ESM: Efficient and Scalable Data Center Multicast Routing

Dan Li, *Member, IEEE*, Yuanjie Li, Jianping Wu, *Senior Member, IEEE*, Sen Su, and Jiangwei Yu

***Abstract*—Multicast benefits group communications in saving network traffic and improving application throughput, both of which are important for data center applications. However, the technical trend of data center design poses new challenges for efficient and scalable multicast routing. First, the densely connected networks make traditional receiver-driven multicast routing protocols inefficient in multicast tree formation. Second, it is quite difficult for the low-end switches widely used in data centers to hold the routing entries of massive multicast groups. In this paper, we propose ESM, an efficient and scalable multicast routing scheme for data center networks. ESM addresses the challenges above by exploiting the feature of modern data center networks. Based on the regular topology of data centers, ESM uses a source-to-receiver expansion approach to build efficient multicast trees, excluding many unnecessary intermediate switches used in receiver-driven multicast routing. For scalable multicast routing, ESM combines both in-packet Bloom Filters and in-switch entries to make the tradeoff between the number of multicast groups supported and the additional bandwidth overhead. Simulations show that ESM saves 40%∼50% network traffic and doubles the application throughputs compared to receiver-driven multicast routing, and the combination routing scheme significantly reduces the number of in-switch entries required. We implement ESM on a Linux platform. The experimental results further demonstrate that ESM can well support online tree building for large-scale groups with churns, and the overhead of the combination forwarding engine is light-weighted.**

***Index Terms*—Data center network, multicast, routing.**

I. Introduction

GROUP communication widely exists in data centers hosting cloud computing [5], [20]. Multicast benefits group communications by both saving network traffic and improving application throughput. Though network-level multicast in the Internet bears a notorious reputation during the past two decades for deploying obstacles and for many open issues such as congestion control, pricing model, and security concern, recently there is a noticeable resurgence of it, e.g., the successful application of IP multicast in IPTV networks [6], enterprise networks, etc. The managed environment of data centers also provides a good opportunity for multicast deployment. However, existing multicast protocols built in data center switches/servers are primarily based on the multicast design for the Internet. Before the wide application of multicast in data centers, we need to carefully investigate whether these Internet-oriented multicast technologies can well accommodate data center networks.

In this paper, we explore network-level multicast routing, which is responsible for building the multicast delivery tree, in modern data center networks. Bandwidth-hungry, large-scale data center applications call for efficient and scalable multicast routing schemes. However, we find the technical trend of modern data center design poses new challenges to achieve the goals.

First, data center topologies usually expose high link density. For example, in BCube [7], both switches and servers have multiple ports for interconnection, while in Fat-Tree [8] and VL2 [9], several levels of switches with tens of ports are used to connect the large population of servers. There are many equal-cost paths between a pair of servers or switches. In this scenario, multicast trees formed by traditional independent receiver-driven multicast routing can result in severe link waste compared to efficient ones.

Second, low-end commodity switches are widely used in most data center designs for economic and scalability considerations. The memory space of the routing table in them is relatively narrow. Previous investigation shows that typical access switches can hold no more than 1500 multicast group states [12]. Besides, it is difficult to aggregate in-switch multicast routing entries since the multicast address embeds no topological information. Hence, it is quite challenging to support a large number of multicast groups in data center networks, especially considering the massive groups in file chunk replications to embrace.

In this paper, we propose ESM, an efficient and scalable multicast routing scheme for data center networks. ESM addresses the challenges above by exploiting the features of modern data center networks. ESM depends on a multicast manager for multicast routing management, leveraging the controllable environment of data centers. By exploiting the multistage graph of data center topologies, the multicast manager builds the multicast tree via a source-to-receiver expansion approach. This approach overcomes the problem in receiver-driven multicast routing since it excludes many unnecessary intermediate switches in the formed multicast tree. Given dynamic receiver

Manuscript received February 17, 2011; revised August 09, 2011; accepted September 19, 2011; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor L. Andrew. Date of publication October 17, 2011; date of current version June 12, 2012. This work was supported by the National Basic Research Program of China (973 Program) under Grants 2011CB302900 and 2009CB320501 and the National Natural Science Foundation of China under Grants 61170291, 61133006, and 61120106008.

D. Li is with Tsinghua University, Beijing 100084, China, and also with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: lidan@csnet1.cs.tsinghua.edu.cn).

Y. Li, J. Wu, and J. Yu are with Tsinghua University, Beijing 100084, China.

S. Su is with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2011.2169985

join/leave, tree links can be gracefully added/removed without reforming the whole tree, in favor of avoiding out-of-order packets during multicast delivery.

In-packet Bloom Filter eliminates the necessity of in-switch routing entries, which helps achieve scalable multicast routing upon low-end switches in data centers. However, both the in-packet Bloom Filter field and the traffic leakage during packet forwarding result in bandwidth waste. To make the tradeoff, ESM combines both in-packet Bloom Filter and in-switch entries to realize scalable multicast routing in data center networks. Specifically, in-packet Bloom Filters are used for small-sized groups to save routing space in switches, while routing entries are installed into switches for large groups to alleviate the bandwidth overhead. The combination approach well aligns with the fact that small groups are most common in data center group communications, represented by file chunk replication in distributed file systems. For the in-packet Bloom Filter routing, ESM employs node-based Bloom Filter to encode the tree and uses a provably loop-free forwarding scheme.

We have implemented a software-based ESM on a Linux platform. We conduct both simulations and experiments to study the performance of ESM. The simulation results show that the tree-building approach in ESM saves 40%∼50% network traffic and doubles the application throughputs compared to receiver-driven multicast routing, and the combination routing scheme significantly reduces the number of in-switch entries required. Experiments on a testbed show that ESM can well support online tree building for large-scale groups with churns, and the combination forwarding engine brings less than 4% CPU overhead at high-speed data delivery.

II. Background and Design Rationale

In this section, we discuss the background of data center multicast and design rationale.

A. Data Center Multicast

One-to-many group communication is common in modern data centers running cloud-based applications. Multicast is the natural technology to benefit this kind of communication pattern, for the purposes of both saving network bandwidth and reducing the load on the sender. For Web search services, the incoming user query is directed to a set of indexing servers to look up the matching documents [4]. Multicast can help accelerate the directing process and reduce the response time. Distributed file system is widely used in data centers, such as GFS [2] in Google, HDFS in Hadoop, and COSMOS in Microsoft. Files are divided into many fix-sized chunks, say, 64 or 100 MB. Each chunk is replicated to several copies and stored in servers located in different racks to improve the reliability. Chunk replication is usually bandwidth-hungry, and multicast-based replication can save the interrack bandwidth. In map-reduce like cooperative computations [3], the executive binary is delivered to the servers participating in the computation task before execution. Multicast can also speed up the binary delivery and reduce the task finish time.

Though multicast is supported by most network devices (routers, switches) and end-hosts, it is not widely deployed in the Internet due to many technological causes, such as the pricing model, multicast congestion control, and security concerns. However, we argue that in the managed environment of data centers, multicast is a natural choice to support data center group communication. For instance, the multicast pricing problem is not an issue in data centers since a data center network is usually used and managed by a single authority, the data center servers are considered trustworthy, etc. However, current multicast protocols implemented at switches and servers are primarily Internet-oriented. Before the wide deployment of multicast in data center networks, we need to carefully investigate whether these multicast protocols can well embrace the data center environment. We focus on multicast routing in this paper, in particular the efficiency and scalability of multicast routing. For efficiency, we seek to build multicast trees occupying as few links to save network bandwidth. For scalability, we aim to support a great number of multicast groups.

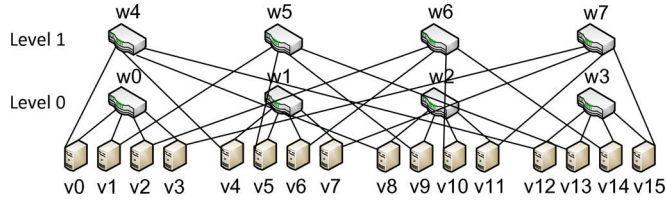


Fig. 1. BCube(4,1) architecture.

B. Data Center Network Architecture

In current practice, data center servers are connected by a tree hierarchy of Ethernet switches, with commodity ones at the first level and increasingly larger and more expensive ones at higher levels. It is well known that this kind of tree structure suffers from many problems. The top-level switches are the bandwidth bottleneck, and high-end high-speed switches have to be used. Moreover, a high-level switch shows as a single-point failure spot for its subtree branch. Using redundant switches does not fundamentally solve the problem, but incurs even higher cost. Recently, there is a growing interest in the community to design new data center network architectures with high bisection bandwidth to replace the tree structure [7]–[11]. BCube and Fat-Tree are the representative ones among these architectures.

