Avalanche: Data Center Multicast using Software Defined Networking

Aakash Iyer, Praveen Kumar, Vijay Mann
IBM Research - India
Email: {aakiyer1, praveek7, vijamann}@in.ibm.com

***Abstract*—Group communication is extensively used in modern data centers. Multicast lends itself naturally to these communication patterns. Traditionally, concerns around reliability, scalability and security have resulted in poor adoption of IP multicast in the Internet. However, data center networks with their structured topologies and tighter control present an opportunity to address these concerns. Software defined networking (SDN) architectures, such as OpenFlow, further provide the opportunity to not merely adopt but also innovate multicast in data centers.**

In this paper, we present Avalanche - An SDN based system that enables multicast in commodity switches used in data centers. As part of Avalanche, we develop a new multicast routing algorithm called Avalanche Routing Algorithm (AvRA). AvRA attempts to minimize the size of the routing tree it creates for any given multicast group. In typical data center topologies like Tree and FatTree, AvRA reduces to an optimal routing algorithm that becomes a solution to the Steiner Tree problem. Avalanche leverages SDN to take advantage of the rich path diversity commonly available in data centers networks, and thereby achieves highly efficient bandwidth utilization. We implement Avalanche as an OpenFlow controller module. Our emulation of Avalanche with Mininet Hi-Fi shows that it improves application data rate by up to 12%, and lowers packet loss by 51%, on an average, compared to IP Multicast. We also build a simulator to evaluate Avalanche at scale. For the PortLand FatTree topology, Avalanche results in at least a 35% reduction, compared to IP Multicast, in the number of links that are less than 5% utilized, once the number of multicast groups exceeds 1000. Lastly, our results confirm that AvRA results in smaller trees compared to traditional IP Multicast routing.

I. INTRODUCTION

Group communication is extensively used in modern data centers. Some examples include Hadoop [1] which uses data replication for higher availability, clustered application servers [2] which require state synchronization, and cloud environments [3] [4] which require OS and application image installation on a group of virtual machines. Multicast lends itself naturally to these communication patterns. IP multicast, which has been in existence for several years, is the most common multicast implementation for traditional networks. It is prudent to carefully consider the adoption of IP Multicast to data centers.

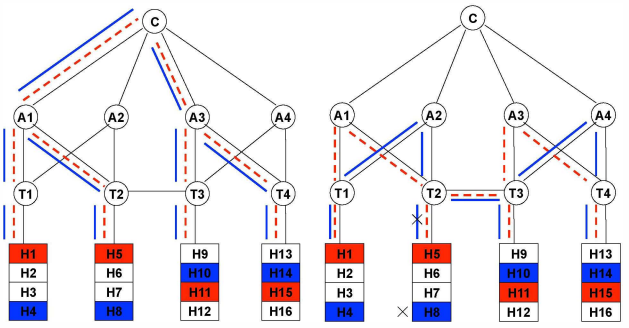
Traditional IP Multicast has remained largely undeployed in the Internet owing to concerns around reliability, scalability and security. Network protocols like TCP, which is the de facto standard for reliable unicast, incur significant latencies when applied to multicast. Ymir et. al. [5] have studied application throughput for data centers that use TCP for reliable multicast. Address aggregatability, a mechanism for reducing unicast forwarding state in switches, is not feasible with IP multicast addresses. This leads to switch state explosion as the number of multicast groups scale up. IP mulitcast allows any host to subscribe to a multicast group and start receiving group traffic. Security, therefore, becomes very difficult to enforce.

Data center networks with their structured topologies and tighter control present an opportunity to address these concerns. As a result, there has been renewed interest in multicast with specific focus on data centers. In [6], the authors propose reliable data center multicast. They leverage the rich path diversity available in data center networks to build backup overlays. In [7], the authors use multi-class bloom filters to compress multicast forwarding state. Security, though, still remains a concern. There are also some additional concerns specific to the adoption of IP multicast to data center networks. IP multicast is not designed to take advantage of path diversity which, unlike traditional IP networks, is an integral part of data center networks. This is likely to result in poor bandwidth utilization. Additionally, IP Multicast routing algorithms, of which Protocol Independent Multicast - Sparse Mode (PIM-SM) is the most common, are not designed to build optimal routing trees. PIM-SM builds trees rooted at either the source of the multicast group, or at a pre-determined rendezvous point (RP) for the group. Optimal tree building is equivalent to solving the Steiner Tree [8] problem. For arbitrary graphs, which is how traditional IP networks are, the Steiner Tree problem is known to be NP-complete. In structured graphs like those found in data center networks, however, it is actually possible to build Steiner Trees in polynomial time for some topologies. Subsequently, it is possible to build optimal or near-optimal routing trees.

The rapid emergence of Software Defined Networking (SDN), which has strong industry backing [9], provides the perfect opportunity for innovating multicast in data centers to address the aforementioned concerns. The SDN architecture uses a centralized control plane, which enables centralized admission control and policy enforcement, thereby alleviating security concerns. It also provides global visibility, as opposed to localized switch level visibility in traditional IP networks, thereby enabling greater intelligence in network algorithms. Multicast routing algorithms can thus leverage topology information to build optimal routing trees, and can leverage link utilization state to efficiently exploit path diversity typically available in data centers. Lastly, it is important to note that not all commodity switches used in data center networks have IP multicast support. SDN can be leveraged to enable multicast in such commodity switches.

In this context, we present **Avalanche** - An SDN based system that enables multicast in commodity switches used in data centers. Avalanche leverages centralized visibility and

978-1-4799-3635-9/14/$31.00 ©2014 IEEE



(a) IP Multicast (b) Avalanche

Fig. 1. Motivating Example

control of SDN to realize secure, bandwidth efficient multicast in switches that do not have any inbuilt IP multicast support. Avalanche, like IP Multicast, supports dynamic joins of multicast members. However, unlike IP Multicast, it is also able to deny admission to a member based on pre-defined policies. As part of Avalanche, we develop **AvRA**, a new multicast routing algorithm. AvRA attempts to minimize the size of the routing tree created for each group. Whenever a new member joins a group, AvRA attempts to attach it to the existing tree at the nearest attachment point. For typical data center topologies like Tree and FatTree, AvRA reduces to an optimal multicast routing (Steiner Tree building) algorithm that can be executed in polynomial time.

