that tree. This approach is different from traditional IP multicast, which requires a receiver *NACK* process initiation to detect a specific sender. Accordingly, the evaluation of the initialization latency

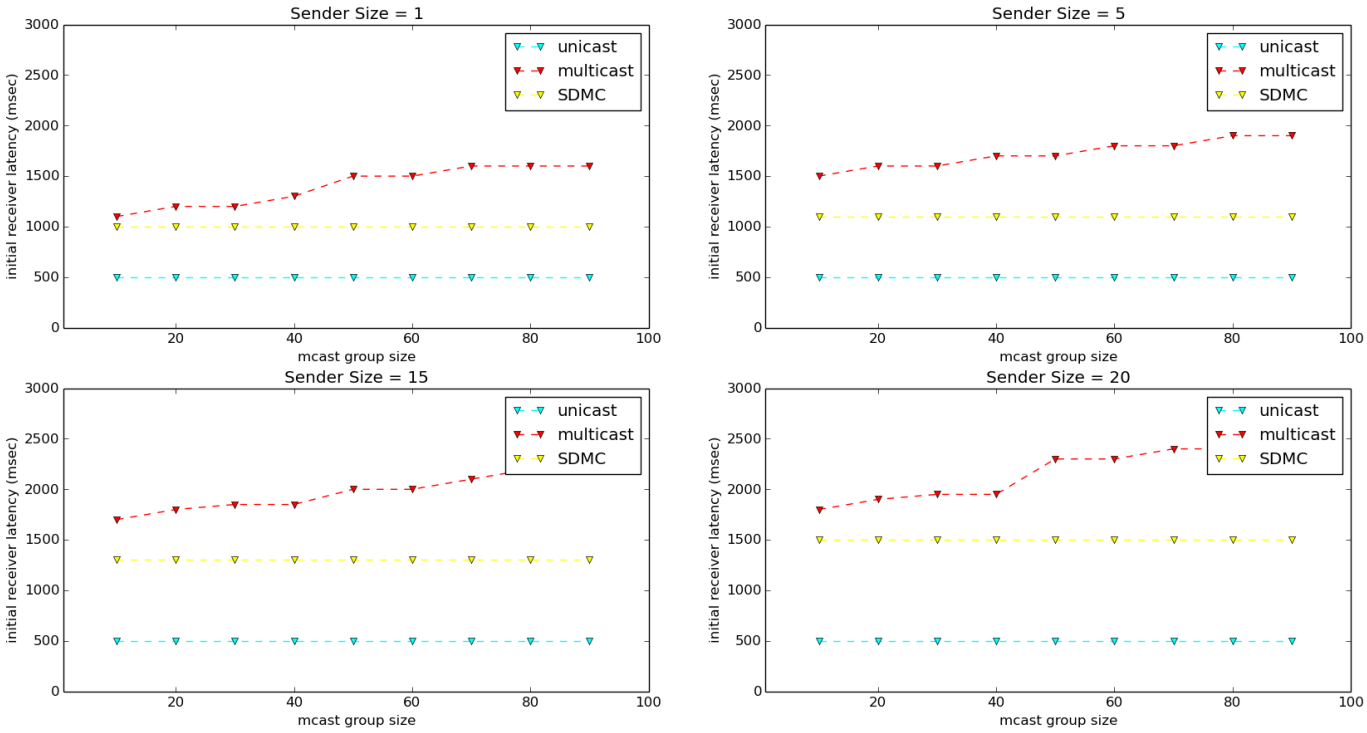


Fig. 8. Receiver Initialization Setup Time

confirms our claims on ensuring better latency performance compared to traditional IP multicast.

Additionally, we measured the initialization setup time for different network topologies in DCNs. Figure 9 the setup delay for the most common data center network topologies, i.e., jellyfish, tree, flat-tree, and random topologies. A close inspection of this shows that our SDN-enabled multicast performs better setup latency compared to traditional multicast in the four DCN topologies. Again, the evaluation results confirms the efficiency of our SDN-enabled multicast compared to traditional IP multicast.