BCube: BCube is a server-centric interconnection topology, which targets for shipping-container-sized data centers, typically sized of 1 k–4 k servers. BCube is constructed in a recursive way. A BCube$(n, 0)$ is simply n servers connecting to an n-port switch. A BCube$(n, 1)$ is constructed from n BCube$(n, 0)$'s and n n-port switches. More generally, a BCubek$(n, k)(k \geq 1)$ is constructed from n BCube$(n, k-1)$'s and n^k n-port switches. Each server in a BCube(n, k) has $k + 1$ ports. Fig. 1 illustrates a BCube(4,1) topology, which is composed of 16 servers and two levels of switches.

Fat-Tree: Fig. 2 illustrates the topology of Fat-Tree, which organizes the switches in three levels. There are n pods ($n = 4$ in the example), each containing two levels of $n/2$ switches, i.e., the edge level and the aggregation level. Each n-port switch at the edge level uses $n/2$ ports to connect the $n/2$ servers, and uses the remaining $n/2$ ports to connect the $n/2$ aggregation-level switches in the pod. At the core level, there are $(n/2)^2$ n-port switches, and each switch has one port

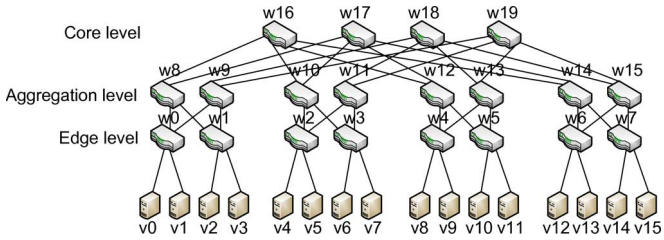


Fig. 2. Fat-Tree architecture with 4-port switches.

connecting to one pod. Therefore, the total number of servers supported in Fat-Tree is $n^3/4$.

In spite of different interconnection structures, consistent themes lie in recently proposed data center network architectures. First, low-end switches are used for server interconnection instead of high-end expensive ones. Second, high link density exists in these networks since a large number of switches with tens of ports are used, or both servers and switches use multiple ports for interconnection. Third, the data center structure is built in a hierarchical and regular way on the large population of servers/switches.

C. Design Rationale

Modern data center network architectures pose new challenges in the design of efficient and scalable multicast routing. First, the redundant links in data center networks make traditional receiver-driven multicast routing protocols inefficient in terms of the number of links of the formed multicast tree. Second, the routing table memory space in low-end commodity switches is quite limited, and it is challenging to support a large number of multicast groups.

In densely connected data center networks, there are often a large number of tree candidates for a group. Given multiple equal-cost paths between servers/switches, it is undesirable to run traditional receiver-driven multicast routing protocols such as PIM for tree building because independent path selection by receivers can result in many unnecessary intermediate links. Without loss of generality, we take the BCube topology as an example, which is shown in Fig. 1. Assume the receiver set is $\{v5, v6, v9, v10\}$ and the sender is $v0$. If using receiver-driven multicast routing, the resultant multicast tree can have 14 links as follows (we represent the tree as the paths from the sender to each receiver):

$$v0 \rightarrow w0 \rightarrow v1 \rightarrow w5 \rightarrow v5$$
$$v0 \rightarrow w4 \rightarrow v4 \rightarrow w1 \rightarrow v6$$
$$v0 \rightarrow w4 \rightarrow v8 \rightarrow w2 \rightarrow v9$$
$$v0 \rightarrow w0 \rightarrow v2 \rightarrow w6 \rightarrow v10.$$

However, an efficient multicast tree for this case includes only nine links if we construct in the following way:

$$v0 \rightarrow w0 \rightarrow v1 \rightarrow w5 \rightarrow v5$$
$$v0 \rightarrow w0 \rightarrow v2 \rightarrow w6 \rightarrow v6$$
$$v0 \rightarrow w0 \rightarrow v1 \rightarrow w5 \rightarrow v9$$
$$v0 \rightarrow w0 \rightarrow v2 \rightarrow w6 \rightarrow v10.$$

The space of the expensive and power-hungry fast memory in the low-end data center switches, such as SRAM, is usually narrow to control the economical cost, especially considering the link speeds in data centers are increasing rapidly [19], [20]. Hence, it is challenging to maintain a potentially large number of routing entries in the small memory space. Compared to unicast, it is especially difficult to aggregate multicast routing entries due to two reasons. First, multicast addresses are logical identifiers without any topological information. Second, each multicast forwarding entry maintains an outgoing interface set rather than a single outgoing interface, so it is less probable for different forwarding entries holding common forwarding rules. Previous investigation showed that typical access switches can contain only 70 $\sim$1500 multicast group states [5].

To address the challenges above, we design ESM, a novel multicast routing scheme in data center networks, by leveraging the managed environment of data centers, the topological characteristics of modern data center networks, as well as the multicast group size distribution pattern.

First, we assume a multicast manager exists in data centers for multicast routing management, such as allocating the multicast addresses, building the multicast trees, distributing the routing configurations to switches/servers, etc. This kind of centralized controller is widely adopted in modern data center design. For instance, in Fat-Tree [8], a fabric manager is responsible for managing the network fabric. In VL2 [9], a number of directory servers are used to map the AA-LA relationship. The emerging OpenFlow [25] framework also uses a controller for routing rule decision and distribution. ESM depends on the controller to calculate the multicast tree, configure the multicast routing rules on switches, and send the tree information to the source server if in-packet Bloom-Filter-based multicast routing is used. Note that, as we show in Section VI, a single commodity machine usually works well for online tree building in ESM in most data centers. If the multicast manager is serving for much larger data center networks, a cluster of servers can work together to play the role, such as the directory servers in VL2 [9].

Second, we exploit the regular data center network topology for multicast routing design. We will show later that the classical Steiner-tree algorithm is too slow when the network size is large. To online build the multicast tree based on the group membership, ESM leverages the hierarchical, regular data center topology to accelerate the tree calculation process. In addition, we depend on the topological characteristic of data center networks to design a loop-free Bloom-Filter-based multicast forwarding engine in ESM.

Third, by observing that small-sized groups are dominant in data centers and large-sized groups bring significant bandwidth waste in in-packet Bloom-Filter-based multicast routing, we use a combination routing scheme to achieve scalable multicast routing in ESM. For the limited number of large groups, multicast routing entries are installed in switches to avoid traffic leakage. As for the large volume of small groups, in-packet Bloom Filter is used to eliminate the requirement of in-switch routing table space.

III. Efficient Multicast Tree Building

Efficient multicast trees are required to save network traffic and accordingly reduce the task finish time of data center

applications. In this section, we present how to build multicast trees in ESM.

A. Problem

To eliminate the unnecessary switches used in independent receiver-driven multicast routing, ESM builds the multicast tree in a managed way. The group join/leave requests on receivers are directed to the multicast manager. The multicast manager then calculates the multicast tree based on the data center topology and group membership distribution. Data center topologies are regular graphs, and the multicast manager can easily maintain the topology information (with failure management).

The problem is then translated as how to calculate an efficient multicast tree on the multicast manager. Note that building the Steiner tree in general graphs is NP-Hard. For recently proposed data center topologies, the multicast tree building in Fat-Tree is polynomial (the optimal multicast tree is determined by randomly selecting a core-level switch). However, we prove that the Steiner tree problem in BCube is also NP-Hard.

Theorem 1: The Steiner tree problem in the BCube network is NP-Hard.

Proof: Please refer to [28]. ■

ESM targets at building efficient multicast trees in generic data center networks. Although there are heuristic algorithms to solve the Steiner tree problem, we will show in Section VI that the computation complexity of this kind of algorithm is too high to meet the requirement of online tree building for large groups. Consequently, we develop an approximate algorithm by exploiting the topological feature of modern data center networks. Next, we present the tree-building approach in ESM.