In this paper, we make the following contributions:

1) Detailed design of AvRA
2) Implementation of Avalanche as an OpenFlow controller module
3) Emulation of Avalanche for application performance benchmarking
4) Design and Implementation of a simulator to evaluate Avalanche at scale

The rest of this paper is organized as follows. Section II discusses the motivation behind developing SDN based multicast for data centers. In section III, we present a detailed design and implementation of Avalanche. Section IV describes our experiments and presents both emulation and simulation results. We end with a conclusion in section V. For the interested reader, appendix A provides our proof showing that the Steiner Tree can be computed in polynomial time for Tree and FatTree topologies.

II. MOTIVATION

Multicast can greatly benefit modern data centers by saving network bandwidth and improving application throughput for group communications. Since IP Multicast has been in existence for several years, it is logical to consider the adoption of IP Multicast to data centers. However, as outlined in section I, there are still unresolved concerns around security, path diversity utilization, and routing tree formation which make the adoption of IP multicast prohibitive. In this work, we identify SDN as the architecture that is capable of addressing these concerns as detailed below.

Security

SDN uses a centralized control plane. In an SDN network, when a new member sends a request to join a multicast group, the request is forwarded to the control plane. The control plane can either admit this new member and appropriately modify forwarding rules in switches, or deny admission to the member based on pre-defined policies. In this manner, SDN based multicast can enable centralized admission control and policy enforcement, thereby alleviating security concerns.

Path Diversity Utilization

In data center topologies with path diversity, there are multiple, often equal length, paths between any given hosts. Ideally, for efficient bandwidth utilization, different multicast trees should be spread out across different paths. Traditional IP networks build multicast trees based on localized switch level views stored in the form of address based routing table entries, as explained below in **Routing Tree Formation**. This results in a lot of the same links being used for different trees, while at the same time leaving a lot of links unused. SDN, on the other hand, can leverage global visibility to take advantage of path diversity and make different multicast groups use different routing trees. This leads to more even distribution of traffic across all links and avoids congestion or oversubscription of links.

Routing Tree Formation

PIM-SM, the most common IP multicast routing protocol, builds a multicast routing tree by choosing a node in the network as the RP for each group and connecting all group members to this RP. PIM-SM relies on IP unicast routing tables, which are based on localized switch level views, to find a path from the member to the RP. This results in RP-rooted trees which may be non-optimal. PIM-SM, for high data rates, provides the option for each member to directly connect to the source. In such a case, instead of RP-rooted trees, there is a combination of source-rooted and RP-rooted trees. This is still likely to be non-optimal since each member is routed to a specific node (RP or source) on the existing tree as opposed to being routed to the nearest intersection on the existing tree. SDN's global visibility, on the other hand, can be leveraged to build near-optimal routing trees. Whenever a new member joins a group, instead of finding a path from it to the source or the RP, SDN can find its nearest attachment point to the existing tree. This results in trees that use fewer hops, and in the case of topologies like Tree and FatTree, reduce to optimal trees.

This is explained with the help of an example in Fig. 1(a) and Fig. 1(b). These figures show an irregular data center topology, increasingly common in unplanned data centers, that is a combination of Tree and Jellyfish [10] topologies. It shows two multicast groups. The first group comprises of Tenant 1, whose virtual machines (VMs) reside on hosts {**H1, H5, H11, H15**}. The second group comprises of Tenant 2, whose VMs reside on hosts {**H4, H10, H14**}. **H8** has a suspicious tenant that wants to hack into Tenant 2. Fig. 1(a) shows the outcome of using IP Multicast. For each of the two multicast groups, we assume that PIM-SM chooses the core switch C as the RP. This is reasonable because C is equidistant from all members in either group. The routing trees built for the two groups are shown by the dashed and solid lines respectively. As can be seen, reliance on unicast routing tables to connect each node to the RP leads to the same links being used for each tree.

Additionally, the tree itself is not the Steiner Tree. Even if the PIM-SM protocol switches to source-based trees, which would itself incur the overhead of distributed message exchanges between all associated switches to detach from the RP-rooted tree and attach to the source-rooted tree, it would still use an overlapping set of links for each tree. It might reduce the size of the tree though. Lastly, host H8 has also been added to the routing tree for Tenant 2 and is able to receive all group traffic Fig. 1(b) shows the outcome of using Avalanche for multicast. By leveraging global visibility, Avalanche is able to exploit path diversity and at the same time build optimal routing trees. Additionally, by leveraging centralized control, it is able to deny admission to H8.

Note that in the above discussion, we have taken for granted that the switches in question have support for IP Multicast. A lot of commodity switches used in data centers have no off-the-shelf IP Multicast support. In such switches, Avalanche, most importantly, enables multicast.

III. Design and Implementation

Avalanche is designed to achieve the following goals :

- Efficiently utilize path diversity
- Enforce admission control
- Build near-optimal multicast trees
- Enable multicast support in commodity SDN switches
- Easily deployable

A. AvRA

AvRA is a polynomial time algorithm that builds a routing tree by attempting to attach each new group member to the existing tree at the nearest intersection. Instead of trying to find the shortest path from this member to a specific node, like PIM-SM does, AvRA tries to find the shortest path to the existing tree. This can be trivially accomplished, in theory, by computing the shortest path from the new member to each node on the existing tree. However, this is computationally prohibitive. AvRA performs this attachment using a unique method which completes in polynomial time. Although in theory, it is possible that AvRA may not always be able to find the best attachment point for all topologies, it does so with high probability in practice for most topologies. Specifically, for Tree and FatTree topologies, it does so with probability 1. In case AvRA is unable to find the optimal path, it still finds a path that is at least as short as that found by PIM-SM.

