Scaling IP Multicast on Datacenter Topologies

Xiaozhou Li and Michael J. Freedman
Princeton University

ABSTRACT

IP multicast would reduce significantly both network and server overhead for many datacenter applications' communication. Unfortunately, traditional protocols for managing IP multicast, designed for arbitrary network topologies, do not *scale* with aggregate hardware resources in the number of supported multicast groups. Prior attempts to scale multicast in general settings are all bottlenecked by the forwarding table capacity of a *single* switch.

This paper shows how to leverage the unique topological structure of modern datacenter networks in order to build the first *scale-out* multicast architecture. In our architecture, a network controller carefully partitions the multicast address space and assigns the partitions across switches in datacenters' multi-rooted tree networks. Our approach further improves scalability by locally aggregating multicast addresses at bottleneck switches that are running out of forwarding table space, at the cost of slightly inflating downstream traffic. We evaluate the system's scalability, traffic overhead, and fault tolerance through a mix of simulation and analysis. For example, experiments show that a datacenter with 27,648 servers and commodity switches with 1000-entry multicast tables can support up to 100,000 multicast groups, allowing each server to subscribe to nearly 200 multicast groups concurrently.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

IP Multicast; Datacenter Networks; Scalability

1. INTRODUCTION

Many datacenter applications rely on multicast communication patterns, such as publish-subscribe services for data dissemination [31], web cache updates [33], system monitoring [28], and so on. Further, emerging datacenter virtualization standards that bridge multiple subnets into one large layer-2 VLAN, such as VXLAN [27] and NVGRE [39], often translate broadcasts in the virtualized subnet into multicasts in the physical network. IP multicast offers a prime means to support these types of communication, as it greatly conserves network bandwidth and reduces server load.

Unfortunately, IP multicast is often unscalable and unstable [11], which causes network operators to eschew its use. While various multicast protocols have been developed by researchers over the past two decades to address reliability [8, 13, 3], security [22, 2], and congestion control [44], our work aims to scale multicast in terms of the number of supported multicast groups in datacenter networks. We believe this to be an important enabling property for the adoption of datacenter IP multicast. Scalability limitations arise because switches only support limited numbers of multicast addresses in their forwarding tables (e.g., 100s to 1000s of entries [30]). This problem is compounded because multicast addresses are not topologically assigned and thus cannot be hierarchically aggregated in the traditional fashion. Network failures also introduce challenges for IP multicast. Rerouted or retransmitted packets may be resent to large numbers of group subscribers, and reconstructing multicast trees is an expensive process.

While these problems with IP multicast are not new and have been well documented in both ISP and enterprise networks [11], we reconsider possible solutions by leveraging the unique topological properties of modern datacenter network architectures. The multicast addressing and routing mechanisms we present scale to much larger numbers of multicast groups than in previous designs, while providing greater robustness to switch and link failures. In particular, we address IP multicast within datacenter networks that are multi-rooted trees [6, 15, 29, 26]. We exploit the properties of these network topologies to make four key contributions.

First, we introduce a general method for scaling out the number of supported multicast groups. Rather than treating each switch as an independent entity—as done by all prior work, perhaps given their starting point using fully decentralized control protocols like IGMP—we leverage ideas from scale-out storage systems to *partition* the multicast address space and *distribute* address partitions across cooperating switches. In particular, all "core" switches in the datacenter network act as one cooperating set, as well as each set of upper-layer switches in each datacenter network "pod". Yet because the aggregate capacity of each pod's upper layer is less than that of the core layer, the number of groups supported by the datacenter depends on a number of factors, which we later quantify. In summary, however, this mechanism allows a fat-tree network, when composed of more than 27K servers and with switches holding at most 1000 group addresses, to support 4K to 30K multicast groups (depending on the distribution and placement of group members).

Second, we further increase the network's group capacity through local multicast address aggregation. Unfortunately, multicast ad-

CoNEXT'13, December 9-12, 2013, Santa Barbara, California, USA.
ACM 978-1-4503-2101-3/13/12.
http://dx.doi.org/10.1145/2535372.2535380.

dresses do not naturally aggregate into compact IP prefixes, as group members may be spread across the network. Instead, we introduce a novel indirection and rewriting mechanism that aggregates local groups into virtual *meta-groups* for addressing and routing. In doing so, we provide an efficient heuristic to the NP-Hard *channelization problem* [5], while minimizing additional downstream traffic load arising from such aggregation. With local meta-groups, the same network as above supports 10K to 100K multicast groups.

Third, we provide mechanisms that are resilient and adapt quickly to switch and link failures. In particular, prior to initiating or during the process of multicast tree reconstruction for longer-term failures, we support fast failover through local multicast rerouting. This rerouting avoids redundant traffic caused by rerouting or retransmitting multicast packets.

Finally, the new multicast mechanisms we introduce are deployable in today's switches. Namely, they can be implemented using OpenFlow-compliant switches [32]—which support prefix forwarding, multicast addresses, packet rewriting, and a remotely-configurable forwarding plane—and software-defined networking (SDN) controllers for remote management. In particular, each set of cooperating switches (i.e., at the core level and in each network pod) can be managed independently for both address allocation and failure recovery. Our mechanisms should also be applicable to datacenter networks primarily using layer-2 forwarding (e.g., [29]), although we do not further detail such approaches.

2. TOWARDS SCALABLE MULTICAST

We next review the current challenges in using IP multicast within datacenters. We also detail the opportunities provided by emerging multi-rooted trees as datacenter network topologies, as well as the use of software controllers to remotely manage switches.

2.1 Limitations of Today's IP Multicast

Many datacenter services use group communication that could benefit greatly from multicast. However, current IP multicast protocols face severe scalability challenges, largely in terms of the number of supported multicast groups and their robustness against network failures. These challenges arise in both control and data planes of the network.

In the control plane, most current multicast deployments use IGMP [10] to manage group membership and PIM protocols [12, 18, 17] to construct multicast routing trees. Such protocols require the switches[1] to track the status of all associated groups, which may involve many control messages and periodic queries to each broadcast domain. Switches need additional memory to handle these control-plane tasks; this serves to limit the number of supported multicast groups [43]. Further, existing multicast protocols behave poorly with network failures. A single point of failure may affect many multicast trees, and reconstructing a multicast tree may need many network communication and redundant states in switches.

The number of multicast groups is also limited by multicast forwarding table sizes in switches. Multicast addresses cannot be aggregated by prefixes; thus, a switch has to maintain per-group routing rules for all associated multicast groups. Though the hardware implementation of multicast forwarding may vary for different switches, the number of multicast entries that a switch supports is typically a scarce resource. This is particularly true for the commodity switches common to datacenters, which have much smaller forwarding tables than the high-end routers found in ISP networks, and even less rule-table space available to multicast entries. For example, benchmarks conducted in 2008 found popular commodity switches to support as few as 70 and as many as 1500 multicast addresses per switch [30].

2.2 Next-Generation Datacenter Architecture

Given the dynamic nature of datacenter traffic and the significant network demands of data-centric processing, modern datacenter network designs seek to provide high bisection bandwidth, reducing or even eliminating any network oversubscription. To do so, the research community has extensively studied the use of multi-rooted tree network topologies (e.g., the Clos network [15] or its fat-tree variants [6, 29, 26]), and such topologies are beginning to be deployed in production [15, 1]. This makes them an attractive target for further investigation, as opposed to some more radical design proposals that use more exotic topologies [16, 4, 37, 38]. While the multicast techniques described in this paper generalize to multi-rooted tree topologies, we focus our design and evaluation on the fat-tree network.

Figure 1 shows three examples of fat trees built with 4-port switches. In general, a 3-tiered fat tree consists of a top layer of core switches and multiple pods. Each pod has two layers of switches: the edge switches that connect to end hosts and the aggregation switches that connect to the core layer. Each pod switch uses half of its ports to connect with higher-level switches and the other half for lower-level devices. All core switches are connected to every pod, and all aggregation switches are connected to every edge switch in the same pod. With such a design, all hosts can communicate with *any* other host at full bandwidth of their network interface through multiple equal-cost paths. The difference between the three types of fat trees in the figure is mainly related to their behavior under failure, as discussed later.