B. Source-Driven Tree Building

We observe that recently proposed data center architectures (represented by BCube and Fat-Tree) use several levels of switches for server interconnection, and switches within the same level are not directly connected. Hence, they are multistage graphs [22]. From the feature of multistage graphs, all the possible paths from the multicast source to all receivers in a group can be expanded as a directed multistage graph with $d+1$ stages (from stage 0 to stage d), where d is the topology of the network. We call it *group spanning graph* for the multicast group. Stage 0 includes the sender only, and stage d is only composed of receivers. Note that any node appears at most once in the group spanning graph. For example, if the sender is $v0$ and the receiver set is $\{v1, v5, v9, v10, v11, v12, v14\}$, the group spanning graph in BCube topology of Fig. 1 is shown in Fig. 3(a). For the same multicast group in Fat-Tree of Fig. 2, the group spanning graph is shown in Fig. 3(b).

We make some definitions on the group spanning graph.

A Covers B (or B Is Covered by A): For any two node sets A and B in a group spanning graph, we say A *covers* B (or B is *covered* by A) if and only if for each node $j \in B$, there exists a directed path from a node $i \in A$ to j in the group spanning graph.

A Strictly Covers B (or B Is Strictly Covered by A): If A covers B and no subset of A covers B, we say A *strictly covers* B (or B is *strictly covered* by A).

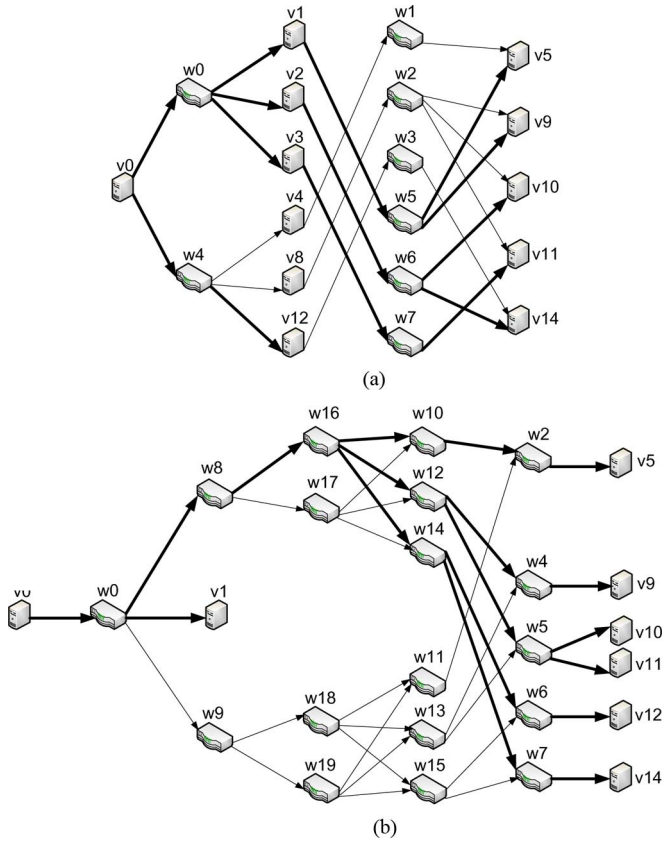


Fig. 3. Exemplified group spanning graphs in (a) BCube and (b) Fat-Tree. The sender is $v0$, and the receiver set is $\{v1, v5, v9, v10, v11, v12, v14\}$. The corresponding data center topologies are shown in Figs. 1 and 2, respectively. The bold links illustrate the tree formed upon the group spanning graph.

We propose to build a multicast tree in a source-to-receiver expansion way upon the group spanning graph, with the tree node set from each stage *strictly covering* downstream receivers. Multicast trees built by our approach hold two merits. First, many unnecessary intermediate switches used in receiver-driven multicast routing are eliminated. Second, the source-to-receiver latency is bounded by the number of stages of the group spanning graph, e.g., the diameter of data center topology, which favors delay-sensitive applications such as redirecting search queries to indexing servers.

The complexity of the algorithm primarily comes from how to select the node set from each stage of the group spanning graph, which is covered by the node set in the previous stage and strictly covers downstream receivers. Generally speaking, it is an NP-Hard problem. However, we can leverage the hierarchically constructed regular data center topologies to design approximate algorithms. In what follows, we discuss the tree-building algorithm based on the group spanning graph in BCube and Fat-Tree, respectively.

BCube: The tree node selection in a BCube network can be conducted in an iterative way on the group spanning graph. For a BCube(n, k) with the sender s, we first select the set of servers from stage 2 of the group spanning graph, which is covered by both s and a single switch in stage 1. Assume the server set in stage 2 is E, and the switch selected in stage 1 is W. The tree node set for BCube(n, k) is the union of the tree node sets for $|E|+1$ BCube$(n, k-1)$'s. $|E|$ of the BCube$(n, k-1)$'s has a

server in E as the source p, and the receivers in stage $2*(k+1)$ covered by p as the receiver set. The other BCube$(n, k-1)$ has s as the source and the receivers in stage $2*k$, which are covered by s while not covered by W, as the receiver set. In the same way, we can further get the tree node set in each BCube$(n, k-1)$ by dividing it into several BCube$(n, k-2)$'s. The process iterates until getting all the BCube$(n, 0)$'s. Hence, the computation complexity is $O(N)$, where N is the total number of servers in BCube.

Fat-Tree: There are at most 6 hops in a group spanning graph of Fat-Tree since the network diameter is 6. The tree node selection approach for each stage is as follows. From the first stage to the stage of core-level switches, any single path can be chosen because any single core-level switch can cover the downstream receivers. From the stage of core-level switches to the final stage of receivers, the paths are fixed due to the interconnection rule in Fat-Tree. The computation complexity is also $O(N)$.

C. Dynamical Receiver Join/Leave

Given multicast receivers dynamically join or leave a group, the multicast tree should be rebuilt to encompass group dynamics. ESM can gracefully embrace this case because the dynamical receiver join/leave does not change the source-to-end paths of other receivers in the group. It is important for avoiding out-of-order packets during packet delivery.

BCube: When a new receiver r_j joins an existing group in a BCube(n, k), we first extend the previous group spanning graph to involve r_j. Then, in the new group spanning graph, we check whether there is a BCube$(n, 0)$ in the previous multicast tree that can contain r_j. If so, we add r_j to the BCube$(n, 0)$. Otherwise, we try to find the BCube$(n, 1)$ in the previous multicast tree that can hold r_j and add a BCube$(n, 0)$ containing r_j to the BCube$(n, 1)$. If we cannot find such BCube$(n, 1)$, we try to find a BCube$(n, 2)$ and add a corresponding BCube$(n, 1)$ and so on and so forth until we successfully add r_j in the multicast tree. In this way, the final tree obeys the tree-building algorithm in ESM, and we do not need to change the source-to-end paths for existing receivers in the multicast group.

Given a receiver r_l leaves the group in a BCube(n, k), we at first regenerate the group spanning graph by eliminating r_l. Then, if the deletion of r_l results in zero BCube$(n, m-1)$'s in a BCube(n, m) in the previous multicast tree, we also eliminate all the nodes in the BCube(n, m). Of course, this process does not change the source-to-end paths for other receivers, either.

Fat-Tree: If a receiver r_j joins a group in Fat-Tree, it should be added at the final stage of the group spanning graph. In the new tree, we need to do nothing except adding the path from the previously selected core-level switch to r_j. Note that the paths from the core-level switch to r_j are fixed, and they may overlap with some links in the previous multicast tree. Similarly, when a receiver r_l leaves a group, we delete it from the final stage of the group spanning graph. In addition, we exclude the tree links with zero downstream receiver due to the removal of r_l. In this way, the receiver join/leave in Fat-Tree does not affect the source-to-end paths for other receivers in the multicast session.

IV. Scalable Multicast Routing

A desirable multicast routing scheme not only builds an efficient multicast tree for each group, but also scales to a large number of multicast groups. In this section, we discuss how to realize scalable routing in ESM.

A. Combination Routing Scheme

It is challenging to support massive multicast groups using the traditional way of installing all the multicast routing entries into switches, especially on the low-end commodity switches commonly used in data centers. In-packet Bloom Filter is an alternative choice since it eliminates the requirement of in-switch entries. However, the in-packet Bloom Filter can cause bandwidth waste during multicast forwarding.