B. Adaptiveness to the Network Load

a) Rationale: In traditional unicast communication, when a sender needs to transmit data to multiple receivers, it has to duplicate the same packets for every receiver. Traditional IP multicast solved this issue by enabling sending only one copy of those packets towards all the receivers, as a way to improve the network performance and reduce the load. In order to evaluate the adaptiveness of our SDN-enabled multicast to this situation, we performed several measurements in four different data center network topologies, i.e., jellyfish, mesh, tree, and random topologies.

b) Analysis: Figure 10 illustrates the latency required by receivers to receive a packet from the sender. It also compares this time delay against the traditional IP multicast as well as the unicast service. A close inspection of Figure 10 shows that the average end-to-end latency experienced by our SDN-enabled multicast performs better results compared to both traditional multicast and unicast services. Indeed, our SDN-enabled solution adapts itself to the network load. This because our approach aimed at making the tradeoff between unicast and multicast, so that when a the network load becomes critical and high data loss could be expected it switches some of the receivers to unicast communication, while keeping their packets reachable. Hence, our SDN-enabled multicast can successfully adapt the transmission rate using the automatic adaptiveness approach managed by the SDN controller, and therefore adapt itself to the network load.

C. Adaptiveness to Switch-Memory Utilization

a) Rationale: There are three critical areas that are related to the performance of the communication in data center networks: the CPU usage, the memory utilization and the switching capacity. CPU load is usually impacted by the control functions to perform connection creation, tear down, layer 2 functions such spanning tree, and more. As the control functions are now removed from switches to the external SDN controller, which run on high performance general purpose computer, CPU load is not a concern in our case. However, the forwarding operations come with the cost of increasing the switch-memory utilization which could be critical issue in data center networks. In particular, memories and queuing buffers hold packets and connection state information of the traffic transiting across the switch. Therefore, there is need to maintain the memory utilization decreasing as much as possible to avoid buffer overflow. Furthermore, the switch capacity can help in estimating how much transit traffic data as

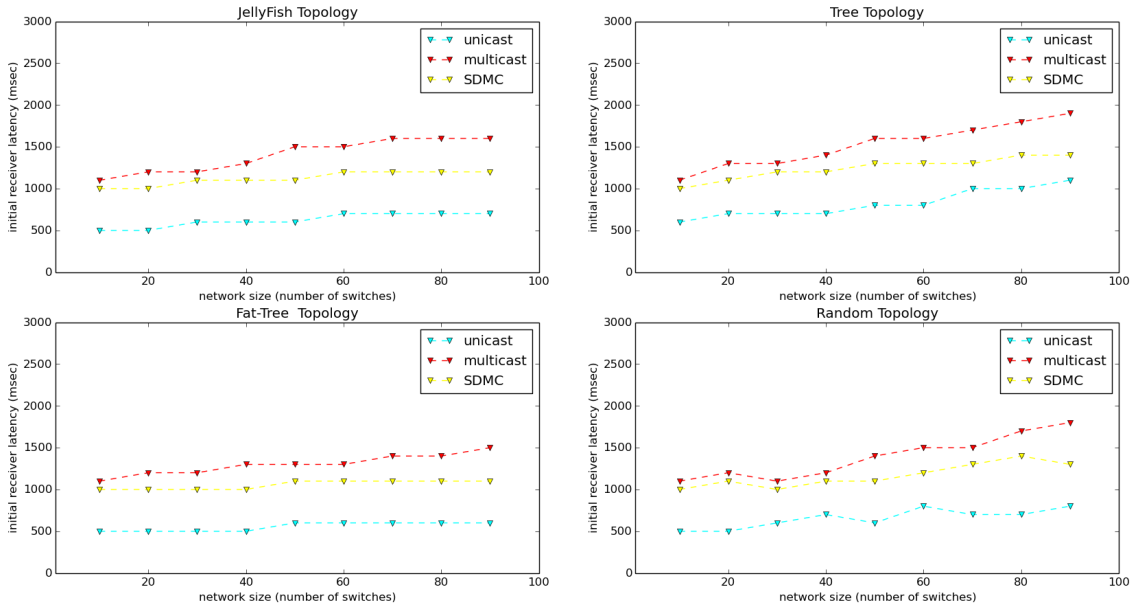


Fig. 9. Receiver Latency for different network topology and size

well as control messages could be forwarded among multiple ports concurrently. Hence, the switch capacity is a important factor to estimate the network load in terms of the overall bandwidth a switch is able to support.