2.3 Scale-Out IP Multicast in Datacenters

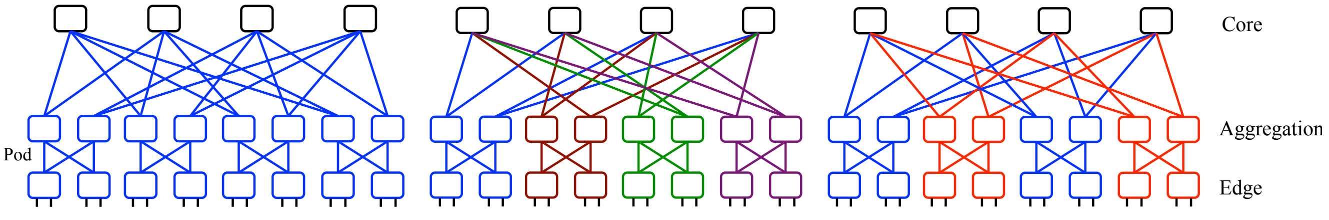
With an eye towards the multi-rooted tree topology of datacenter networks, we introduce three techniques to increase the number of multicast groups and enhance communication robustness against common network failures. Here, we present a high-level motivation and description of these techniques, while Section 3 specifies our algorithms and provides analysis justifying each technique.

1. Partition and distribute the multicast address space to increase the number of groups at core and aggregation layers. To leverage the potential advantages of a multi-rooted tree topology, we partition the multicast address space into multiple blocks with unique prefixes. Each partition is assigned to some core switches at the top level, and to some aggregation switches in every pod. So, each core or aggregation switch only has a fraction of the multicast address space. Yet, the system coordinates the forwarding state between switches, so that the entire multicast forwarding state is stored collectively by all core switches, and by all aggregation switches in each pod (similar to keyspace partitioning in scale-out storage [23]).

With such an approach—which we refer to as *multicast address distribution*—the datacenter network as a whole can support many more groups than the capacity of a single switch. When combined with the multi-rooted tree topology, these partition assignments ensure that multicast packets of each group still have a sufficient number of equal-cost paths to reach all group members.

2. Enable local multicast address aggregation to further increase the number of groups in each pod. Multicast addresses are hard to aggregate, since they are not assigned based on locations.

[1] We use the term switches and routers indistinguishably. Datacenter network devices handle both layer-2 and layer-3 traffic, and they support a variety of management and routing protocols.



(a) "Standard" fat tree [6] (b) Portland fat tree [29] (c) AB fat tree [26]

Figure 1: Three types of fat tree topologies built with 4-port switches. All use a basic pod structure with fully connected aggregation and edge switches, and differ only in the connectivity between core and aggregation switches. Pods using different connectivity patterns in each network are represented by different colors.

We introduce a local aggregation[2] mechanism to aggregate multicast entries in switches, by locally rewriting the addresses of groups with same or similar forwarding state to new local addresses sharing a common IP prefix. In this fashion, the switch can forward multicast packets through prefix matching.

This process requires per-group address translation in switches both above and below the layer of interest (which in this figure is the aggregation layer): the layer above maps the real multicast address to a new, local address, and the layer below maps it back to its real one. To enable the greatest scalability, the layer of interest across which this occurs should be the pod's bottleneck in terms of the number of supported groups.

Local group aggregation may cause redundant traffic on downstream links, as all traffic to the common prefix will be sent out all downstream ports that have groups sharing the prefix. We model this problem—minimizing link congestion while aggregating groups subject to switches' multicast address capacity—as an optimization problem. While this problem is NP-hard, we provide an heuristic that aggregates groups effectively with low cost.

3. Handle failures with fast rerouting and multicast tree reconstruction. It usually takes a longer period for multicast routing to recover from network failures, since reconstructing a multicast tree is more complex than recomputing a unicast route. Thus, instead of waiting for the multicast tree to be reconstructed every time a failure occurs, we seek to quickly reroute the packets to bypass the failed links or switches, especially when dealing with short-term failures.

Recent work [26] shows how to rapidly restore unicast routes after a switch or link failure, by using local rerouting. However, rerouting multicast traffic introduces its own challenges, as a switch may re-multicast a rerouted packet to all group members, even if most are not affected by the failure. This would introduce lots of redundant traffic.

To enable local rerouting in multicast, we add a *location identifier* to the headers of the rerouted packets. This allows switches to know where to forward the rerouted packet (i.e., to which pod), without causing unnecessarily duplicated traffic. Our evaluation shows that such local rerouting yields both low route stretch and high availability. While such rerouting occurs, the network controller can recompute the multicast routing tree (with less urgent priority) and disseminate updated rules to the appropriate switches.

2.4 Managing Multicast using SDN

Rather than using decentralized management protocols like IGMP, emerging standards in *software-defined networking* (SDN) [32] provide new opportunities for programmatic control and global visibility over network switches. Our design assumes the datacenter runs an SDN platform with centralized or distributed software controllers to manage the membership of multicast groups, to compute multicast routing trees, and to disseminate forwarding rules to the appropriate switches. We designed our protocols to use features found in today's OpenFlow-enabled commodity switches, which are commonly capable of prefix routing, multicast forwarding, VLAN tagging, rewriting packet headers, and group operations. Such SDN architectures have already been deployed in production datacenters (e.g., [24]), as well as in backbone networks linking datacenters (such as by Google [20]).

Rather than having switches query their subnets to determine multicast group membership (as in IGMP), network controllers can collect information about the group subscriptions of virtual machines (VMs). Our architecture is agnostic about how this information is collected. For example, datacenters using a centralized VM manager (like VMWare vSphere) can have this manager pass such information to the controllers directly; alternatively, each server's hypervisor can keep track of the group memberships of its VMs and inform its controller(s) about any changes. Both approaches reduce the control overhead and memory use at switches.

Our protocols can be easily supported by fully *distributed* network controllers (e.g., Onix [24]). One of the controllers can compute the initial address partitioning and switch assignment when the network is brought online. Then each controller can compute the local address aggregation for switches in its subnet, and update multicast forwarding tables in the local switches in response to the group join/leave events or network failures in the subnet. Recent studies show that a network controller running on a commodity server can handle 1.6 million requests per second [41], and today's OpenFlow switches can set up 600-1000 flows per second [34], enough to handle common multicast group dynamics.

2.5 Related Work

Given the large body of research literature on IP multicast, we restrict our consideration to those proposals applicable in datacenters. In short, to our knowledge, no prior work introduces a scale-out design for datacenter multicast.

PortLand [29] supports multicast routing on fat trees, as well as multicast tree reconstruction, but does not address any scalability issues. ESM [25] leverages the hierarchical topology to accelerate the multicast tree computation process, but again, it does not utilize the multi-rooted tree's properties to scale beyond the capacity of a single switch.

Several other proposals seek to increase the number of multicast groups that can be supported by a *single* switch; work that is complementary to our interests in scaling out capacity by leveraging many switches and datacenter topologies. For example, some proposals

[2] There is some risk of terminology confusion here: *address aggregation* corresponds to grouping together like multicast addresses under the same multicast IP prefix, while an *aggregation switch* is the standard term for the upper-layer switch in each pod, as it aggregates connections to the core layer.

C	multicast address capacity of a switch
k	# of ports per switch
n	# of partitions of the multicast address space
r_c	# of core switches with the same partition
r_a	# of aggr switches with the same partition in each pod
r_p	# of pods over which a group spreads on average
r_e	# of edge switches over which a group spreads in each pod on average

Table 1: Key notations and definitions in this paper

use Bloom filters to reduce multicast forwarding states. FRM [35] encodes group addresses into group Bloom filters in routers, while LIPSIN [21] and ESM [25] encode forwarding states into Bloom filters carried in packet headers. The overhead of these approaches arise from the trade-off between Bloom filter sizes and their false-positive rates (which lead to unnecessary multicast transmissions). To keep false-positive rates manageable, such approaches would lead to prohibitively large filters given the number of multicast groups we seek to support.

Dr. Multicast [43] scales multicast by only supporting *real* IP multicast operations to selected addresses that fit within a *single* switch's capacity, and otherwise using iterated unicast by end hosts. This approach can expose large numbers of multicast groups to applications, but sacrifices network bandwidth and server load to do so. Dr. Multicast also aggregates multiple groups with similar members into one group, but again in a manner to fit in a single switch's forwarding table. Such network-wide aggregation is not as scalable or efficient as our pod-based aggregation (see §3.2.2).

3. SYSTEM DESIGN

We now detail our IP multicast mechanisms for datacenters and analyze their scalability and fault tolerance.

3.1 Multicast Address Distribution

Recall that we partition the multicast address space into multiple blocks and assign these partitions to core switches and aggregation switches in each pod. Using such a strategy, the core switches can cooperatively support a much larger number of multicast groups across the entire datacenter. Similarly, each pod's aggregation layer can support a larger number of groups within its pod.