AvRA first assigns a level to all nodes in the network. This level classifies the node's distance, in number of hops, from a physical server. Thus, all physical servers are assigned level 0, all top-of-racks (ToRs) are assigned level 1, and so on. While creating the routing tree for a group, AvRA iterates through the group members one by one and attaches them to the tree. In this regard, the tree created is a function of the order in which members appear. Regardless of the ordering though, the tree created is near-optimal and at least as small as that created by PIM-SM. Optionally, once the group reaches a steady state in terms of number of subscribers, a steady state tree can be reconstructed. The steady state tree can be chosen as the smallest tree obtained from all possible orderings. In our system, we have not implemented steady state tree reconstruction because the trees created in the first attempt efficiently satisfy all design goals.

Algorithm 1 Avalanche Routing Algorithm

```
procedure AvRouter(members)
    AssignLevel(nodes)
    m ← Size(members)
    if m < 2 then
        return
    end if
    tree ← ShortestPath(members[1], members[2])
    for i = 3 to m do
        tree ← Hook(tree, members[i])
    end for
    return tree
end procedure
procedure Hook(tree, member)
    for all adj ∈ neighbors(member) do
        if tree.covers(adj) then
            tree.add(edge(adj, member))
            return tree
        end if
        adj.visited ← True
    end for
    for all adj ∈ neighbors(member) do
        for all adj2D ∈ neighbors(adj) do
            if adj2D.visited == False then
                if tree.covers(adj2D) then
                    tree.add(edge(adj2D, adj))
                    tree.add(edge(adj, member))
                    return tree
                end if
                adj2D.visited ← True
            end if
        end for
    end for
    if member.level < maxLevel then
        candidate ← RandUpperNeigh(member)
        tree ← Hook(tree, candidate)
        if tree ≠ null then
            tree.add(edge(candidate, member))
        end if
    else
        return null
    end if
    if tree == null and member.level == 0 then
        tree ← tree.add(BFS(member, tree))
    end if
    return tree
end procedure
```

Tree building begins once there are at least 2 members in the group. To connect the first 2 members, the algorithm chooses the shortest path between them. Subsequently, whenever a new member appears, the algorithm tries to find its nearest intersection to the existing tree. To do so, it first checks if any of its adjacent nodes reside on the existing tree. Thus, when a new member, which would by definition be a level 0 node, appears, all its adjacent (level 1) nodes are checked. If any of these nodes already reside on the existing tree, the new member is simply attached to the tree at this point. If none of these adjacent nodes lies on the tree, the algorithm then looks at all neighbors (level 0, level 1 and level 2) of the adjacent nodes. If any of these neighboring nodes lies on the existing tree, the algorithm attaches the new member to the tree at this point.

If neither this new member's adjacent nodes nor their

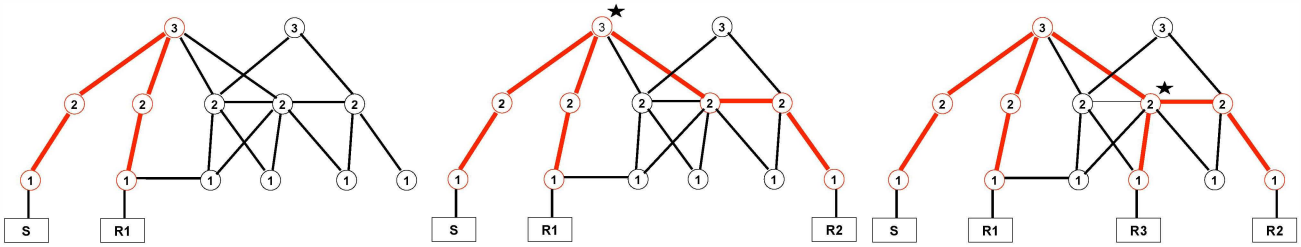


Fig. 2. Avalanche : Routing tree formation with AvRA

neighbors lie on the existing tree, then one of the member's adjacent nodes at the next higher level is randomly chosen. Note that in this case, the new member has not yet been attached to the tree so the algorithm continues. Next, this chosen adjacent node is set as the current node. Now, its adjacent nodes (some of which would have already been examined in the previous iteration) and their neighbors are examined to see if any fall on the existing tree. If any of them does, the new member is connected to the tree at this node by tracing the path chosen from the new member onwards. If, on the other hand, neither of them lies on the existing tree, the algorithm continues by randomly choosing one of the current node's adjacent nodes at the next higher level. This chosen node is now set as the current node. In this manner, the algorithm continues until either the new member is connected to the existing tree or until the level of the current node reaches the highest level in the topology.

If the algorithm has already reached the highest level and has still been unable to attach the new member onto the tree, then it resorts to a breadth first search (BFS) with a stop condition that terminates as soon as an attachment point to the tree is found. For typical data center topologies, which are characterized by rich path diversity and large number of edges at higher levels, it is unlikely that the algorithm would reach a highest level node in the topology without attaching the new member to the tree. At every iteration that the algorithm is unable to attach the member, it randomly moves to a higher level thereby increasing its chances of finding an attachment point (owing to the larger number of edges at the higher level) in the next iteration. In case the algorithm randomly selects a node that is headed in a direction away from the tree, in the next iteration a random selection once again is likely to head it back towards the tree. This approach of randomly selecting higher level nodes, from the perspective of building routing trees for different multicast groups, also contributes towards better utilization of the available path diversity, leading to more balanced link utilizations. If in case the algorithm does indeed reach the highest level without converging, which as mentioned above is unlikely, the overhead incurred from this unsuccessful search is very small since typical data center topologies are only 3 to 4 levels high. This is computationally far less expensive than using BFS for each new member. Additionally, as demonstrated in section IV, this approach still results in smaller routing trees than PIM-SM. Specifically, for Tree and FatTree topologies, this algorithm always finds the optimal attachment point for each new member without having to resort to BFS.