Accordingly, any routing algorithm design should take into account the performance requirements of the memory utilization as well as the network load to avoid drastic problems. We therefore evaluate our approach in designing the SDN-enabled multicast in terms of switch-memory utilization and compare it against traditional IP multicast and unicast service as well.

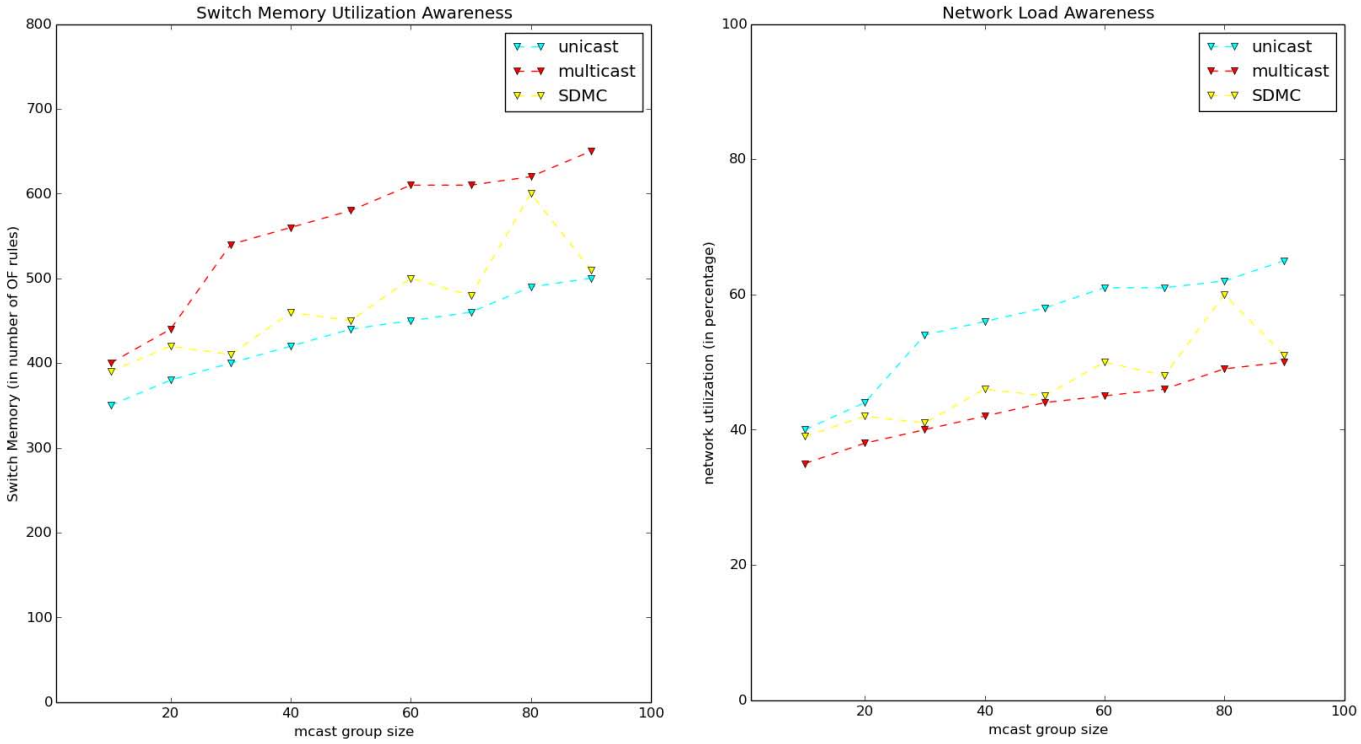
b) Analysis: To evaluate the effectiveness of our approach in avoiding the buffers overflow, Figure 10a depicts the switch-memory utilization in cases of IP multicast and unicast services and our SDN-enabled multicast. Indeed, our approach performs better memory utilization in terms of number of OpenFlow rules that can be hold in the buffers compared to traditional IP multicast. This mainly due 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 ask for new . rules the controller could inject to help the switch forwarding packet to their destination. 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 succeeded better memory utilization compared to IP multicast.