3.1.1 Calculating the Multicast Group Capacity

Before detailing the distribution process for the multicast address space, we briefly analyze the number of multicast groups (or *group capacity*) supported by a fat-tree datacenter network. We assume the partitioning parameters from Table 1.

A 3-tiered fat tree with k-port switches can support a total of $k^3/4$ end hosts, connected by $k^2/4$ core switches and k pods, each with $k/2$ aggregation and $k/2$ edge switches [6].

Let C denote the number of multicast addresses supported by a single switch. If we partition the multicast address space into n blocks with unique prefixes, and assign each partition to r_c core switches ("r" for replication factor), then the maximum number of multicast addresses that can be supported by the core layer is

$$C_{\text{cores}} = \frac{k^2}{4r_c} \cdot C \tag{1}$$

If each partition is assigned to r_a aggregation switches in *each* pod, then the maximum number of multicast addresses supported by the aggregation layer of each pod is

$$C_{\text{aggrs}} = \frac{k}{2r_a} \cdot C \tag{2}$$

Suppose the host members of each group in a pod spread across r_e edge switches *on average*, then the multicast addresses capacity of the edge layer in the pod is

$$C_{\text{edges}} = \frac{k}{2r_e} \cdot C \tag{3}$$

Figure 2 shows an example of multicast address space distribution in a 3-tiered fat tree with 8-port switches, with $n = r_c = 4$, and $r_a = 2$ for all pods. The network can support up to $4C$ groups at the core layer, and each pod's aggregation layer can support up to $2C$ groups.

In practice, for a datacenter network with $k = 48$-port switches, each holding $C = 1000$ multicast addresses, if we set $n = 64$ and $r_c = 9$, then the core layer of the network can support up to $64,000$ multicast addresses. This scale is far beyond the group capacity of any current multicast system. We will further discuss each pod's address capacity in §3.2.

3.1.2 Distributing the Multicast Address

Figure 2 provides an intuition on how to distribute multicast addresses to increase the system's group capacity. Some common steps in multicast address distribution include:

1. Choose the values of n, r_c and r_a. Smaller values of r_c and r_a lead to increased system capacity of multicast addresses, while greater values of r_c and r_a give better fault tolerance and load balance for multicast routing.
2. Partition and allocate the multicast address space, as the connectivity pattern between the pods and core switches affects how to assign address partitions to switches. Each core or aggregation switch with an address space partition should connect to at least one switch with the same partition at the other layer. It only needs to be processed once when the datacenter network is brought online.
3. Assign a multicast address to every new group. The new address should belong to a partition with (1) fewer addresses currently in use and (2) fewer addresses associated with heavy-traffic groups, if necessary.

The concrete algorithm for allocating address space partitions varies for different types of fat trees. Here we list three types of well studied fat trees,[3] with simple examples built with 4-port switches shown in Figure 1.

- The "**Standard**" [6] fat tree (Fig. 1a) uses the same connection pattern in all pods. Distributing multicast addresses in a standard fat tree is very simple. However, it does not perform well when network failures occur [26].
- The **PortLand** [29] fat tree (Fig. 1b) has a different connection pattern for each pod, yielding a more complex network structure. It can handle network failures better, although its multicast address distribution is more complex.
- The **AB** [26] fat tree (Fig. 1c) improves routing fault tolerance. AB fat trees have two different types of pods ("A" and "B"), each with different connection patterns. Its address distribution algorithm is relatively simple: first allocate the address partitions

[3] The 'Standard" and PortLand topologies were not explicitly formulated in their papers, but rather just illustrated (see Fig. 3 in [6] and Fig. 1 in [29]).

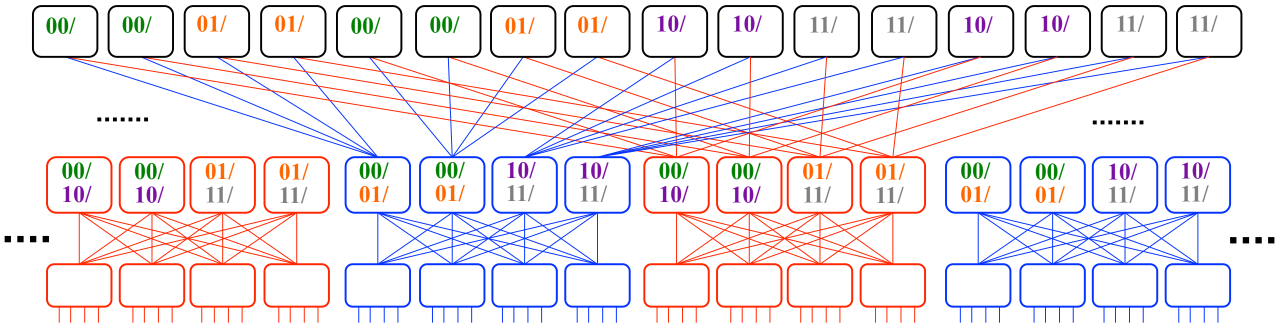


Figure 2: Example of multicast address distribution in a 3-tiered AB fat tree with 8-port switches. Each 2-bit prefix represents a partition. Each switch stores separate forwarding rules for all multicast addresses with associated partition prefixes. Red and blue colors represent the two different connection patterns between pods and core switches.

Algorithm 1 Multicast Address Distribution in AB Fat Trees

```
Require: $n \cdot r_c = k^2/4,\ r_a^2 = r_c,\ 2r_a | k$.
function DISTRIBUTION($k$, $n$, $r_c$, $r_a$)
    $N_p \leftarrow 2nr_a/k$                      ▷ number of partitions in each aggr switch
    $aggrS \leftarrow$ aggr switches of a type-A pod
    for $i_a$ in $[0, k/2)$ do                    ▷ each aggr switch
        $coreS \leftarrow$ core switches connected to $aggrS[i_a]$
        $i_c \leftarrow 0$                         ▷ index of $coreS$
        $R \leftarrow \lfloor i_a/r_a \rfloor$     ▷ the range index of the partitions to be assigned
        for $R \cdot N_p \le i_p < (R+1) \cdot N_p$ do    ▷ assign $N_p$ partitions
            ASSIGN($partition[i_p]$, $aggrS[i_a]$)
            for $0 \le count < r_c/r_a$ do
                ASSIGN($partition[i_p]$, $coreS[i_c]$)
                $i_c \leftarrow i_c + 1$
    for all other pods do
        $aggrS \leftarrow$ aggr switches of the pod
        for $i_a$ in $[0, k/2)$ do                ▷ each aggr switch
            $coreS \leftarrow$ core switches connected to $aggrS[i_a]$
            for $i_c$ in $[0, k/2)$ do
                $i_p \leftarrow$ the partition assigned to $coreS[i_c]$
                ASSIGN($partition[i_p]$, $aggrS[i_a]$)
```

to the core switches and aggregation switches in one pod, then assign each aggregation switch in other pods with all the partitions in the core switches connected to it.

The AB fat tree is both structurally simple and more fault tolerant, so we focus on its use for much of this paper. Algorithm 1 shows an example of address distribution with the given parameters for an AB fat tree. Due to the topological simplicity of AB fat trees, partition assignments made to one pod determine the assignments in all other pods. The algorithm first assigns the address partitions to the aggregation switches in a type-A pod, then to core switches connected to each aggregation switch in that pod accordingly. Finally, each aggregation switch in all other pods is assigned all the partitions of the core switches to which it connects. This algorithm runs only once when the datacenter network is brought online.

Additional optimizations for managing forwarding table space could be explored. For example, multicast addresses for important, popular, and/or long-lived groups can be replicated at more switches. On the other hand, addresses for groups rapidly changing could be replicated at fewer switches, to reduce the switches' update load.With sufficient multicast address replication in both core and aggregation layers, we can apply similar load-aware [7, 36] or load-agnostic [15, 9] flow-scheduling techniques for load balancing and traffic engineering. We leave the further exploration of these optimizations and techniques to future work.

For more general multi-rooted tree networks, C and k in Table 1 may be different for the core, aggregation, and edge switches. In this case, the detailed address distribution scheme needs to be adjusted for specific network settings, but the key approach and its effect on scaling the number of multicast groups remain the same. For example, a datacenter may have fewer core switches, each with more ports and higher multicast address capacity. Then each core switches can be assigned more address partitions, which will not affect the scalability of multicast group numbers of the whole network.