We explain AvRA with the help of an example in Fig. 2. It demonstrates how a tree is constructed as new members join a multicast group. Initially, there is one sender S and one

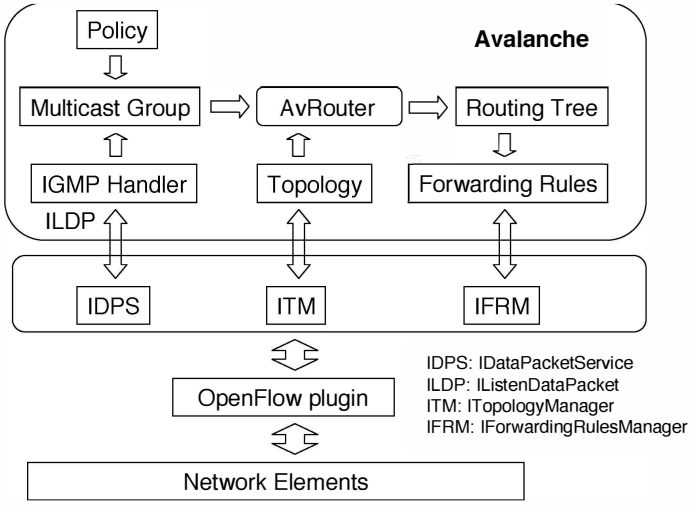


Fig. 3. Architecture of the Avalanche OpenDaylight module

receiver R1. The tree is constructed by choosing the shortest path from R1 to S. Subsequently, a receiver R2 also subscribes to the multicast group. None of this receiver's adjacent nodes are on the tree, nor are the neighbors of these adjacent nodes on the tree. In fact it has only one adjacent node, a level 1 node, which is therefore chosen by default as the node that will lead this member to the tree. Next, setting this level 1 node as the current node, AvRA looks at all its adjacent nodes as well as their neighbors. Again, this level 1 node has only one neighbor, a level 2 node, so it is chosen by default. Now, this level 2 node becomes the current node, and AvRA looks at its adjacent nodes. None of its adjacent nodes are on the tree. However, at least one of the neighbors of one of these adjacent nodes is on the tree. This adjacent node is the level 2 node located horizontally to the left of the current (level 2) node. Thus, AvRA selects this adjacent node. Finally, AvRA attaches the new member to the existing tree at this adjacent node's neighbor (the level 3 mode marked by a *)

Finally, a last receiver R3 arrives. In the first iteration for R3, AvRA chooses the level 1 node immediately adjacent to it since there is no other choice. In the next iteration, with this level 1 node set as the current node, AvRA first looks at its adjacent nodes. As it turns out, one of its adjacent nodes (the level 2 nodes marked by a *) is on the tree. So, it attaches to the tree at this node.

B. Avalanche System Implementation

We implement Avalanche as an OpenFlow controller module, using the OpenDaylight [9] SDN platform, as outlined in Fig. 3. The Avalanche module listens for subscription requests and topology changes from the network, and dynamically updates the appropriate multicast routing trees. It registers