Furthermore, in order to evaluate the performance of the SDN-enabled multicast against the IP multicast in terms of network load, Figure 10b shows the bandwidth utilization in both cases. The average performance of both the IP multicast and SDN-enabled multicast is close to 40% of the bandwidth. Both approaches succeeded in avoiding network overhead.

D. Evaluating the Packet Loss

a) Rationale: In order to provide in depth inspection of the average relative error of the throughput described in Section IV-C, we evaluate the packet loss and study 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 too short-lived burst 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 of our SDN-enabled multicast approach.

b) Analysis: To evaluate the link utilization, Figure 11 illustrates the packet loss for the IP multicast and unicast services and compare them to our approach in SDN-enabled multicast. This Figure shows that our approach present an average of 4% of loss, which is little better than the traditional multicast that experiences and average of 6% of packet loss. Indeed, the current version of the OpenFlow does not include any QoS service differentiation, such as the DSCP fields in DiffServ approach, to enable per-class packet classification, scheduling and forwarding. Thus, traffic prioritization is not applied to protect packets against any computing flows. Nevertheless, the packet error in our experiments does not drastically degrade the performance of the communication because 96% of the packets are sent send their corresponding destination. Those results show that our SDN-enabled multicast can successfully support the group communication in data center



(a) Switch-Memory Utilization (b) Network Load

Fig. 10. Adaptiveness of SDN-enabled Multicast to the Network Load and the Switch-Memory utilization

networks.

V. Conclusions

Data center networks (DCNs) rely heavily on the use of group communication for various tasks. Although the traditional multicast protocol has been used in several cloud environments, however, it became clear that it suffers from its lack of support for efficient and scalable group communication. The introduction of intelligence in the network through the software-defined network (SDN) paradigm allows creating new flexible group communication solutions that solve the dynamic group membership problem in data center environments. This paper presents the design, implementation and evaluation of a SDN-enabled multicast solution for flexible, network load-aware, switch memory-efficient group communications for DCNs. Our approach uses a combination of unicast and software-defined multicast, and dynamically adapts by switches between them while remaining agnostic to the applications yet providing superior performance over individual cases. Experimental evaluation of the proposed solution shows that our approach performs better than traditional multicast in terms of initial receiver latency, network load-awareness, and switch-memory utilization efficiency.

The following lessons were learned conducting this research, which highlights the current limitations of the work while simultaneously highlighting opportunities for further research in this area.

- **Reliable SDN-enabled Multicast for DCNs:** Our SDN-enabled multicast holds promise to reduce packet loss compared to traditional IP multicast. It also showed better performance in terms of initial receiver latency and network load-awareness as well. With increasing number of routers that will support our SDN-enabled multicast in the DCNs, packets will be transmitted across multiple available paths to support an acceptable level of fidelity. We argue that supporting multi-path data dissemination in DCNs need more sophisticated reliable multicast support to alternate paths to minimize both tree and recovery costs [12]. Such an approach could use the fast-failover group feature of the current OpenFlow for better resource management and efficient group communication [13]. We believe that extending our current work on SDN-enabled group communication by supporting reliable multicast communication will allow efficient resource utilization, will help in handling link failure, and provide more efficient routes.
- **Secure SDN-enabled Multicast Key Management:** Although our proposed SDN-enabled Multicast is a promising approach to improve network performance required for simultaneous group communication, supporting group communication in DCNs poses new demands on security. In particular, because of the programmable aspect of the SDN-enabled multicast, it becomes exposed to an increasing number of DDoS, malware attacks, spam and