3.2 Scaling Address Aggregation in Pods

Earlier, we showed that the number of multicast groups supported by the aggregation and edge layer in each pod is much lower than that of the core layer (Eq. 1, 2, 3), which results in the core layer being under-utilized, especially if many pods have similar groups. In this section, we analyze the multicast address capacity in the pods, and propose a technique using local address translation and aggregation to further increase the number of groups in each pod.

The bottleneck for a pod's group capacity may arise at either its aggregation or edge layer, depending on how many addresses are stored by each layer's switches. This number depends on both the network configuration and the distribution of end-host's group memberships. Given these two layer's capacities (Eq. 2 and 3), the maximum number of supported multicast groups in a pod is

$$C_{\text{pod}} = \frac{C \cdot k}{2 \cdot \max(r_a, r_e)}\ , \tag{4}$$

which means a pod's group capacity is limited by the *bottleneck layer* in the pod that can support fewer groups.

In this section, we introduce a new scheme that increases the number of groups supported in a pod to the group capacity of the *non-bottleneck layer*, i.e., calculated with the min of (r_a, r_e) in the denominator of Eq. 4. This leads to a significant improvement if the values of r_a and r_e differ greatly.

Let r_p denote the average number of pods over which a group may spread, p_m denote the percentage of groups across multiple pods. In combination with Eq. 1, the maximum number of supported multicast groups of the system is

$$C_{\text{system}} = \min(C_{\text{cores}}/p_m\ ,\ \textstyle\sum_{\text{pod}} C_{\text{pod}}/r_p). \tag{5}$$

3.2.1 Apply Local Address Aggregation

To increase the multicast address capacity in a pod, we introduce an address translation mechanism to enable multicast address aggregation at the *bottleneck layer*. With local aggregation, the bottleneck layer in a pod can support as many groups as the other layer's address capacity.

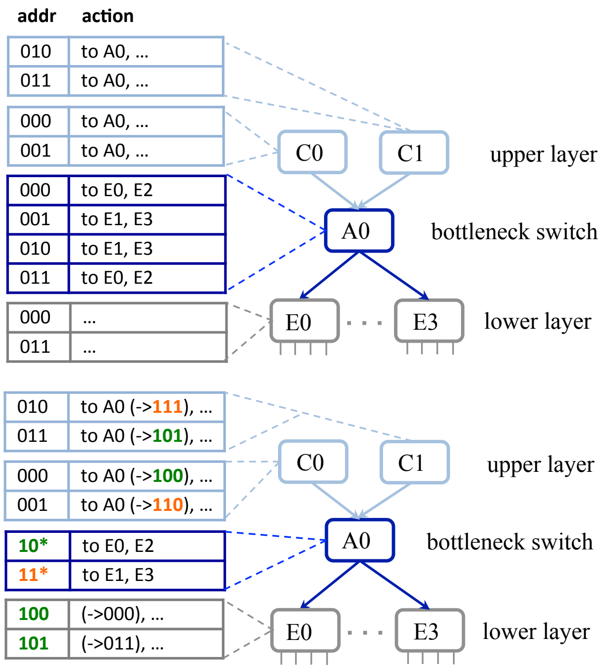


Figure 3: Example of local group aggregation. Action -> signifies rewriting the destination address to that specified. A0's multicast forwarding table has 4 entries initially (up), but 2 entries after local aggregation (bottom).

Suppose the network controller wants a *bottleneck switch* with multicast address capacity C to support $N(> C)$ groups. It can aggregate the N groups into M $(\leq C)$ meta-groups, and assign a unique prefix to each meta-group. The multicast groups in each meta-group have the similar forwarding states in the bottleneck switch, and they are assigned new local addresses that share their meta-group's prefix.

The switches above and below this bottleneck layer maintain a map between the global and local multicast addresses, and forward multicast packets to the bottleneck switch with translated local addresses. Then, the bottleneck switch forwards the multicast packets based on local meta-group prefixes. The lower-layer switches rewrite local addresses back to their original global ones. If such aggregation is applied at an edge switch, this lower-layer functionality must be performed on soft switches running on each end host.

Figure 3 shows an example in which the bottleneck arises at the aggregation switches. As shown, the aggregation switch initially has to keep one entry for each group; after local address translation and aggregation, the aggregation switch only needs to keep two entries for the two meta-groups.

3.2.2 The Multicast Group Aggregation Algorithm

Figure 3 shows an ideal case for group aggregation, where groups—in this example, (`000` and `011`) and (`001` and `010`)—have multicast receivers at the identical edge switches. Thus, these groups have identical forwarding state in higher-layer switches. In practice, however, multicast groups in the same meta-group usually do not have exactly the same forwarding state, which results in lower-layer switches or hosts receiving traffic from multicast groups to which they do not subscribe.

Therefore, we want an aggregation algorithm that minimizes unnecessary network load, or *cost*. The general *group aggregation problem* (sometimes called the *channelization problem*) is NP-complete [5], although various heuristic solutions have been proposed [5, 40, 43]. Compared to prior work, however, our solution:

1. Scales much better due to its decomposition into many smaller *independent subproblems*, since groups only need to be aggregated locally to increase the bottleneck's capacity. Each one only has $k/2$ possible outgoing ports, and at most $C \cdot r_a$ or $C \cdot r_e/r_a$ groups to be aggregated. Both numbers are much smaller than those in traditional problems. This reduces the network and computational overhead by orders of magnitudes. For a datacenter with *distributed network controllers*, each controller only needs to compute the local aggregation for switches in its subnet.

   On the contrary, past projects tried to aggregate all multicast groups across the entire network, where the number of possible members (VMs) across all groups may reach hundreds of thousands or more. This scale makes groups with similar members much less common, and the computation very slow.

2. Can be applied with a more realistic model. While traditional group aggregation methods assume routing paths to all receivers have a same linear cost function, our model is able to handle the real-world link cost function, which could be nonlinear and convex because additional traffic increasingly degrades performance when links become *congested*. This is much more computationally expensive for traditional approaches that operate on the whole network.

Algorithm for group aggregation. To design an effective algorithm for local group aggregation, we first model it as a characterized *channelization problem* [5], and introduce a practical optimization objective function. Suppose we have following initial sets for a bottleneck switch:

- A set of downlink ports P, where $|P| = k/2$.
- A set of multicast groups G.
- A set of meta-groups M, where $|M| \leq C$.

For $g \in G$, $p \in P$, $m \in M$, define three types of sets:

- $P_g = \{p \mid p \text{ subscribes to } g\}$
- $G_m = \{g \mid g \text{ is assigned to } m\}$
- $M_p = \{m \mid p \text{ subscribes to } m\} = \{m \mid \exists\, g \in G_m \wedge p \in P_g\}$

Let λ_g denote the traffic rate of group g, $F(.)$ denote the congestion cost function on a downlink of the bottleneck switch, and μ_p denote the background traffic (e.g., unicast) rate over the downlink of port p. Then given input sets P_g for each g, we want to find optimal aggregation sets G_m and M_p to minimize the total congestion cost (Φ) of all links after local aggregation:

$$\Phi(G,M) = \sum\nolimits_p F\left(\sum\nolimits_{m\in M_p}\left(\sum\nolimits_{g\in G_m}\lambda_g\right)+\mu_p\right) \qquad (6)$$

The optimization problem is NP-hard, so we provide an heuristic method to greedily assign values G_m under the constraints, while keeping the value of the optimization objective cost function Φ as low as possible. Algorithm 2 shows the core part of our heuristic method. $\textsc{Cost}(g,m)$ computes the increase in Φ if group g is assigned to meta-group m, and $\delta(g,m)$ computes the cost threshold to create a new meta-group. The running time of the algorithm is $O(|G| \cdot |M| \cdot |P|)$, which is equal to $O(k \cdot C \cdot |G|)$.

To handle churn in the group membership (P_g) and changes in the traffic rates (λ, μ), we track cost changes on groups or meta-groups, and then recompute the aggregation if necessary. The track function can be either run periodically, or it can be triggered by certain group membership or traffic rate changes.