Bandwidth Overhead Ratio: The bandwidth waste of in-packet Bloom Filter comes from two aspects. First, the Bloom Filter field in the packet brings network bandwidth cost. Second, false-positive forwarding by Bloom Filter causes traffic leakage. Moreover, switches receiving packets by false-positive forwarding may further forward packets to other switches, incurring not only additional traffic leakage, but also possible loops. We define the *bandwidth overhead ratio* of in-packet Bloom Filter, r, as the ratio of additional traffic caused by in-packet Bloom Filter over the actual payload traffic to carry. Assume the packet length (including the Bloom Filter field) is p, the length of the in-packet Bloom Filter field is f, the number of links in the multicast tree is t, and the number of actual links covered by Bloom Filter based forwarding is c. Then, r is calculated as

$$r = \frac{t*f + (c-t)*p}{t*(p-f)}. \tag{1}$$

In (1), $t*f$ is the additional traffic in the multicast tree resulting from the addition of the Bloom Filter field in the packet, $(c-t)*p$ is the total traffic carried by the links beyond the tree, and $t*(p-f)$ is the actual payload traffic on the tree. To reduce the bandwidth overhead of in-packet Bloom Filter, we need to either control the false positive ratio during packet forwarding or limit the size of Bloom Filter field. However, the two goals are in conflict with each other: Given a certain group size, reducing the false positive ratio indicates enlarging the Bloom Filter field, and vice versa.

We investigate the bandwidth overhead ratio of Bloom-Filter-based routing against the Bloom Filter length. BCube network and Fat-Tree network are studied respectively. Given the group size, the group members are randomly selected from all the data center servers. Note that in either BCube or Fat-Tree, there are often multiple equal-cost trees for a given group. We always choose the one with the minimum bandwidth overhead ratio. The total packet size is set as 1500 B, which is the typical MTU in Ethernet.

We fix the network size as BCube(8,3) and Fat-Tree with 48-port switches, choose the number of hash functions minimizing the false positive ratio, and measure the bandwidth overhead ratio for different group sizes along with the growth of the in-packet Bloom Filter length. The result is shown in Fig. 4. From this figure, we have the following observations. First, for a given group size, there is an "optimal" length of the in-packet Bloom Filter that minimizes the bandwidth overhead ratio. When the Bloom Filter length is shorter than the optimal value, false-positive forwarding is the major factor for bandwidth overhead ratio. However, when the length grows larger than

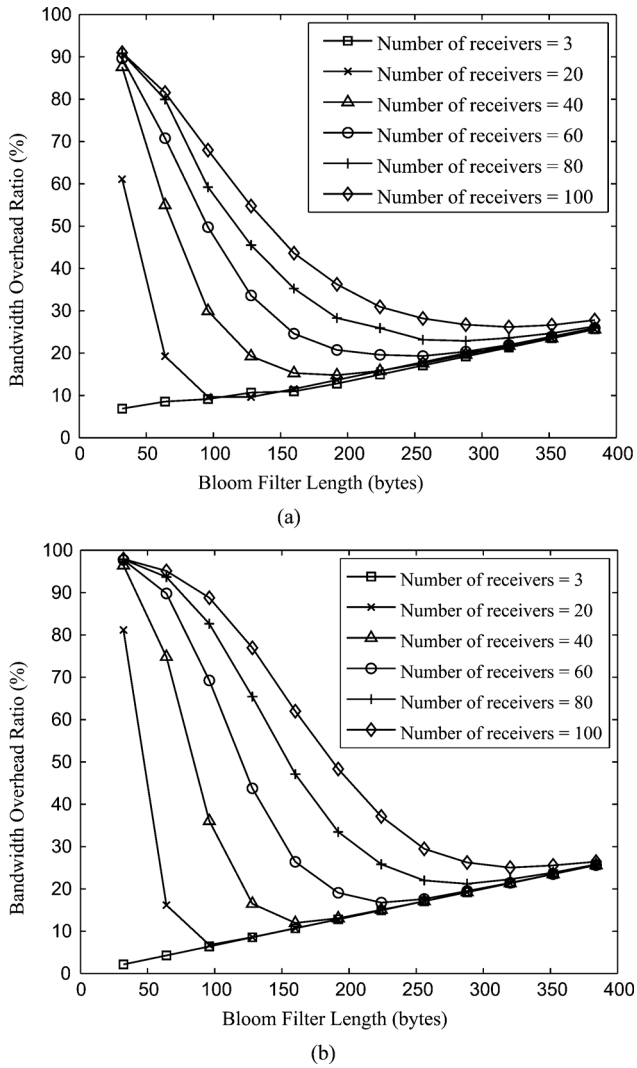


Fig. 4. Bandwidth overhead ratio against group size and in-packet Bloom Filter length. Network size: BCube(8,3), Fat-Tree with 48-port switches. Number of Hash functions: the one minimizing the bandwidth overhead ratio (a) BCube (b) Fat-Tree.

the optimal value, the Bloom Filter field itself dominates the bandwidth overhead. Second, for a certain length of in-packet Bloom Filter, the bandwidth overhead ratio increases with the growth of group size. It is straightforward because more elements in the Bloom Filter result in higher false positives during packet forwarding.

We find that when the group size is larger than 100, the minimum bandwidth overhead ratio becomes higher than 25%. Hence, in-packet Bloom Filter causes significant bandwidth overhead for large groups in both BCube and Fat-Tree. Though there are proposed schemes to reduce the traffic leakage during Bloom Filter forwarding, such as using multiple identifiers for a single link or a virtual identifier for several links [21], they cannot fundamentally solve the problem in data center networks where the link density is high and the server population is huge.

Group Size Pattern: We further investigate the group size distribution in typical data center applications generating group communication patterns. In distributed file systems [1], [2], large files are divided into chunks with 64 or 100 MB. In a data center with Petabytes of files (such as the Hadoop cluster in Facebook [23]), there can be as many as billions of small groups for chunk replication if all these files are put into distributed file system. However, for large groups such as binary distribution to thousands of servers in Map-Reduce computations, the number is quite limited because too many concurrent Map-Reduce computations defer the task finish time. As a result, we envision that small groups are the most common in typical data center applications.

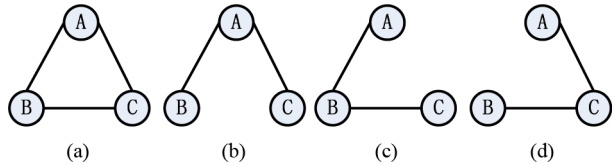


Fig. 5. Example to show the tree ambiguity problem in node-based Bloom Filters. (a) Topology. (b)–(d) Three possible multicast trees for (a).

Our Approach: Hence, we propose to combine in-packet Bloom Filters with in-switch routing entries for scalable multicast routing in ESM. Specifically, in-packet Bloom Filters are used for small-sized groups to save routing space in switches, while routing entries are installed into switches for large groups to alleviate the bandwidth overhead. A predefined group size, z, is used to differentiate the two types of routing schemes. The value of z is globally set to accommodate the routing space on data center switches and the actual group size distribution among data center applications. In this way, our combination routing approach well aligns with the group size pattern of data center applications.

Intermediate switches/servers receiving the multicast packet check a special TAG in the packet to determine whether to forward the packet via in-packet Bloom Filter or looking up the in-switch forwarding table. Before implementing the combination forwarding engine, we explore two important design issues of in-packet Bloom Filter, i.e., tree encoding scheme and loop avoidance.

B. Tree Encoding Scheme

There are two ways to encode the tree information with in-packet Bloom Filter, i.e., node-based encoding and link-based encoding. In node-based Bloom Filters, elements are the tree nodes, including both switches and servers, while in link-based Bloom Filters, elements are the directed physical links.

In general graphs, ambiguity may exist in node-based Bloom Filter on encoding the tree structure. Fig. 5 shows a simple example. Fig. 5(a) shows the topology of the graph. Assume the multicast sender is node A, and nodes B and C are the two receivers. There are three possible multicast trees for the group. The first tree includes link $A \rightarrow B$ and link $A \rightarrow C$, shown in Fig. 5(b). The second one includes link $A \rightarrow B$ and link $B \rightarrow C$, shown in Fig. 5(c). The third one is composed of link $A \rightarrow C$ and link $C \rightarrow B$, shown in Fig. 5(d). However, if using node-based encoding, the Bloom Filter is the same for all three trees. The problem is obvious: When node B (or C) receives the packet from node A, it has no idea whether to forward the packet to node C (or B) or not. We call this problem the *tree ambiguity problem*.

Link-based Bloom Filters do not have the tree ambiguity problem because each tree link is explicitly identified. Hence, in LIPSIN [21], link-based encoding is used instead of node-based encoding. However, the tree ambiguity problem for node-based encoding does not exist if the topology and multicast tree satisfy certain requirements.

Theorem 2: Given a tree T built on top of a topology G, the tree ambiguity problem for node-based encoding can be avoided given that, except for the links from T, there are no other links in G directly connecting any two nodes in T.