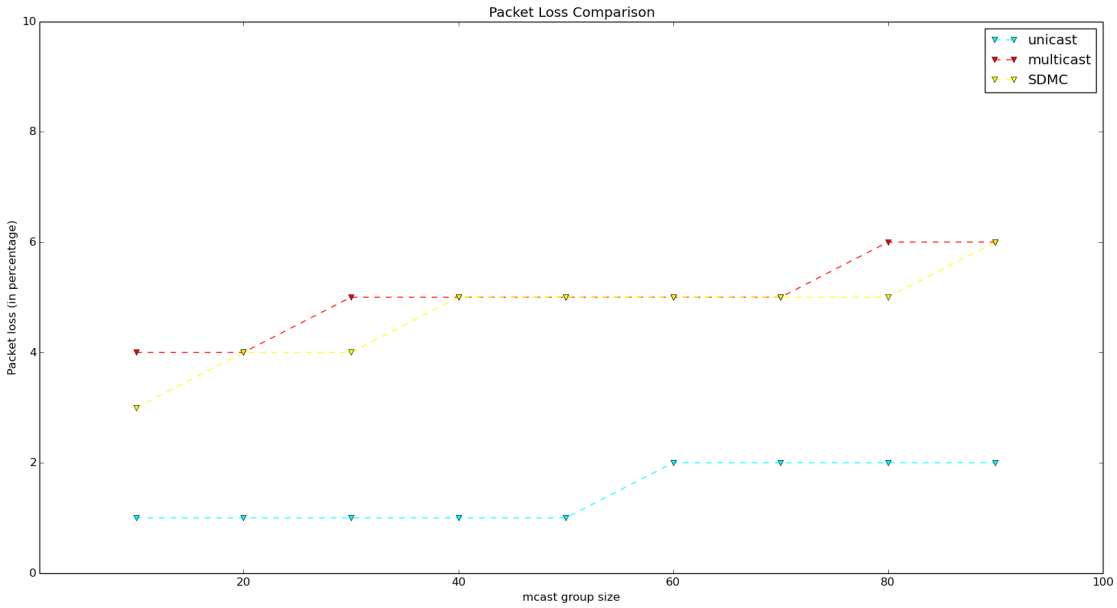


Fig. 11. The Packet Loss of the SDN-enabled multicast

phishing activities. Those attacks will be propagated simultaneously to the multicast groups. As security policies are minimally specified in SDN, those policies require downtime to orchestrate a topology and reconfigure the overall network to enforce multiple security services. Accordingly, we believe that more enhanced services should be added to enforce the trustworthiness in SDN-enabled multicast [14]. Such a security enforcement service should involve more sophisticated encryption and authentication mechanisms based on key/ID management [15] to prevent hackers and recover packets from failure. These form additional dimensions of future work.

REFERENCES

[1] D. Li, H. Cui, Y. Hu, Y. Xia, and X. Wang, "Scalable data center multicast using multi-class bloom filter," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*. IEEE, 2011, pp. 266–275.

[2] M. McBride and H. Lui, "Multicast in the data center overview," Internet Engineering Task Force, Jun 2012.

[3] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[4] P. Pan and T. Nadeau, "Software-defined network (sdn) problem statement and use cases for data center applications," Tech. Rep. 00, 2013.

[5] D. Li, M. Xu, M.-c. Zhao, C. Guo, Y. Zhang, and M.-y. Wu, "Rdcm: Reliable data center multicast," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 56–60.

[6] M. Imase and B. M. Waxman, "Dynamic steiner tree problem," *SIAM Journal on Discrete Mathematics*, vol. 4, no. 3, pp. 369–384, 1991.

[7] Y. Yu, Q. Zhen, L. Xin, and C. Shanzhi, "Ofm: A novel multicast mechanism based on openflow." *Advances in Information Sciences & Service Sciences*, vol. 4, no. 9, 2012.

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

[9] L. Hong, D. Zhang, H. Guo, X. Hong, and J. Wu, "Openflow-based multicast in ip-over-lobs networks: A proof-of-concept demonstration," in *2012 17th Opto-Electronics and Communications Conference*, 2013.

[10] C. A. Marcondes, T. P. Santos, A. P. Godoy, C. C. Viel, and C. A. Teixeira, "Castflow: Clean-slate multicast approach using in-advance path processing in programmable networks," in *Computers and Communications (ISCC), 2012 IEEE Symposium on*. IEEE, 2012, pp. 000 094–000 101.

[11] Y. Nakagawa, K. Hyoudou, and T. Shimizu, "A management method of ip multicast in overlay networks using openflow," in *Proceedings of the first workshop on Hot topics in software defined networks*, ser. HotSDN '12, 2012, pp. 91–96.

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

[13] T. Pfeiffenberger, J. L. Du, P. Bittencourt Arruda, and A. Anzaloni, "Reliable and flexible communications for power systems: Fault-tolerant multicast with sdn/openflow," in *New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on*, July 2015, pp. 1–6.

[14] N. Roy and D. Das, "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*, Jan 2015, pp. 1–6.

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