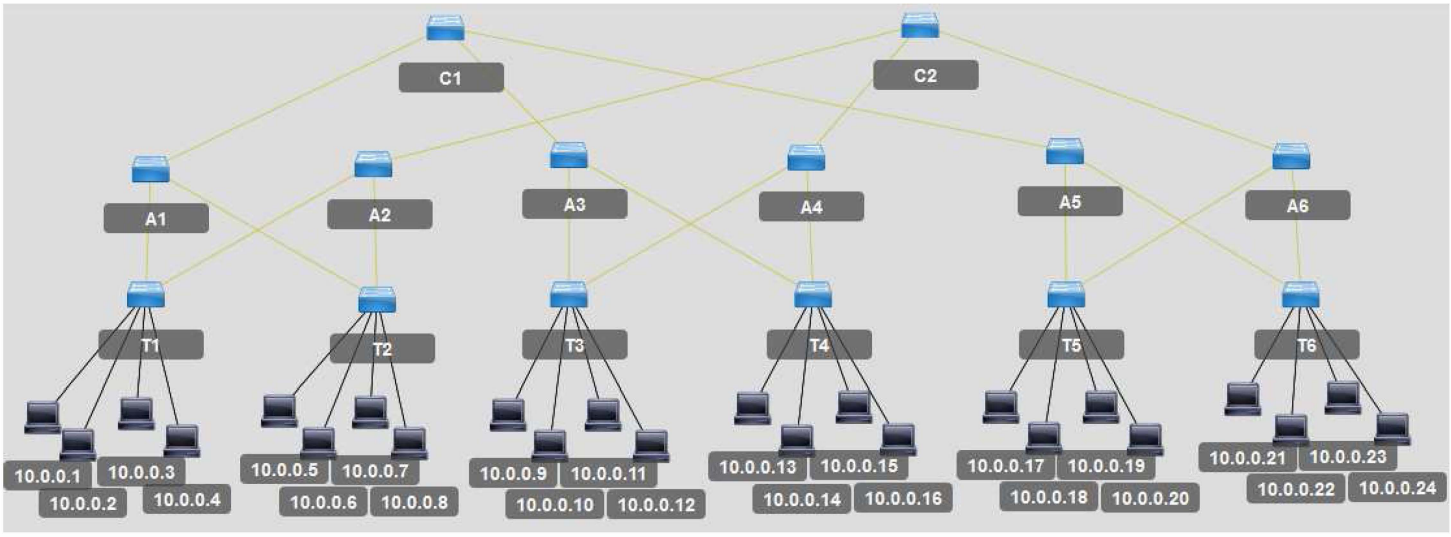


Fig. 4. Mininet topology used for emulating Avalanche

with the *IListenDataPacket* service of OpenDaylight to indicate that it wants to receive all data packets sent to the *IDataPacketService* of the controller. In our implementation, we adopt the IP Multicast addressing scheme, so hosts still send subscription requests through IGMP packets. Avalanche implements IGMP snooping to learn about hosts that wish to subscribe to or unsubscribe from a multicast group. On receiving an IGMP packet, Avalanche finds the address of the host as well as the multicast group it wishes to join or leave. Subsequently, it examines security policies to ensure that this member can be admitted and, if so, updates the multicast routing tree using AvRA. When a multicast group sender, which hasn't subscribed to the group yet since senders do not send IGMP packets, starts sending multicast traffic, the controller is notified. Subsequently, Avalanche automatically adds the sender to the multicast tree, once again, assuming policies permit this. Avalanche also appropriately modifies routing trees whenever a topology change is registered from the *ITopologyManager* in OpenDaylight. Anytime Avalanche needs to update the routing tree, it effectively has to add, delete or modify routing rules in appropriate switches. This is done through the *IForwardingRulesManager*.

The OpenDaylight controller's service abstraction layer uses a southbound plugin to communicate with network elements. Currently, OpenDaylight has only one southbound plugin which supports the OpenFlow v1.0 protocol. Avalanche can be completely implemented using the features provided by OpenFlow v1.0. Avalanche can work with higher versions of OpenFlow as well.

IV. Results

A. Emulator

To validate and evaluate our implementation of Avalanche, we used Mininet Hi-Fi [11], an OpenFlow v1.0 enabled network emulation platform. We created a Mininet network topology and connected it to the OpenDaylight controller. For this emulation, we chose a FatTree topology, as shown in Fig.4. FatTree is a common data center topology which has sufficient path diversity to highlight the benefits of Avalanche.

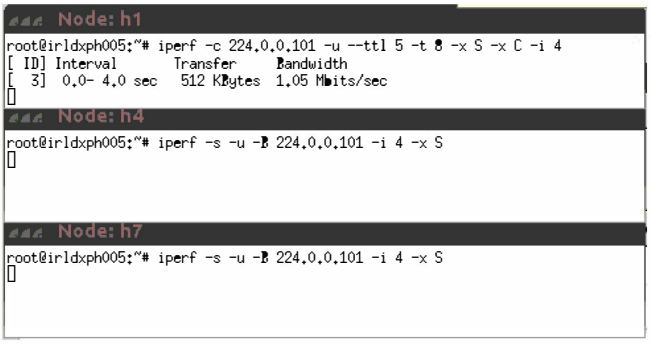


Fig. 5. Multicast is not supported out-of-the-box for OVS switches

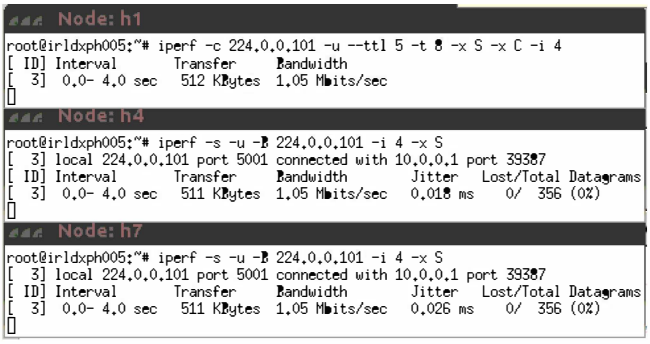


Fig. 6. Avalanche enables multicast for OVS switches

Our topology comprises of 24 hosts, 6 ToR switches, 6 aggregation switches, and 2 core switches. The link capacity for each link in this network is set to 10Mbps. For performance benchmarking, we use iperf [12]. Throughout this section, host **hx** refers to the host with IP address 10.0.0.x in Fig. 4.

First, we seek to validate the functionality of Avalanche by ensuring that it is able to enable multicast in a Mininet network. OVS switches used by Mininet do not have any out-of-the-box multicast support. This is demonstrated in Fig. 5. A multicast group is created with hosts h1, h4 and h7. An iperf client (sender) is started on h1, while iperf servers (receivers) are started on hosts h4 and h7. As evident, the servers do not receive any multicast traffic. Next, we include Avalanche in

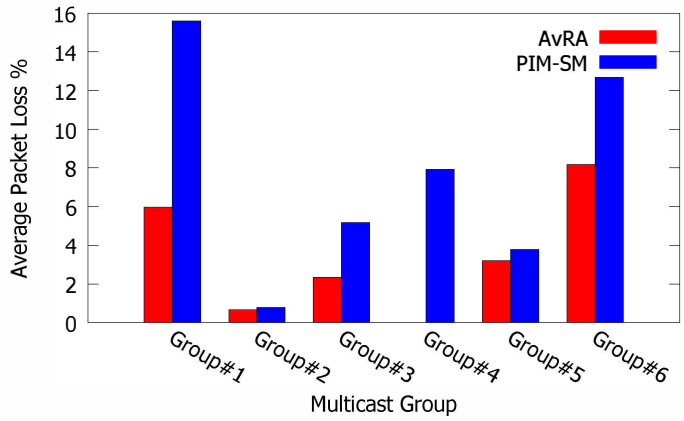


Fig. 7. Average Packet Loss Percent

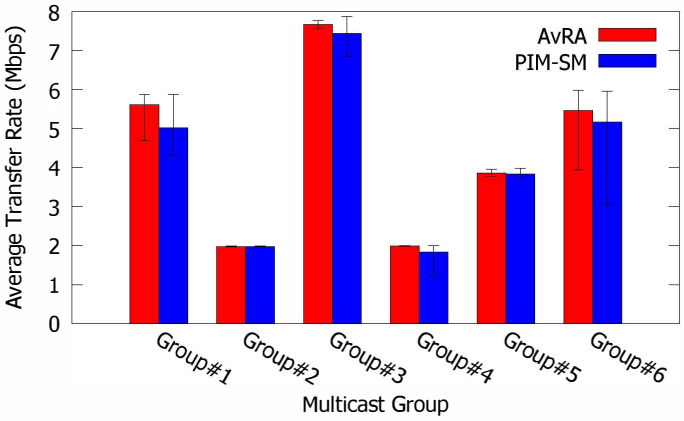


Fig. 8. Average Transfer Rate

the OpenDaylight controller and restart the iperf client and servers. As shown in Fig. 6, multicast has now been enabled.