Proof: Please refer to [28]. ■

Interestingly, we find that recently proposed data center networks, including BCube and Fat-Tree, and the tree-building algorithm in ESM presented in Section III can well meet the requirement in Theorem 2 (please refer to [28]). As a result, both node-based encoding and link-based encoding can be used in the in-packet Bloom Filter of ESM. We choose node-based encoding since it can ride on the identifiers of servers/switches such as MAC address or IP address for node identification.

C. Loop-Free Forwarding

A potential problem with in-packet Bloom Filter is that false-positive forwarding may cause loops, no matter in node-based encoding or link-based encoding. Take Fig. 2 as the example. Assume there is a multicast group originated from server $v0$, and switches $W8$, $W16$, $W10$ are in the multicast tree. Now, $W8$ forwards the packet to $W17$ false positively. $W17$ checks that $W10$ is in the Bloom Filter and then forwards it to $W10$. After $W10$ receives the packet from $W17$, it finds $W16$ in the Bloom Filter, so it further forwards it to $W16$. Then, $W16$ forwards it to $W8$ since $W8$ is also in the Bloom Filter. In this way, the packet is forwarded in the path of $W8 \rightarrow W17 \rightarrow W10 \rightarrow W16 \rightarrow W8$, and the loop forms.

We design a *distance*-based Bloom Filter forwarding scheme in ESM to solve this problem. We define the *distance* between two nodes as the number of their shortest-path hops on the data center topology. When a node j receives a multicast packet with in-packet Bloom Filter originated from server s, j only forwards the packet to its neighboring nodes (within the Bloom Filter) whose distances to s are larger than j itself. The only requirement on the switch forwarding engine to support the distance-based loop-free forwarding is that it maintains the whole network topology (without failure). The forwarding decision can be made based solely on the network topology and the incoming port of a packet. Note that the data center topology is regular, and it is natural to assume each switch knows the topology information.

Theorem 3: The distance-based Bloom Filter forwarding scheme is both correct and loop-free.

Proof: Please refer to [28]. ■

In either BCube or Fat-Tree, it is quite easy for each switch or server to determine whether its neighboring nodes are closer or farther from the multicast source than itself. Let us still check the scenario above. When node $W8$ falsely forwards the packet to $W17$ and $W17$ forwards it to $W10$, $W10$ will not send the packet back to $W16$ because $W16$ is closer to the source $v0$ than $W10$ itself. In this way, loop is effectively avoided.

V. Simulations

In this section, we evaluate the tree-building algorithm and the combination routing scheme in ESM by simulations.

A. Simulation Setup

We use a BCube(8,3) (4096 servers in total) and a Fat-Tree with 48-port switches (27 648 servers in total) as the representative data center topologies in the simulation. The speed of all links is 1 Gb/s. One thousand groups are generated for each network. For any group, the sender/receivers are randomly selected from all the data center servers. Each group joins the network and leaves at random time. The simulation runs until all the group sessions end.

We choose two group size distribution patterns. One is uniform distribution between $[3, N]$, where N is the total number of servers. The other is power-law distribution based on the assumption that smaller groups are more common in data centers than larger groups. More formally, for power-law distribution, there is $y = x^{\alpha}$, where x is the group size between $[3, N]$ and y is the number of groups. We set $\alpha = -1$ in the simulation.

B. Tree-Building Algorithm

We evaluate the tree-building approach of ESM in two aspects. First, we compare the number of links in the formed trees by our algorithm, typical Steiner-tree algorithm, and receiver-driven multicast routing, respectively. Second, we compare the throughputs of the multicast sessions under the three tree-formation algorithms. For both the metrics, we also take unicast routing as a benchmark, which demonstrates the benefits of multicast in data centers.

Number of Tree Links: We first compare the number of links occupied by ESM, Steiner-tree algorithm, and receiver-driven multicast routing, respectively. The Steiner-tree algorithm we choose is the one described in [14], whose benefit is computation speed.[1] The algorithm works as follows. At first, a virtual complete graph with cost is generated upon the group members. Then, a minimum spanning tree is calculated on the virtual complete graph. Finally, the virtual link in the virtual complete graph is replaced by the shortest path between any two group members in the original topology, with unnecessary links deleted. In the receiver-driven multicast routing, each receiver independently selects a path to join the source-rooted tree. In case of multiple equal-cost paths, a random one is chosen. We also use unicast as a benchmark.

Figs. 6 and 7 show the cumulative distribution function (CDF) of groups occupying different numbers of links in BCube and Fat-Tree networks, respectively. The figures suggest that in both BCube and Fat-Tree, ESM has similar performance with the Steiner-tree algorithm, saving 40%$\sim$50% links than receiver-driven multicast routing. It follows our expectation since independent receiver-driven path selection results in many unnecessary intermediate switches in densely connected network environment.

Then, we compare the number of tree links between ESM and the Steiner-tree algorithm. Fig. 6 shows that there is a little difference for the two algorithms in BCube network. When the group size is small, the Steiner-tree algorithm is better because,

[1]There are algorithms with better approximation ratio but higher computation complexity, such as [15].

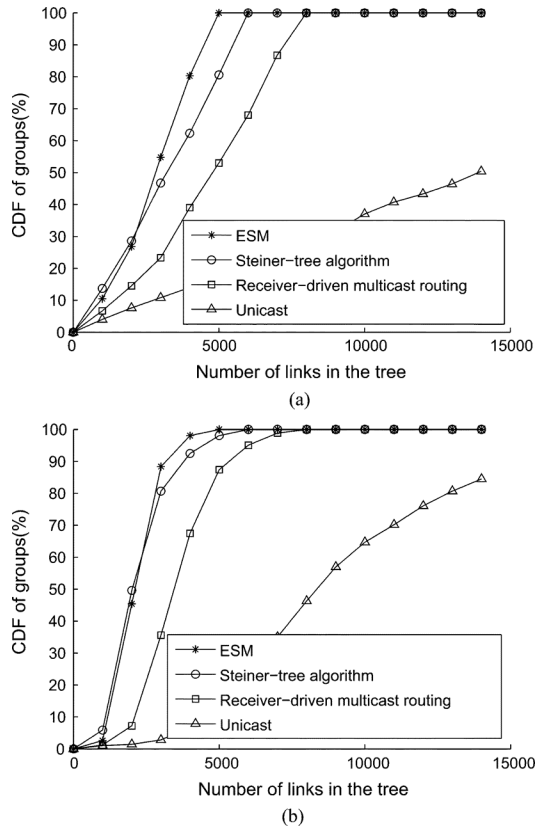


Fig. 6. CDF of groups occupying different number of links under three tree-building algorithms in BCube. Unicast plays as the benchmark. (a) Uniform group size distribution. (b) Power-law group size distribution.

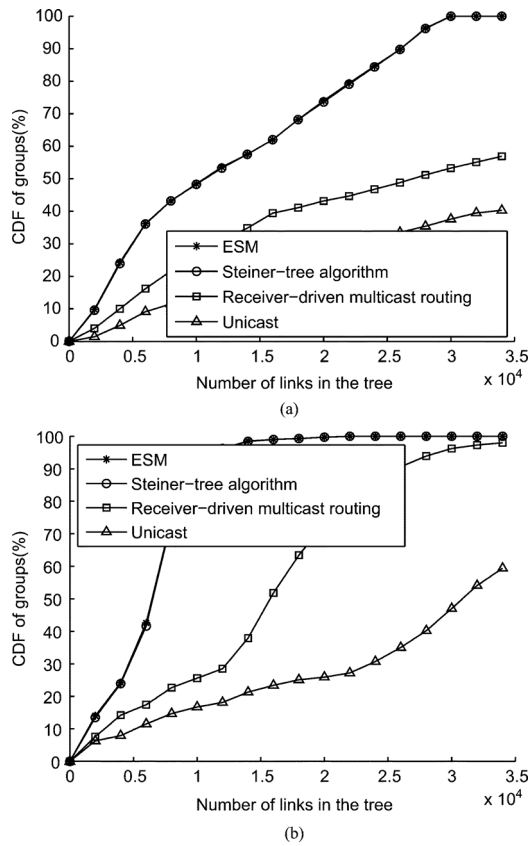


Fig. 7. CDF of groups occupying different number of links under different tree-building algorithms in Fat-Tree. Unicast plays as the benchmark. (a) Uniform group size distribution. (b) Power-law group size distribution.

in this case, the group members are diversely distributed within the network. ESM restricts the tree depth as the number of stages in the group spanning graph, while there is no such restriction in the Steiner-tree algorithm. However, for larger groups, ESM better exploits the topological feature of BCube, and the resultant tree has less links than the Steiner-tree algorithm, even with limitation on tree depth. In Fat-Tree, the two algorithms perform exactly the same, as shown in Fig. 7. This is because in Fat-Tree structure, both algorithms can form the optimal multicast tree.