Algorithm 2 Multicast Group Local Aggregation

```
function AGGREGATION($X$)
    $\tilde{M} \leftarrow M - \{m_0, \ldots, m_{|P|}\}$        ▷ initial set of empty meta-groups
    for $g \in G$ do
        if $|P_g| \geq |P| - \Delta$ then
            ASSIGN($g$, $m_{|P|}$)        ▷ $m_{|P|}$ forwards to all ports
        else if $|P_g| = 1$ then
            $p \leftarrow p \in P_g$
            ASSIGN($g$, $m_p$)        ▷ $m_p$ only forwards to port $p$
        else
            $\acute{m} \leftarrow \arg\min_{m \in \tilde{M}}$ COST($g$, $m$)        ▷ minimal extra cost
            if $|M| > 0$ and COST($g, \acute{m}$) $> \delta(g, \acute{m})$ then
                $m \leftarrow \tilde{M}$.pop()        ▷ create new meta-group
                ASSIGN($g$, $m$)
            else
                ASSIGN($g$, $\acute{m}$)

function ASSIGN($g, m$)        ▷ assign group $g$ to meta-group $m$
    update $G_m$, $M_p$, $\Phi(G, M)$ and other auxiliary data
```

3.3 Recovering Quickly from Failures

Network failures in datacenters may significantly impact network performance, especially for multicast traffic, since reconstructing the multicast tree can be an expensive process. A recent study [14] shows that many of the failures in datacenter networks are short term, however, so immediately reconstructing the multicast tree can be inefficient and unnecessary.

A better approach may be to have switches locally reroute the packets immediately after a failure, and let the network controller compute the new multicast tree in the background if necessary. Therefore, our fault-tolerant routing solution consists of two parts: (1) short-term local rerouting for fast failover and (2) long-term multicast tree reconstruction.

3.3.1 Using Local Rerouting for Fast Failover

Unicast local rerouting has been well studied by Liu et al. [26] for AB fat trees (Figure 1c and 2), from which we base our design for multicast rerouting. Here, we focus on the unique problems of rerouting multicast traffic, without limiting our solutions to any specific types of fat tree.

Upstream rerouting (flows from end hosts up to aggregation or core switches) is easy to support in our design. Because all multicast address partitions are assigned to multiple aggregation switches in each pod and more core switches, each edge or aggregation switch has multiple uplinks for all its multicast addresses. Thus, if one uplink fails, the switch only needs to forward the packets to other upper-layer switches that are assigned with the associated partitions.

Downstream rerouting (flows from core or aggregation switches down to end hosts) is more complex. Liu et al. [26] proposed two downstream rerouting schemes for core switches in an AB fat tree. Let $coreS_x$ denote the x-th core switch, and $aggrS_{i.a}$ and $edgeS_{i.a}$ denote the a-th aggregation and edge switch in the i-th pod. Suppose $coreS_x$'s downlink to $aggrS_{i.a}$ fails, in most cases, the packets can still reach the i-th pod by a *three-hop rerouting* through an aggregation switch in other pods (e.g., $coreS_x \rightarrow aggrS_{j.c} \rightarrow coreS_y \rightarrow aggrS_{i.b}$). In less common cases, the packets can reach the i-th pod by a *five-hop rerouting*, which involves both aggregation and edge switches in other pods (e.g., $coreS_x \rightarrow aggrS_{j.c} \rightarrow edgeS_{j.d} \rightarrow aggrS_{j.e} \rightarrow coreS_y \rightarrow aggrS_{i.b}$).

However, multicast rerouting cannot directly use such techniques. When a core switch receives rerouted multicast packets, it would otherwise forward them out all ports listed in its forwarding table's group entry. This behavior adds network overhead, as only the pod experiencing the failure needs the rerouted packets. To make local rerouting effective, we add a location identifier to the rerouted packets (specifying either a destination pod or edge switch), allowing other switches to know the true destination of rerouted packets.

Core → aggregation route failure. When a core switch notices that its downlink to one pod fails, it seeks to reroute all multicast packets to that pod through other pods. To avoid forwarding rerouted multicast packets to unwanted pods, it adds a *pod identifier* to the header of the rerouted packets. Other switches can then quickly determine to which pod these packets should be sent, and they forward them via unicast routing accordingly.

Aggregation → edge route failure. When an aggregation switch detects a downlink failure, it needs to reroute the packets through other edge and aggregation switches in the same pod. The process is similar as above. When sending a rerouted packet, the aggregation switch adds an *edge identifier* to the packet header. Then other edge and aggregation switches can tell to which edge switch they should forward upon receiving the rerouted packet.

The location identifiers can be implemented in various ways. For example, if using VLAN tags, a switch pushes a VLAN tag representing the correct destination when sending rerouted packets to other pods or edge switches. The switch in the correct destination then pops the VLAN tag before further processing. An alternative is to utilize some unused bits in the multicast address, e.g., these bits could be set to zero by default, and switches could then rewrite them to specify certain destinations. That said, the current OpenFlow *specification* only supports setting the entire address to some specified constant, not selected bits.

Edge identifiers can be reused in different pods. Thus, a datacenter built with k-port switches only needs $3k/2$ VLAN tags or $\lceil \log_2(3k/2) \rceil$ unused bits in the header, and each edge and aggregation switch needs $k/2$ and $3k/2$ unicast rules, respectively, to support this multicast rerouting protocol.

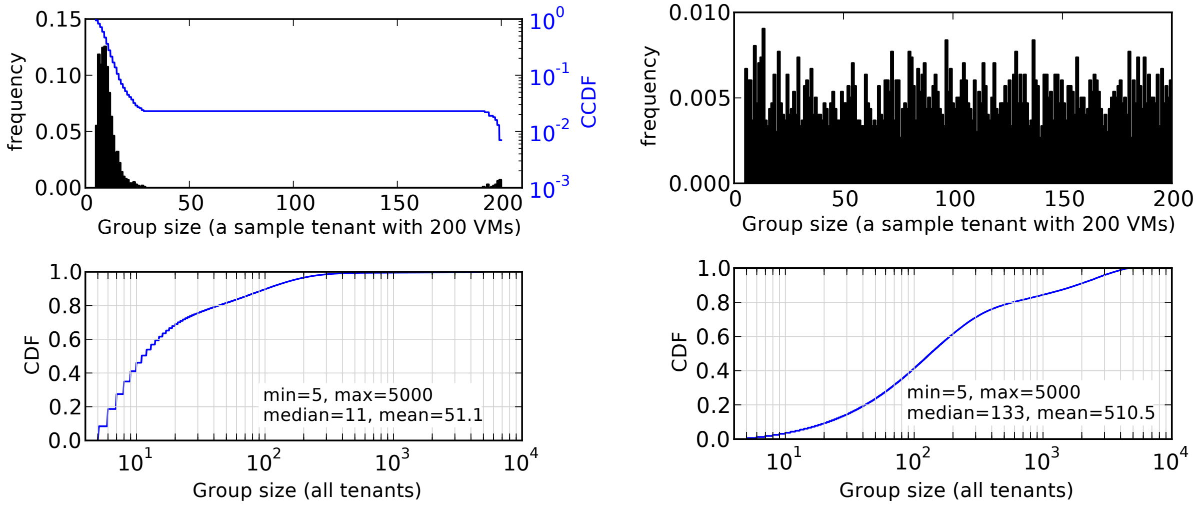
OpenFlow support. There are two possible approaches to local rerouting with OpenFlow-compliant switches. First, the switch can include local backup rules, so that the switch will immediately figure out what to do by itself after a failure happens. However, the current OpenFlow specification [32] only supports backup rules for unicast forwarding. Second, the network controller can be informed about every network failure and then install the local rerouting rule to associated switches.

3.3.2 Reconstructing the Multicast Tree

When long-term failures happen, we eventually want to rebuild the multicast tree for affected multicast groups. However, with fast failover to local rerouting, we can allow some reasonable delay until multicast tree reconstruction, and thus lessen the computational burden on network controllers.

After failure, if at least one core switch can still reach all the end hosts belonging to the group, we can reconstruct the multicast tree by changing the root core switch for corresponding groups.

If multiple failures happen, it is possible that no single core switch can reach all group members, as the symmetry of the network's multi-rooted tree structure can be disrupted. In such cases, the packets of the group have to be forwarded to multiple core switches during upstream routing, and some pods may need more than one aggregation switches to receive downstream packets. We can reconstruct the multicast tree by solving a set cover problem for the set of hosts and switches that are associated with the multicast group [29].



(a) Group size follows WVE distribution in each tenant.

(b) Group size follows uniform distribution in each tenant.

Figure 4: Two types of group size distributions in the datacenter.

4. EVALUATION

This section evaluates the datacenter's multicast address distribution and failover behavior through a series of simulation experiments. Our evaluation answers the following questions:

- How do multicast address distribution and local aggregation increase the number of supported groups?
- Can the network controller and switches handle common multicast group dynamics?
- How do network failures affect multicast routing?