Next, we seek to evaluate Avalanche by comparing its iperf performance with that of IP Multicast. Once again, we would like to point out that we adopt the addressing scheme of IP Multicast. The iperf application has been written to use the IP addressing scheme, and we merely leverage that for ease of implementation. Using this addressing scheme for Avalanche doesn't have any bearing on the results. The Avalanche system is completely independent and separate from IP Multicast.

Since IP Multicast is not supported out of the box with OVS switches, we also implement an adaptation of IP Multicast for our software defined environment. While we still leverage the OpenDaylight controller to create IP Multicast routing trees by installing appropriate forwarding rules in switches, there are two important differences in the implementation of IP Multicast compared to Avalanche. These are to ensure that our implementation of IP Multicast mirrors traditional IP Multicast.

1) IP Multicast does not leverage central visibility available to the OpenDaylight controller. It relies on localized switch level views.
2) IP Multicast uses PIM-SM for routing.

We tried to incorporate XORP, a routing engine which implements PIM-SM, along with RouteFlow, a service that facilitates communication between the controller and the routing engine. However, the current implementation of RouteFlow is not capable of converting the multicast routing tree generated by XORP into corresponding OpenFlow rules. Hence, in our

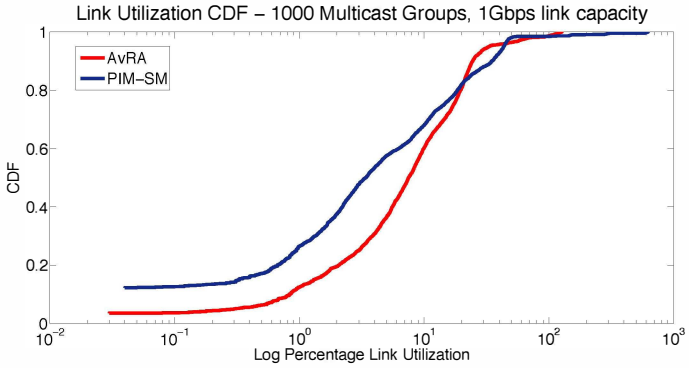


Fig. 9. Link Utilization CDF for 1000 Multicast Groups, Portland Topology

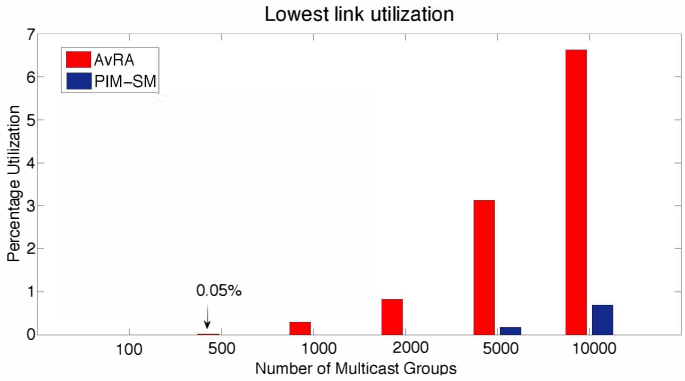


Fig. 10. Lowest Percentage Link Utilization

system, we implement PIM-SM ourselves.

For iperf performance comparison, we create 6 random multicast groups of sizes varying from 3 to 6 multicast members. The sender in each group uses iperf in client mode to send multicast traffic at a rate chosen randomly from {2, 4, 6, 8} Mbps. The remaining members of the group use iperf in the UDP server mode to bind to the multicast group. The packet loss percentages associated with Avalanche and IP Multicast are shown in Fig. 7, while data rates are shown in Fig. 8. The results from our emulation show that Avalanche results in throughput increase by upto 12% and packet loss reduction by 51% on average, compared to IP Mulitcast.

B. Simulator

To evaluate the performance of Avalanche, we built a multicast simulator that comprises of the following modules:

1) Topology Generation
2) VM Placement
3) Multicast Group Generation
4) Avalanche
5) IP Multicast

The simulator first creates a network topology based on user input specifying the total number of servers, the number of servers per rack, and the type of topology. The topology generation module, currently, is capable of generating two types of topologies - Tree and FatTree. For either type of topology, the topology generator determines the number of switches and arranges them appropriately by creating the necessary host-switch and switch-switch edges. Alternately, if a topology other than Tree or FatTree needs to be used, it can be supplied as a file to the simulator. This would bypass