We take the number of links occupied by unicast routing as the benchmark. We find that all three multicast routing schemes perform remarkably better than unicast routing in both the networks. It demonstrates the necessity and importance of multicast routing in data center networks. For example, in BCube(8,3) network, all multicast groups can build a multicast tree with less than 5000 links in ESM, while only around 20% groups occupy less than 5000 links using unicast routing.

Furthermore, we find that the number of links used under power-law distribution is less than that under uniform distribution. It follows our intuition because there are more small groups in power-law distribution, which matches the actual group distribution in data center networks better.

Multicast Throughput: We then calculate the throughputs of all 1000 multicast groups with different tree-building algorithms, in both BCube and Fat-Tree Networks. Figs. 8 and 9 show the results.

From the figures, we observe that the average multicast throughput of ESM at least doubles that of receiver-driven multicast routing in most cases. It follows the link-saving results in multicast trees. When each multicast group occupies less links, the average number of groups competing for the network resource on the same link will be less, and thus the multicast throughput can be higher. In BCube network, ESM also outperforms the Steiner-tree algorithm because the average number of links used by a group in ESM is less than that in the Steiner-tree algorithm. In the Fat-Tree network, the multicast throughputs of the two algorithms are the same because they both build the optimal multicast trees.

Compared to unicast, the throughput of the multicast groups in ESM is tens of times higher. It demonstrates the necessity of multicast again from the angle of the end-to-end performance of applications. In addition, we find that the multicast throughput in power-law group size distribution is higher than that in uniform distribution.

C. Combination Routing Scheme

We then evaluate the combination routing scheme by installing different numbers of multicast routing entries in the switches. To demonstrate the performance of the combination routing, the number of routing entries installed in each switch is less than the total number of groups in the network. The number of larger groups taking in-switch routing equals the maximum number of in-switch entries, while the other groups use in-packet Bloom-Filter-based routing. In other words, the threshold in the combination routing is set according to the

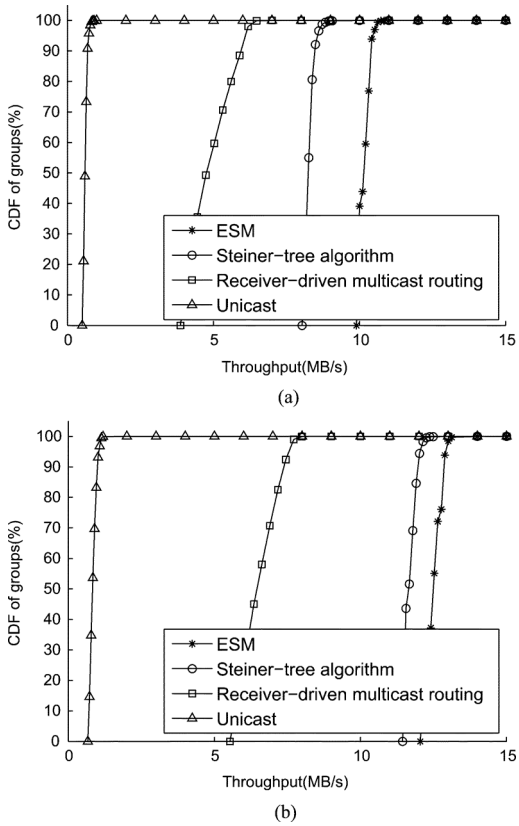


Fig. 8. CDF of groups with different network throughputs under three tree-building algorithms in BCube. Unicast plays as the benchmark. (a) Uniform group size distribution. (b) Power-law group size distribution.

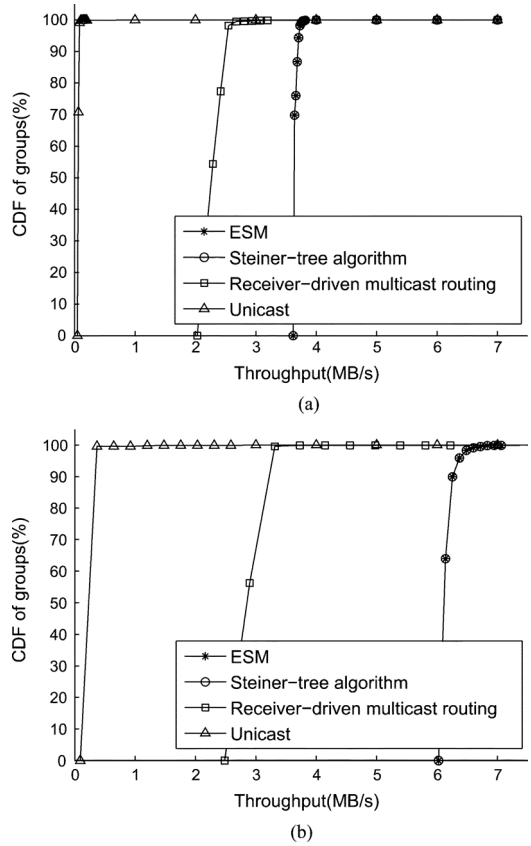


Fig. 9. CDF of groups with different network throughputs under three tree-building algorithms in Fat-Tree. Unicast plays as the benchmark. (a) Uniform group size distribution. (b) Power-law group size distribution.

maximum routing table memory in switches. We also investigate the impacts of the group size distribution pattern and the Bloom Filter length.

Figs. 10 and 11 show the bandwidth overhead ratio of the 1000 multicast groups in BCube and Fat-Tree networks, respectively, by assuming each group has the same amount of traffic. We observe that the combination routing scheme in ESM can effectively reduce the requirement of in-switch routing table memory. From Fig. 10(b), in power-law group size distribution in BCube, when the maximum number of in-switch routing entries is 400, we can control the bandwidth overhead ratio as low as 0.1%. It indicates that we can save 60% in-switch routing table memory with little traffic leakage. When the group size is uniformly distributed, the number of in-switch routing entries ESM requires is more because the average group size is larger, as shown in Fig. 10(a). However, we can also save 40% routing table memory by limiting the bandwidth overhead ratio below 0.5%.

Fig. 11 demonstrates the results for the Fat-Tree network. ESM needs more in-switch routing entries to achieve low bandwidth overhead ratio. It is because, in Fat-Tree, each server only has one NIC port to connect the edge-level switch. Hence, an edge-level switch has to maintain a multicast routing entry as long as there is a group member from the connected servers (it is not the case in BCube because every server is connected to multiple switches). However, under power-law group size distribution, ESM can still save 40% in-switch routing table memory by controlling the bandwidth overhead ratio less than 10%, while in uniform group size distribution, ESM also saves 40% in-switch routing table memory if we can tolerate the bandwidth overhead ratio of 12%.

VI. Implementation and Experiments

In this section, we present the implementation and experiments of ESM.

A. Implementation

Our implementation of ESM includes three parts—namely, the multicast manager, the multicast library on servers, as well as the combination forwarding engine. The multicast manager is responsible for calculating the multicast tree, configuring the multicast routing rules on switches, and sending the tree information to the source server if in-packet Bloom-Filter-based multicast routing is used. The multicast library on servers redirects the multicast group join/leave requests to the multicast manager, inserts the Bloom Filter field into an outgoing packet header if required, and removes the Bloom Filter field from an incoming packet if it exists. We insert a 32-B Bloom Filter field into the IP option field if Bloom-Filter-based routing is required.