Because our primary technical focus is on multicast *scalability*, we simulate full-sized datacenter topologies, rather than implement our protocols on a small testbed. Even so, it is difficult to run packet-level network simulation at such scale; further, we are not focused on network throughput or latency. Instead, we first construct the entire fat-tree network topology in simulation, then generate multicast groups with different distributions, and statistically compute the network's group capacity and the performance of multicast routing failover (in terms of route stretch and availability). We focus on multi-tenant environments where each tenant runs multiple multicast applications across its VMs, although our findings also apply to settings eschewing virtualization.

As discussed in Section 2.5, our work is the first *scale-out* datacenter multicast solution. Prior solutions were all ultimately bottlenecked by the address capacity of each switch, which is orders-of-magnitude smaller than our datacenter solution. Thus, we omit further detailed comparison of our protocol with previous work.[4]

4.1 Experimental Setup

We simulate a 3-tiered AB fat-tree datacenter network built with 48-port switches, which connects 27,648 physical end hosts. We split the multicast address space into 64 partitions, each of which is replicated at 9 core switches and 3 aggregation switches per pod.

In our experiments, the datacenter network is populated by 3000 tenants. The number of VMs per tenant follows an exponential distribution, with $min = 10$, $median = 100$, and $max = 5000$. Each physical end host has at most 20 VMs.

We assume a tenant's VMs do not share the same physical host for reasons of resilience and load balance, and then evaluate two types of VMs placement policies: (i) a tenant's VMs are placed on hosts near to one another, and (ii) a tenant's VMs are distributed across the network uniformly at random.

We then assign multicast groups to each tenant. Each tenant is assigned to some number of groups roughly proportional to the tenant's size. Each tenant's group sizes—the number of VMs belonging to that group—follow a similar distribution, although scaled by the tenant's size. Each group's members in each tenant are picked uniformly at random from among all VMs of the tenant. We set the minimum group size to five, and evaluate two different distributions.

We generate the first group-size distribution by analyzing a trace of multicast patterns from IBM WebSphere Virtual Enterprise (WVE), which has 127 nodes and 1364 groups [19]. In this trace, most groups have only a small set of members, while a few have nearly all nodes as members. We model this WVE distribution and scale it to each tenant's size. Figure 4a shows the WVE distribution of group sizes (for a particularly-sized tenant), as well the as the size distribution across all tenants. While the average group size is 51, nearly 80% of groups have fewer than 40 members, while about 0.4% of groups have more than 2000 members.

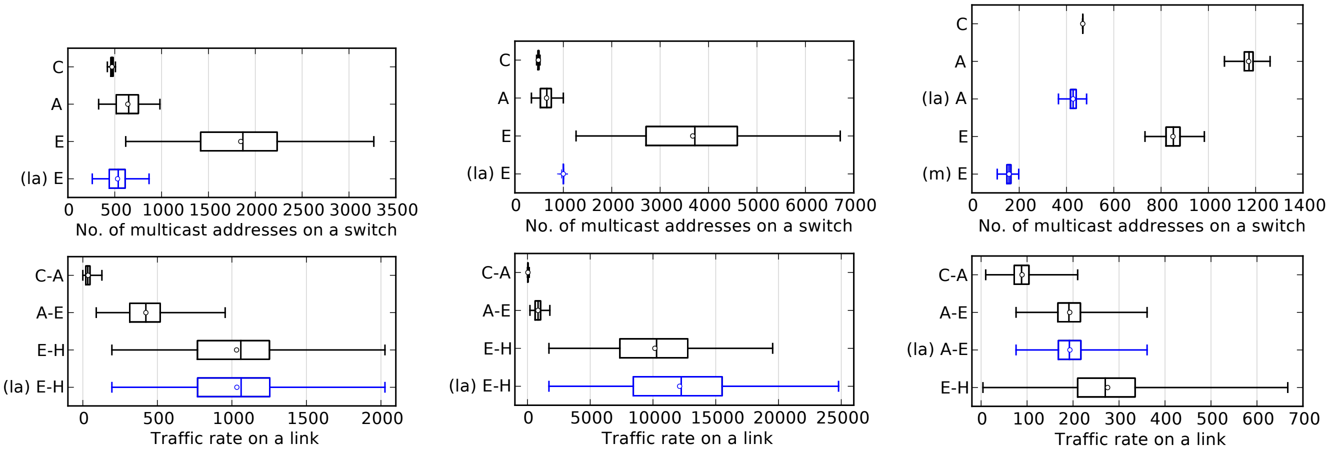
We generate the second distribution by setting each tenant's group sizes to be uniformly distributed between the minimum and entire tenant size. Figure 4b shows such a distribution, where the average group size is 510, which is 10 times larger than that of the WVE distribution.

We also tested different network sizes, tenant size distributions, and numbers of VMs per host. The key factors to scalability are the size of multicast groups and the distribution of group members decided by these settings. More multicast groups can be supported when the network is larger, when the average group size is smaller, and when the group members are closer to each other.

4.2 Multicast Address Capacity

We first evaluate how much the datacenter's multicast address capacity can be improved by multicast address space partition and local aggregation. We have shown that with fixed network settings,

[4] We attempted at first to construct a "scale-out" version of Dr. Multicast [43], but found ourselves recreating the algorithms introduced in this paper, as confirmed by Dr. Multicast's author [42].



(a) 100K groups. Group size follows WVE distribution in each tenant. A tenant's VMs are placed on nearby hosts. Apply local aggregation at the edge layer.

(b) 100K groups. Group size follows uniform distribution in each tenant. A tenant's VMs are placed on nearby hosts. Apply local aggregation at the edge layer.

(c) 30K groups. Group size follows WVE distribution in each tenant. A tenant's VMs are distributed randomly. Apply local aggregation at the aggr layer.

Figure 5: Simulation results of the no. of multicast entries in switches and link rates. *C* stands for core switch, *A* for aggr switch, *E* for edge switch, *H* for host, (la) for local aggregation, (m) for entries with multiple forwarding ports. Whiskers represent the min and max value; boxes show the 25th, 50th (midline), and 75th percentiles; circles show the means.

the datacenter's multicast address capacity is greatly affected by group distributions, and it is proportional to the multicast address capacity of a single switch (Eq. 4 and 5). We conduct our simulations with different tenants and multicast group distributions.

The traffic rate of each group is randomly chosen from 1 to 10. We evaluate the traffic rate overhead for local group aggregation based on this setting. The rate unit is not specified since it does not affect simulation results in these experiments. For most experiments, we set each switch to have a multicast address capacity of 1000 entries, although we also evaluate scalability under different capacities (Figure 6).

When a tenant's VMs are placed on nearby hosts, local aggregation significantly increases group capacity. With this VM placement policy, most multicast groups are spread over a very small number of pods ($max = 12$, $mean = 2.5$ in our simulation). As a result, the group capacity bottleneck should arise at the edge layer.

Figure 5a shows the simulation results for 100K multicast groups, with group sizes for each tenant following the WVE distribution (Fig. 4a). Each core and aggregation switch have less than 1000 multicast addresses. However, without local aggregation, more than 95% of edge switches have more than 1000 multicast addresses, and the maximum number reaches 330% of the switch capacity, which means the datacenter can support up to about 30K groups. If we apply local aggregation to the edge switches, the multicast entries on edge switches are reduced to 866 at maximum and 530 on average, and thus can support 100K multicast groups. The traffic overhead introduced by local aggregation is only about 0.2%.

Figure 5b shows the simulation results for 100K multicast groups, with group sizes of each tenant following uniform distribution (Fig. 4b). The number of multicast addresses in core and aggregation switches are about the same as the previous distribution, while the edge switches have many more multicast addresses. Without local aggregation, the maximum number of multicast entries in an edge switch is 700% of the switch capacity, so the datacenter can only support up to 15K multicast groups. By applying local aggregation on the edge switches, however, the datacenter can support all 100K multicast groups, which allows each end host to subscribe to 1850 different groups on average. Since the group sizes increase greatly, the traffic overhead introduced by local aggregation now becomes about 19.4% on average, and about 24.5% for 1% of the links with the highest traffic rate. However, this overhead is still very small compared to non-IP-multicast solutions (considering the average group size is 510).

When a tenant's VMs are distributed across the network randomly, the use of unicast entries combined with local aggregation significantly increases group capacity. With this VM placement policy, groups may spread over many pods ($max = 48$, $mean = 16$ in our simulation), but only to a small number of edge switches within each pod ($mean = 2.2$). As a result, the group capacity bottleneck should arise at the aggregation layer.