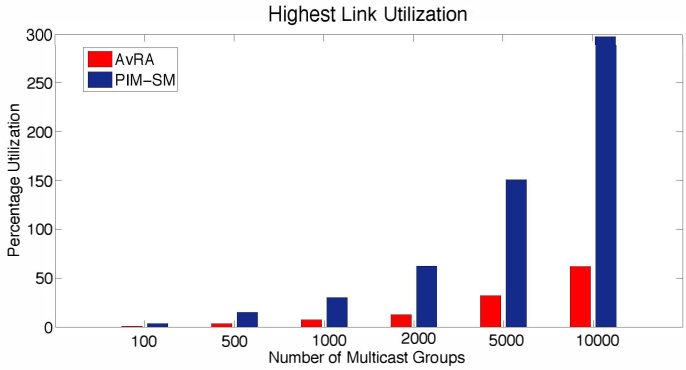


Fig. 11. Highest Percentage Link Utilization

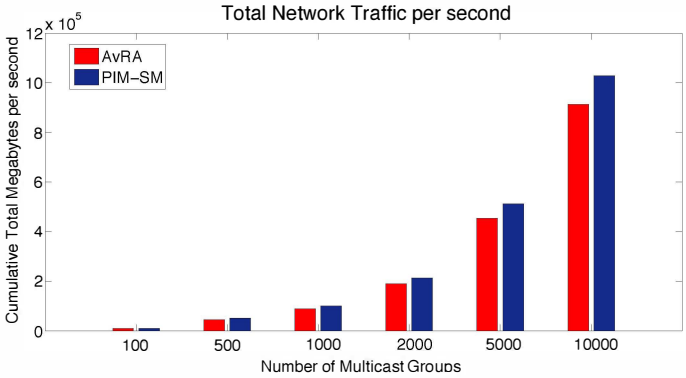


Fig. 12. Cumulative total Megabytes in the network per second

the topology generation module. Next, the simulator prompts the user to specify the number of Virtual Machines (VMs) running in the data center. Each VM is mapped randomly onto one of the servers. All communication is assumed to be between VMs. Next, the simulator asks the user to specify the number of multicast groups that need to be routed. The simulator also allows the user to supply the member VMs for each multicast group, along with the group's associated data rate . If member VMs are not supplied, the simulator randomly chooses anywhere between 3 to 20 VMs as the members for each group. Additionally, it also randomly assigns a data rate (in 100s of kbps) to each group's traffic. The simulator implements both Avalanche and IP Multicast. For Avalanche, the simulator assumes an SDN environment with centralized visibility into the network. It uses AvRA to build multicast trees. For IP Multicast, the simulator assumes localized views derived from routing tables stored in switches. It uses PIM-SM to build multicast trees.

In our simulations, we create both Tree and FatTree topologies. For each topology, we specify 11520 servers assembled into 40 servers per rack, thereby resulting in 288 racks. This distribution of servers is adopted from PortLand [13] which in turn interprets this from [14] [15] [16]. The simulator determines the number of switches required as 288 top-of-rack(ToR) switches, 24 aggregation switches and 1 (for Tree) or 8 (for FatTree) core switches . We specify the number of VMs as 100,000, and the simulator randomly places each VM on the 11520 servers. Since there is no available real data trace for data center multicast, we let the multicast groups be generated automatically by the simulator. To create these multicast groups, the simulator applies the methodology described in [17]. Additionally, the simulator assigns a random data rate

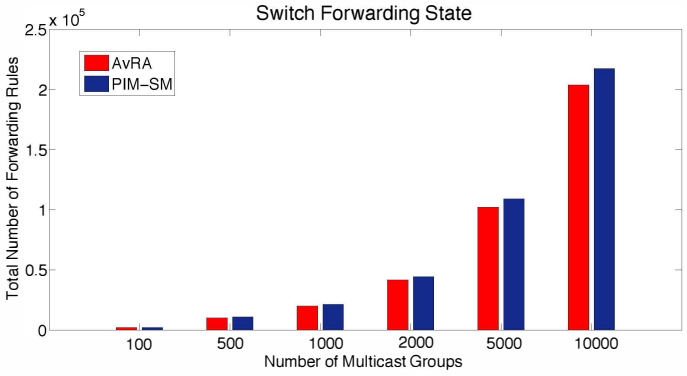


Fig. 13. Multicast Forwarding State across all switches

for each multicast group, which is randomly chosen from the range 100kbps to 10Mbps. Here, we only present results from our simulation runs on the FatTree topology. The results from Tree topology were similar.

For our first simulation run, we set the link capacity for each link to 1 Gbps and create 1000 multicast groups. Fig. 9 shows the CDF of link utilizations in the network. The following observations are made from this plot:

- The percentage of unutilized links is 0% in Avalanche while in IP Multicast it is 16%
- The percentage of links that have less than 5% utilization in Avalanche is 49% while in IP Multicast it is 65%.
- The maximum link utilization in Avalanche is 73%, while that in IP Multicast is 301%. In fact, IP Multicast has 1.5% links with utilization greater than 100%.

The above observations establish that Avalanche is able to take better advantage of the available bandwidth in the network. As the number of multicast groups increase, the inability of traditional IP multicast to efficiently utilize path diversity gets magnified even further.

For the rest of our simulation runs, we vary the number of multicast groups from 100 to 10000. Also, we increase link capacity from 1Gbps to 10Gbps in order to accommodate this large number of multicast groups. Our results are as follows:

- Fig. 10 shows that the lowest link utilization percentage in IP Multicast stays at 0% even when the number of multicast groups becomes as high as 2000. Avalanche, on the other hand, has non-zero lowest link utilization (i.e. it utilizes all links), once the number of groups exceeds 500.
- Fig. 11 shows that the highest link in IP Multicast goes beyond 150%, which implies 50% oversubscription, for 5000 multicast groups and beyond 300%, which implies 200% oversubscription, when the number of groups becomes 10000. The maximum utilization for Avalanche, on the other hand, stays well below 100%.
- Fig. 12 shows the cumulative total number of megabytes sent out in the network every second. Avalanche leads to up to 8% lesser traffic than IP Multicast. These gains are expected to be higher for asymmetric topologies like Jellyfish.
- Fig. 13 shows the cumulative total number of multicast forwarding rules stored in switches. Avalanche leads to up to 5% reduction in switch state implying that it creates shorter trees. The results are expected to be higher for asymmetric topologies like Jellyfish.

V. Conclusion

Reliability, scalability and security concerns have resulted in IP Multicast's poor adoption in traditional networks. Data center networks with their structured topologies and tighter control present an opportunity to address these concerns. However, they also introduce new design challenges, such as path diversity utilization and optimal tree formation, that are not critical in traditional networks like the Internet. In this paper, we presented Avalanche - An SDN based system for enabling multicast in data centers. Avalanche leverages global visibility and centralized control of SDN to create secure and bandwidth efficient multicast. Avalanche implements its own routing algorithm, AvRA, that creates optimal routing trees for common data center topologies. Our implementation of Avalanche as an OpenFlow controller module validates its deployability, and our simulation establishes its scalability. We are currently working on incorporating reliability in Avalanche and exploring the adoption of reliability protocols such as PGM [18]. We are also working on porting common group communication applications, such as OS image synchronization and Hadoop, to use Avalanche.