The combination forwarding engine modifies the data plane of switches to realize the combination routing scheme in ESM. We can employ OpenFlow [25] framework for the implementation. The key of the combination forwarding engine is to introduce Bloom-Filter-based forwarding when the tag of a packet indicates that in-packet Bloom Filter is used. The Bloomed switch identifier is a wildcarded bitmask with k bits

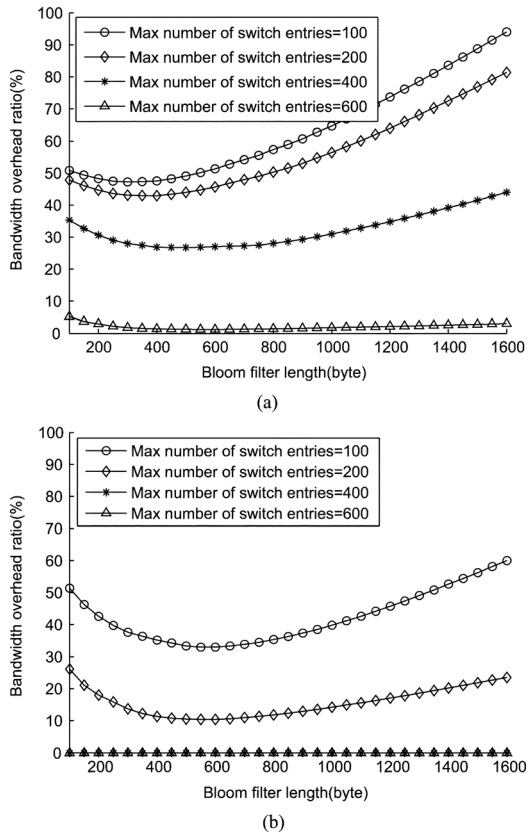


Fig. 10. Bandwidth overhead ratio of the combination routing scheme against the number of in-switch routing entries and the Bloom Filter length in BCube. (a) Uniform group size distribution. (b) Power-law group size distribution.

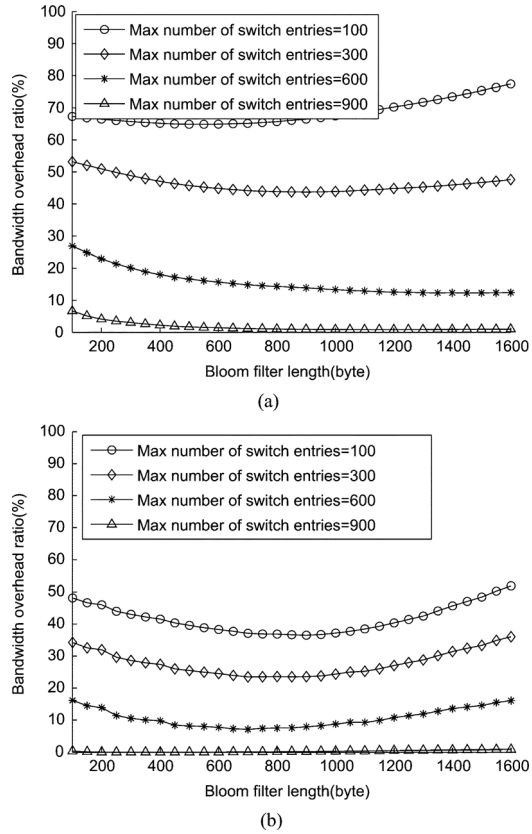


Fig. 11. Bandwidth overhead ratio of the combination routing scheme against the number of in-switch routing entries and the Bloom Filter length in Fat-Tree. (a) Uniform group size distribution. (b) Power-law group size distribution.

set to one. It has been shown that only several lines of code have to be modified based on the current OpenFlow reference implementations [26] to support this kind of special behavior in the OpenFlow data path. More specifically, we need to modify the function flow_fields_match in openflow1.0.0/udatapath/switch-flow.c or openflow0.9.0/datapath/flow.c [26]. This supports our belief that our proposed combination forwarding scheme can be well incorporated into existing commodity switches. At the current stage, we have implemented a software-based combination forwarding engine on a Linux platform to embrace legacy applications using IP multicast.

B. Experiments

We conduct experiments based on our implementation of ESM to evaluate the computation overhead of the tree-building algorithm on the multicast manager, as well as the forwarding overhead of the combination forwarding engine.

Computation Overhead of the Tree-Building Algorithm: We deploy the multicast manager on a desktop with AMD Athlon II X2 245 2.91 G CPU and 2 GB DRAM. To evaluate the computation overhead in real large-scale data center networks, we emulate the group join/leave requests in BCube and Fat-Tree networks, respectively. One thousand groups are generated to dynamically join the network. The group size is distributed within $[3, N]$ with a power-law distribution, where N is the total number of servers in the network.

We compare the computation time on the multicast manager for building the tree of each group under the ESM algorithm and the Steiner-tree algorithm. Fig. 12 illustrates the CDF of groups taking different computation time. We find that the computation speed of ESM is *orders of magnitudes* faster than the Steiner-tree algorithm. The reason is that ESM fully leverages the topological feature of data center topologies. Under the Steiner-tree algorithm, only about 32% of groups in BCube and 47% of groups in Fat-Tree from all the randomly generated groups can form the tree within 1 s. However, ESM can finish the tree computation for all groups within 20 ms in BCube and 260 ms in Fat-Tree.

Forwarding Overhead on the Combination Forwarding Engine: We evaluate the performance of our combination forwarding engine on a simple testbed. The testbed is composed of five hosts, i.e., one multicast sender, three multicast receivers, and one forwarder. The forwarder has one 2.8 G Intel Core 2 Duo CPU, 2 GB DRAM, and four Realtek RTL8111/8168B PCI Express Gigabit Ethernet NICs, each connecting one of the other four hosts. The forwarder is installed with Ubuntu Linux 9.10 (karmic) with kernel 2.6.31–14-generic and also equipped with the ESM combination forwarding engine. The three receivers send group join requests to the multicast manager. The multicast manager then configures the forwarder about the multicast forwarding rule. The sender sends out UDP multicast packets at different speeds.

Fig. 13 shows the CPU utilization ratio on the forwarder and the packet loss ratio for different packet rates. We can see that the ESM combination forwarding engine can control the packet loss ratio under 0.1% even at packet speeds close to 1 Gb/s.

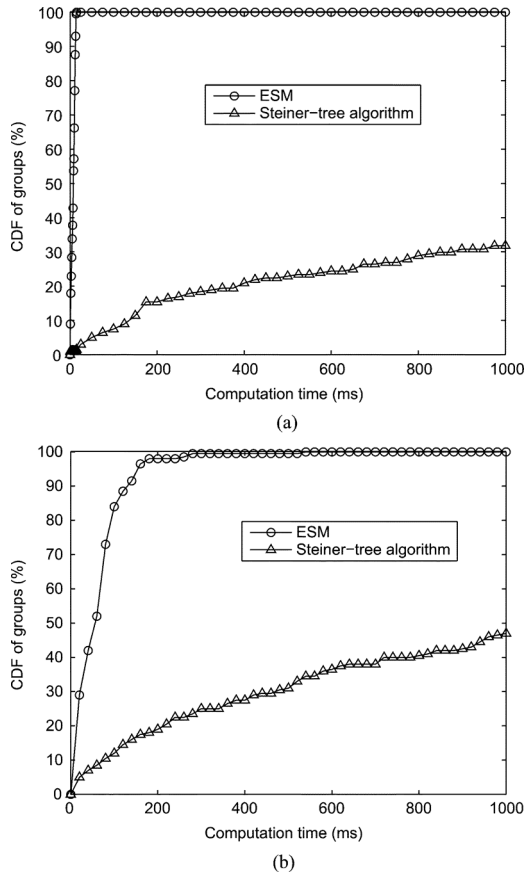


Fig. 12. CDF of groups taking different computation time by ESM and the Steiner-tree algorithm. (a) BCube. (b) Fat-Tree.

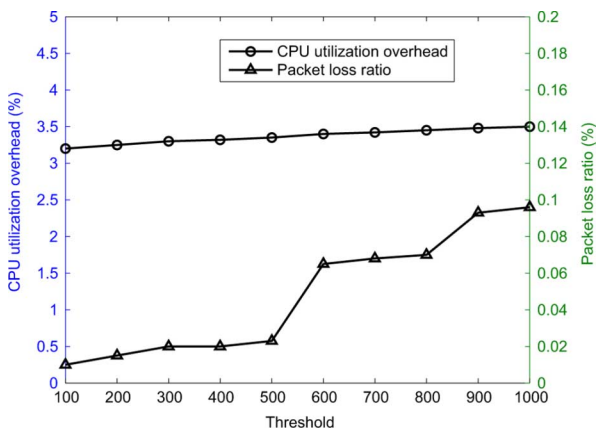


Fig. 13. CPU utilization on the forwarder as well as the packet loss ratio against different data rates.

Meanwhile, the CPU overhead on the forwarder is less than 4%, demonstrating the light weight of the forwarding engine.

VII. Related Work

In this section, we discuss the related work on data center multicast routing, including multicast tree formation and scalable multicast routing.