Random VM placement greatly reduces the multicast address capacity of the whole system, since addresses are replicated at more switches and more pods. Figure 5c shows the simulation results for 30K groups, with group sizes of each tenant following the WVE distribution. Each core and edge switch has fewer than 1000 multicast addresses, but each aggregation switch has more than 1000 multicast addresses. Applying local aggregation to the aggregation layer reduces the number of entries in each aggregation switch to fewer than 500. Because the maximum number of multicast entries in edge switches is near to 1000, in this configuration, the datacenter can support up to about 30K multicast groups, fewer than that shown in Figures 5a and 5b.

With random VM placement, members of the same multicast group have a low probability of being connected to the same edge switch. As a result, most multicast entries in the edge switches only have one forwarding port, which could be handled by a unicast entry. In other words, an edge switch can reduce its multicast forwarding table size by only allocating address entries with multiple forwarding ports in the multicast forwarding table. Figure 5c (top) shows that there are at most only 200 multicast address entries in each edge switch that have multiple forwarding ports, shown in $(m)E$ in the figure. Therefore, with the same group distribution in this example, the datacenter can support up to 150K groups by using about 4K more unicast entries in each edge switch.

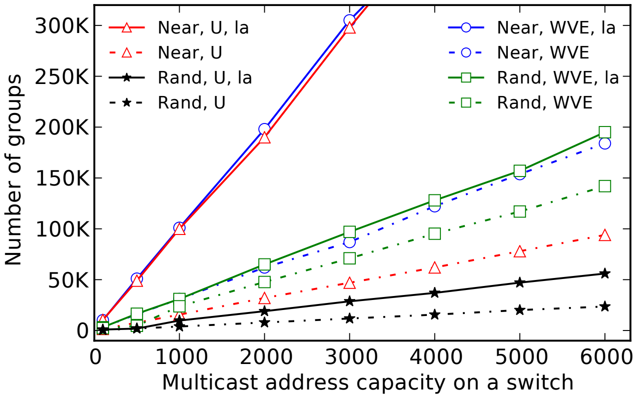


Figure 6: A datacenter's maximum number of supported multicast groups with different switch settings and group distributions. *Near* and *Rand* stand for a tenant's VMs are placed on *nearby* hosts or *randomly*, *WVE* and *U* for group size follows *WVE* or *uniform* distribution in each tenant, *la* for values after local aggregation. *y* axis is cut at 300K. The two *Near+la* lines continue to near-linearly increase as the switch address capacity increases to 6000.

A datacenter's multicast group capacity is linearly related to the switch's multicast address capacity. We now vary switches' address capacity and test the datacenter's corresponding group capacity. To do so, we continuously generate multicast groups with same distributions, until at least one switch reaches its capacity. Figure 6 shows the results. Generally, the number of supported groups is higher when a tenant's VMs are placed on hosts near to another, because a same address would be replicated on fewer switches; and when the group size follows the WVE distribution in each tenant, because the average group size is much smaller (1/10th) than that of uniform distribution.

We also simulate different replication factors (r_c, r_a) for multicast address partitions. The results are fairly obvious: smaller replication factors lead to more supported multicast groups. For example, when a tenant's VMs are placed on hosts near to another and the group sizes follow the WVE distribution, by setting $r_a = 2$ rather than 3 in each pod, the datacenter can support about 50% more groups (e.g., 150K when the single switch's address capacity is 1000).

4.3 Group Membership Dynamics

In this section, we analyze and evaluate how frequently the switches would update their multicast forwarding tables when (i) a new group is created or a host joins an existing group, and (ii) a group is removed or a host member leaves a group. Once a dynamic event happens, the associated datacenter network controller will get informed and then decide whether to install or remove rules in the relevant switches.

Generally, a group member change will always trigger an edge switch update, but will only trigger updates at higher layers if it is the first join or last leave of this group at the lower layer. If a multicast group has multiple join or leave events during a short period of time, we can reduce the number of updates on switches with *batch* operations. There are two types of batch operation effects: first, collocated leaves and joins at about the same time may lead to zero update on upper level switches; second, burst joins or leaves only need one round of updates on related switches.

We next evaluate the benefit of batch operations, first using an example to analyze the number of switch updates per dynamic event within one group (§4.3.1), and then evaluate the group membership dynamics for all multicast groups in the datacenter (§4.3.2).

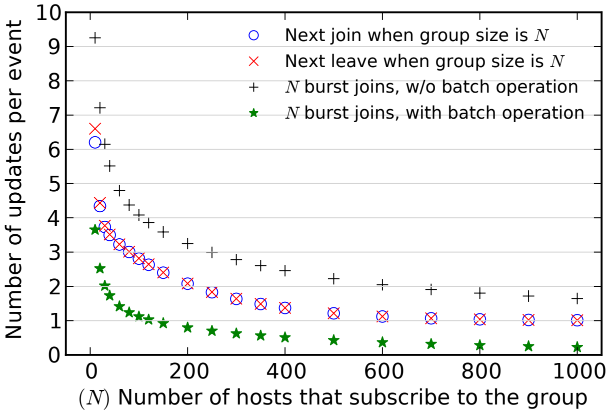


Figure 7: Average number of switch updates per join or leave event, for multicast groups in a tenant whose VMs spread over all 4608 end hosts in 8 pods.

	Join	Leave
number of events	500004	499996
average no. of edge updates per event	1	1
average no. of aggr updates per event	0.75	0.74
average no. of core updates per event	0.01	0.01
average no. of total updates per event	1.76	1.75

(a) A tenant's VMs are placed on hosts near to one another.

	Join	Leave
number of events	149923	150077
average no. of edge updates per event	1	1
average no. of aggr updates per event	1.89	1.86
average no. of core updates per event	1.64	1.55
average no. of total updates per event	4.53	4.41

(b) A tenant's VMs are distributed across the network randomly.

Table 2: Number of switch updates for dynamic events

4.3.1 *Membership Dynamics within One Group*

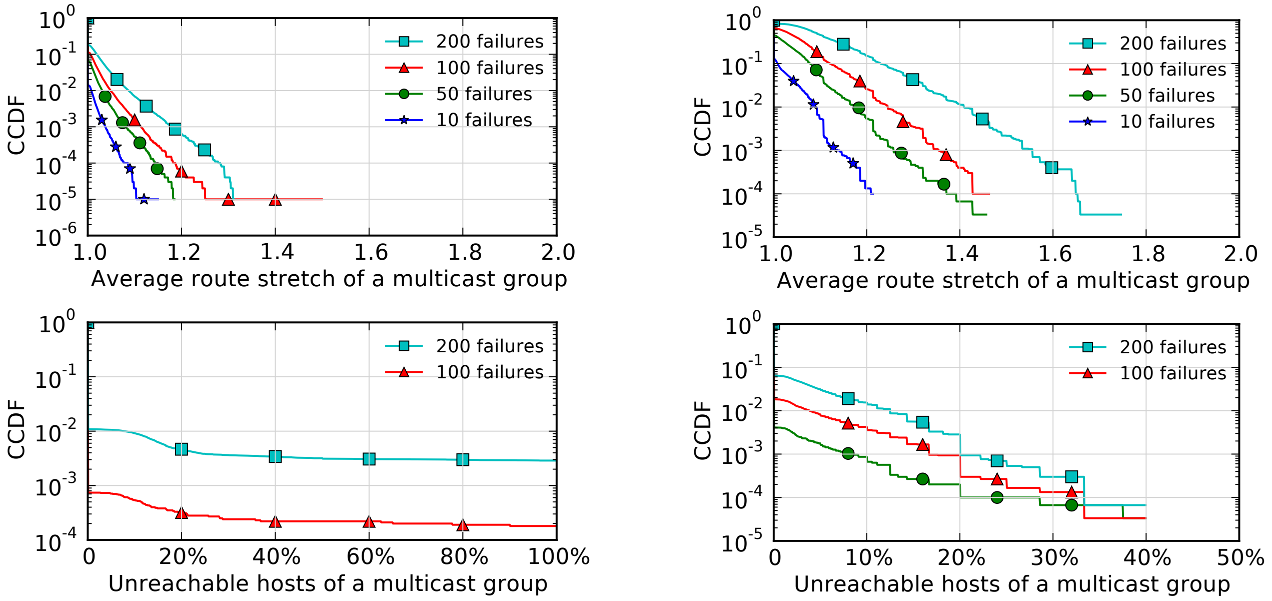
Suppose a tenant's VMs are spread over all 192 edge switches and 4608 end hosts in 8 pods. A multicast group *G* of the tenant experiences three steps of group membership churn:

1. *G* is created, with no end host members.
2. *N* random hosts join *G* at about the same time.
3. End hosts randomly join or leave *G*.