References

[1] "Hadoop." [Online]. Available: http://hadoop.apache.org/

[2] "IBM WebSphere." [Online]. Available: http://www-01.ibm.com/software/websphere/

[3] "Amazon EC2." [Online]. Available: http://aws.amazon.com/ec2/

[4] "Windows Azure." [Online]. Available: http://www.windowsazure.com/en-us/

[5] D. Basin, K. Birman, I. Keidar, and Y. Vigfusson, "Sources of instability in data center multicast," in *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware*. ACM, 2010, pp. 32–37.

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

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

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

[9] OpenDaylight. [Online]. Available: http://opendaylight.org/

[10] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proc. of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2012, pp. 17–17.

[11] "Mininet Hi-Fi." [Online]. Available: https://github.com/mininet/mininet

[12] "iperf." [Online]. Available: http://iperf.fr/

[13] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: a scalable fault-tolerant layer 2 data center network fabric," in *SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 39–50.

[14] "Inside Microsoft's $550 Million Mega Data Centers." [Online]. Available: www.informationweek.com/news/hardware/data_centers/showArticle.jhtml:q?articleID=208403723

[15] "Cisco Data Center Infrastructure 2.5 Design Guide." [Online]. Available: www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf

[16] Configuring IP Unicast Layer 3 Switching on Supervisor Engine 2. [Online]. Available: www.cisco.com/en/US/docs/routers/7600/ios/12.1E/configuration/guide/cef.html

[17] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. of the 7th USENIX conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2010, pp. 19–19.

[18] J. Gemmell, T. Montgomery, T. Speakman, and J. Crowcroft, "The pgm reliable multicast protocol," *Network, IEEE*, vol. 17, no. 1, pp. 16–22, 2003.

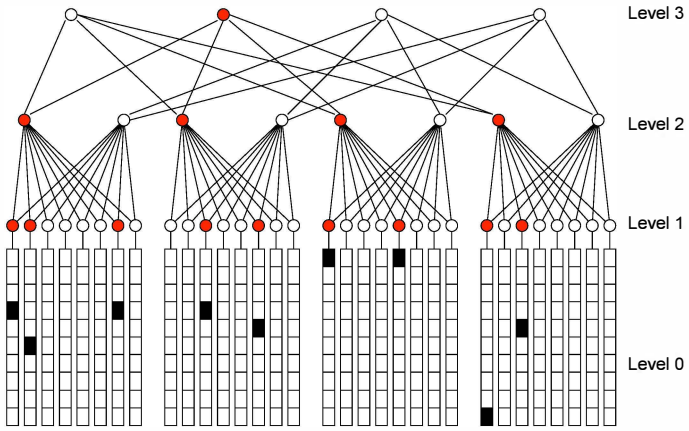


Fig. 14. Steiner Tree Building

Appendix A
Steiner Tree in Polynomial time for FatTree

Steiner Tree Problem

Given a connected undirected graph G=(V,E) and a set of vertices N, find a sub-tree T of G such that each vertex of N is on T and the total length of T is as small as possible.

The Steiner Tree Problem for an arbitrary graph is NP-complete. In this section, we prove that the Steiner Tree problem for FatTree topologies can be solved in polynomial time. Fig 14 shows a FatTree graph with the set of vertices that need to be connected, N, indicated by the black tiles.

Our proof strategy comprises of the following two steps:

1) Build a tree (in polynomial time) that connects all vertices in N.
2) Show that the tree thus constructed is the Steiner tree.

Owing to symmetry in FatTree graphs, a given cluster of X nodes at level L connect identically to a given cluster of Y nodes at level (L+1). In other words, there is a mapping $X(L) \Leftrightarrow Y(L+1)$. Specifically, in the graph shown in Fig. 14, clusters of 10 nodes (hosts) at level 0 connect to 1 node (ToR) at level 1, clusters of 8 nodes (ToRs) at level 1 connect to clusters of 2 nodes (aggregations) at level 2, and clusters of 4 nodes (aggregations) at level 2 connect to clusters of 2 nodes (cores) at level 3. For each Level L cluster X, one specific node is chosen as the **designated node** from its corresponding Level (L+1) $y \epsilon Y$ cluster for all group traffic to or from X. The relative orientation of y(L+1) w.r.t cluster X(L) is kept identical across every mapping $X(L) \Leftrightarrow Y(L+1)$. This is shown in Fig. 14 with the help of red dots. For every vertex in N, which is a level 0 node, there is only choice for a level 1 designated node. The designated level 1 nodes for those level 0 node clusters that have at least one multicast group member are shown in red. Next, for each level 1 cluster (there are 4 clusters of level 1 nodes with 8 nodes in each), the first (left) of the two level 2 nodes is chosen as the designated node. Finally, for each level 2 node thus chosen, the second (right) level 3 node is chosen as the deisgnated node. The choice of designated node doesn't matter as long as the relative orientation of each level (L+1) designated node w.r.t its child Level L cluster is the same throughout the topology. It can be seen that the tree thus created by joining all designated nodes connects all group members to each other, and is thus a multicast routing tree. This tree can be constructed in polynomial time, since each new member can be connected to the existing tree in bounded time (it is just a matter of following designated nodes from it until the tree is reached) and the number of members itself is bounded. Also, it can be seen that the tree thus created connects any given pair of nodes in N by the shortest path between them. Therefore, it is the Steiner Tree.

Hence, a Steiner Tree can be created in polynomial time for FatTree topologies. Since Tree is a special case of FatTree, Steiner Trees can be created in polynomial time for Tree topologies by corollary.