We assume all the hosts who do not subscribe to *G* have equal probability to join, and all the hosts who subscribe to *G* have equal probability to leave. Figure 7 shows the average number of switch updates for group *G*, across different sizes for *G*. For example, with 300 burst joins at the beginning, the average number of updates per join is 2.78 without batching, yet only 0.62 with batching. With a current group size is 300, the average number of switch updates for the next join is 1.63, and the number for the next leave is 1.64.

4.3.2 *Membership Dynamics of All Groups*

For large scale evaluation, we simulate multicast membership dynamics with the network and tenants settings specified in §4.1.



(a) A tenant's VMs are placed on hosts near to one another

(b) A tenant's VMs are distributed across the network randomly

Figure 8: Route stretch and percentage of unreachable hosts per group when using fast failover with local rerouting, when some random aggregation switches among the 1152 aggregation switches in 48 pods of the network fail. All hosts are reachable when only a small number of switches fail (e.g., 50 failures when each tenant's VMs are near to one another). Group size follows WVE distribution in each tenant.

Group size follows the WVE distribution in each tenant. Join and leave events are generated randomly for each group. The number of events for a group is proportional to the group size. Table 2a shows the results for 1 million join/leave events for 100K multicast groups in the network where a tenant's VMs are placed on hosts near to one another. Table 2b shows the results for 300K join/leave events for 30K multicast groups in the network where a tenant's VMs are distributed across the network randomly. The number of switch updates are small on average, especially when a tenant's VMs are placed on hosts near to one another.

We can use this analysis to calculate the update load on switches. Suppose each pod has 1000 dynamic events per second, then (in the worse case with Table 2b) each edge switch has 42 updates per second on average, each aggregation switch has 78 updates, and each core switch has 133 updates. Given that (per §2.4), a commodity network controller can handle more than a million requests per second, and today's OpenFlow switches can handle up to 1000 updates per second, these update rates can be easily handled even without batching.

4.4 Fast Failover

If an edge switch fails, all hosts connected to it would lose their connection to the network, and can do nothing but wait for their switch to be fixed. If a core switch fails, aggregation switches can just redirect upstream flows to other cores, without increasing the routing stretch.

The fast failover protocols are most interesting when aggregation switches fail. So, we simulate the network with random aggregation switch failures and compute the route stretch and percentage of unreachable hosts. Figure 8 shows our findings. Most multicast groups have very low route stretch and high routing availability with local rerouting, even if 5% of the switches fail. This indicates that fast failover with local multicast rerouting can work well for short-term network failures, and it can be used to provide additional time for multicast tree reconstruction by the network controller.

5. CONCLUSION

Multicast is an important communication primitive in datacenter networks. Providing multicast at the network layer achieves better bandwidth and computational efficiency than emulating it within applications or overlays. However, the use of IP multicast has been traditionally curtailed due to scalability limitations.

This paper overcomes these traditional limits by leveraging the structural properties of multi-rooted tree topologies and the capabilities of a centralized network management platform emerging in today's datacenters. Our architecture can support large numbers of multicast groups even given modest multicast forwarding tables, and its mechanisms are robust to network failures. The system is transparent to applications and deployable in today's networks.

Acknowledgments. The authors are grateful to Jennifer Rexford for her insightful feedback, as well as to Ymir Vigfusson for providing the IBM WVE traces and valuable suggestions. Matvey Arye, Aaron Blankstein, Michael Chan, Xin Jin, Rob Kiefer, Wyatt Lloyd, Ariel Rabkin, Peng Sun, the anonymous CoNEXT reviewers, and our shepherd, Dejan Kostic, provided helpful comments. This work was supported by funding from the National Science Foundation.

References

[1] Cisco's Massively Scalable Data Center. `http://www.cisco.com/en/US/solutions/ns340/ns414/ns742/ns743/ns994/landing_msdc.html`.

[2] IETF Multicast Security Working Group (Concluded). `http://datatracker.ietf.org/wg/msec/`.

[3] IETF Reliable Multicast Transport Working Group (Concluded). `http://datatracker.ietf.org/wg/rmt/`.

[4] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly. Symbiotic Routing in Future Data Centers. In *SIGCOMM*, 2010.

[5] M. Adler, Z. Ge, J. F. Kurose, D. Towsley, and S. Zabele. Channelization Problem in Large Scale Data Dissemination. In *ICNP*, 2001.

[6] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM*, 2008.

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.

[8] M. Balakrishnan, K. Birman, A. Phanishayee, and S. Pleisch. Ricochet: Lateral Error Correction for Time-Critical Multicast. In *NSDI*, 2007.

[9] Y. Cai, L. Wei, H. Ou, Y. Arya, and S. Jethwani. Protocol Independent Multicast Equal-Cost Multipath (ECMP) Redirect. RFC 6754, 2012.

[10] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376, 2002.

[11] C. Diot, B. Neil, L. Bryan, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14:78–88, 2000.

[12] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised). RFC 4601, 2006.

[13] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. *Trans. Networking*, 5(6):784–803, Dec. 1997.

[14] P. Gill, J. Navendu, and N. Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *SIGCOMM*, 2011.

[15] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandular, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, 2009.

[16] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: a High Performance, Server-centric Network Architecture for Modular Data Centers. In *SIGCOMM*, 2009.

[17] M. Handley, I. Kouvelas, T. Speakman, and L. Vicisano. Bidirectional Protocol Independent Multicast (BIDIR-PIM). RFC 5015, 2007.

[18] H. Holbrook and B. Cain. Source-Specific Multicast for IP. RFC 4607, 2006.

[19] IBM WebSphere. `www-01.ibm.com/software/webservers/appserv/was/`.

[20] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a Globally-Deployed Software Defined Wan. In *SIGCOMM*, 2013.

[21] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: Line Speed Publish/Subscribe Inter-Networking. In *SIGCOMM*, 2009.

[22] P. Judge and M. Ammar. Security Issues and Solutions in Multicast Content Distribution: A Survey. *IEEE Network*, 17:30–36, 2003.

[23] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *STOC*, 1997.

[24] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutevski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A Distributed Control Platform for Large-scale Production Networks. In *OSDI*, 2010.

[25] D. Li, Y. Li, J. Wu, S. Yu, and J. Yu. ESM: Efficient and Scalable Data Center Multicast Routing. *Trans. Networking*, 20(3):944–955, 2012.

[26] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson. F10: A Fault-Tolerant Engineered Network. In *NSDI*, 2013.

[27] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. IETF Internet-Draft, May 2013.

[28] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30:817–840, 2004.

[29] R. N. 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*, 2009.

[30] D. Newman. 10 Gig access switches: Not just packet-pushers anymore. *Network World*, 25(12), Mar. 2008.

[31] Object Management Group. Data Distribution Service. `http://portals.omg.org/dds/`.

[32] Open Networking Foundation. `www.opennetworking.org`.

[33] Oracle Coherence. `http://coherence.oracle.com/display/COH35UG/Network+Protocols`.

[34] I. Pepelnjak. FIB update challenges in OpenFlow networks. `blog.ioshints.info/2012/01/fib-update-challenges-in-openflow.html`, Jan. 2012.

[35] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP Multicast. In *SIGCOMM*, 2006.

[36] S. Sen, D. Shue, S. Ihm, and M. J. Freedman. Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing. In *CoNEXT*, 2013.

[37] J.-Y. Shin, B. Wong, and E. G. Sirer. Small-World Datacenters. In *SOCC*, 2011.

[38] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking Data Centers Randomly. In *NSDI*, 2012.

[39] M. Sridharan, A. Greenberg, Y. Wang, P. Garg, N. Venkataramiah, K. Duda, I. Ganga, G. Lin, M. Pearson, P. Thaler, and C. Tumuluri. NVGRE: Network Virtualization using Generic Routing Encapsulation. IETF Internet-Draft, Aug. 2013.

[40] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchial Clustering of Message Flows in a Multicast Data Dissemination System. In *IASTED PDCS*, 2005.

[41] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On Controller Performance in Software-Defined Networks. In *HotICE*, 2012.

[42] Y. Vigfusson. Personal communication, 2013.

[43] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, R. Burgess, G. Chockler, H. Li, and Y. Tock. Dr. Multicast: Rx for Data Center Communication Scalability. In *EuroSys*, 2010.

[44] J. Widmer and M. Handley. Extending Equation-based Congestion Control to Multicast Applications. In *SIGCOMM*, 2001.