A. Multicast Tree Formation

Building a multicast tree with the lowest cost covering a given set of nodes on a general graph is well known as the Steiner Tree problem [13]. The problem is NP-Hard, and there are many approximate solutions [14], [15]. For the Internet, multicast routing protocols are designed to build the delivery tree, among which PIM is the most widely used one. In PIM-SM [16] or PIM-SSM [17], receivers independently send group join/leave requests to a rendezvous point or the source node, and the multicast tree is thus formed by the reverse unicast routing in the intermediate routers.

For data center multicast, BCube [7] proposes an algorithm to build server-based multicast trees, and switches are used only as dummy crossbars. However, obviously network-level multicast with switches involved can save much more bandwidth than the server-based one. In VL2 [9], it is suggested that traditional IP multicast protocols are used for tree building, like in the Internet. In Fat-Tree [8], a simple tree-building algorithm is also designed. However, in this paper, we target at efficient multicast tree formation in generic data center networks by exploiting the technical trend on data center design.

B. Scalable Multicast Routing

As one of the deploying obstacles of multicast in the Internet, scalable multicast routing has attracted much attention in the research community. For data center networks where low-end switches with limited hardware routing space is used, the problem is even more challenging.

Bloom Filter can be used to compress in-switch multicast routing entries. In FRM [18], Bloom-Filter-based group information is maintained at border routers to help determine inter-domain multicast packet forwarding. A similar idea is adopted in BUFFALO [19], though it is primarily designed for scalable unicast routing. In ESM, we use in-packet Bloom Filter to further save the multicast routing entries on switches.

LIPSIN [21] also adopts in-packet Bloom Filter for multicast routing. In ESM, we achieve scalable multicast routing by making the tradeoff between the number of multicast groups supported and the additional bandwidth overhead. By exploiting the feature of future data center networks, we use node-based Bloom Filter to encode the tree instead of the link-based one in LIPSIN and design a loop-free forwarding scheme.

In MCMD [5], scalable data center multicast is realized in the way that only a selection of groups are supported by multicast according to the hardware capacity, and the other groups are translated into unicast communications. Our work differs from MCMD in that we use multicast to support all group communications instead of turning to unicast since multicast exposes significant gains over unicast in traffic saving and throughput enhancement.

VIII. Conclusion

In this paper, we design ESM, an efficient and scalable multicast routing scheme for modern data center networks. Given that receiver-driven multicast routing does not perform well in densely connected data center networks, ESM efficiently build the multicast tree by leveraging the multistage graph feature of data center networks. ESM combines both in-packet Bloom Filters and in-switch entries to make the tradeoff between the number of multicast groups supported and the additional bandwidth overhead, so as to achieve scalable routing. The simulation results show that the tree-building algorithm is effective in

reducing the number of links in a multicast tree and improving the multicast throughput. The combination routing scheme also performs well in saving the in-switch routing table memory with low cost of traffic leakage. We have implemented the prototype of ESM. The experiments further demonstrate that the tree-building algorithm of ESM enjoys much lower computation complexity than the general Steiner-tree algorithm, and the software-based combination forwarding engine can run at high packet rate with less than 4% CPU overhead.

ACKNOWLEDGMENT

Some preliminary results of this paper were published in the *Proceedings of IEEE INFOCOM 2011* [28]. In this paper, we have made the following improvements. First, we make more arguments and explanations about the design motivation and the design rationale; Second, we add the evaluation of end-to-end multicast throughput and the combination routing scheme; Third, we have developed the whole system.

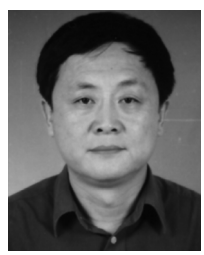
REFERENCES

[1] "Hadoop," Apache Software Foundation, 2011 [Online]. Available: http://hadoop.apache.org/
[2] S. Ghemawat, H. Gobio, and S. Leungm, "The Google file system," in *Proc. ACM SOSP.*, 2003, pp. 29–43.
[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. OSDI*, 2004, vol. 6, p. 10.
[4] T. Hoff, "Google architecture," Jul. 2008 [Online]. Available: http://highscalability.com/google-architecture
[5] Y. Vigfusson, H. Abu-Libdeh, and M. Balakrishnan, "Dr. multicast: Rx for data center communication scalability," in *Proc. ACM Eurosys*, Apr. 2010, pp. 349–362.
[6] A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao, "Towards automated performance diagnosis in a large IPTV network," in *Proc. ACM SIGCOMM*, 2009, pp. 231–242.
[7] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, and C. Tian, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. ACM SIGCOMM*, Aug. 2009, pp. 63–74.
[8] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, Aug. 2008, pp. 63–74.
[9] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM*, Aug. 2009, pp. 51–62.
[10] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "FiConn: Using backup port for server interconnection in data centers," in *Proc. IEEE INFOCOM*, May 2009, pp. 2276–2285.
[11] D. Li, M. Xu, H. Zhao, and X. Fu, "Building mega data center from heterogeneous containers," in *Proc. IEEE ICNP*, Oct. 2011, to be published.
[12] D. Newman, "10 Gig access switches: Not just packet-pushers anymore," *Netw. World*, vol. 25, no. 12, Mar. 2008.
[13] "Steiner tree problem," 2011 [Online]. Available: http://en.wikipedia.org/wiki/Steiner_tree_problem
[14] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for Steiner trees," *Acta Inf.*, vol. 15, pp. 141–145, 1981.
[15] G. Robins and A. Zelikovsky, "Tighter bounds for graph Steiner tree approximation," *SIAM J. Discrete Math.*, vol. 19, no. 1, pp. 122–134, 2005.
[16] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol independent multicast-Sparse mode (PIM-SM): Protocol specification," RFC 2362, Jun. 1998.
[17] S. Bhattacharyya, "An overview of source-specific multicast (SSM)," RFC 3569, Jul. 2003.
[18] S. Ratnasamy, A. Ermolinskiy, and S. Shenker, "Revisiting IP multicast," in *Proc. ACM SIGCOMM*, Aug. 2006, pp. 15–26.
[19] M. Yu, A. Fabrikant, and J. Rexford, "BUFFALO: Bloom filter forwarding architecture for large organizations," in *Proc. ACM CoNEXT*, Dec. 2009, pp. 313–324.
[20] D. Li, H. Cui, Y. Hu, Y. Xia, and X. Wang, "Scalable data center multicast using multi-class bloom filter," in *Proc. IEEE ICNP*, Oct. 2011, to be published.
[21] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line speed publish/subscribe inter-networking," in *Proc. ACM SIGCOMM*, Aug. 2009, pp. 195–206.
[22] "Multistage interconnection networks," 2010 [Online]. Available: http://en.wikipedia.org/wiki/Multistage_interconnection_networks
[23] "Hadoop Summit," 2010 [Online]. Available: http://developer.yahoo.com/events/hadoopsummit2010
[24] Y. Lan, A. Esfahanian, and L. Ni, "Multicast in hypercube multiprocessors," *J. Parallel Distrib. Comput.*, vol. 8, no. 1, pp. 30–41, 1990.
[25] "OpenFlow," Stanford University, Stanford, CA, 2008 [Online]. Available: http://www.openflowswitch.org/
[26] C. Rothenberg, C. Macapuna, F. Verdi, M. Magalháes, and A. Zahemszky, "Data center networking with in-packet bloom filters," in *Proc. SBRC*, May 2010, pp. 553–566.
[27] A. Brodery and M. Mitzenmacherz, "Network applications of bloom filters: A survey," in *Proc. Internet Math.*, 2003, vol. 1, no. 4, pp. 485–509.
[28] D. Li, J. Yu, J. Yu, and J. Wu, "Exploring efficient and scalable multicast routing in future data center networks," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1368–1376.

Dan Li (M'10) received the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2007.

He is now an Assistant Professor with the Computer Science Department, Tsinghua University. Before joining the faculty of Tsinghua University, he spent two years as an Associate Researcher with the Wireless and Networking Group, Microsoft Research Asia, Beijing, China. His research interests include Internet architecture and protocols, data center networking, and green networking.

Yuanjie Li is an undergraduate student with Tsinghua University, Beijing, China.



Jianping Wu (SM'05) received the M.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 1997.

He is now a Full Professor with the Computer Science Department, Tsinghua University. In the research areas of the network architecture, high-performance routing and switching, protocol testing, and formal methods, he has published more than 200 technical papers in academic journals and proceedings of international conferences.

Sen Su received the B.S. degree in 1992 from Sichuan University, Chengdu, China, in 1992, and the M.S. and Ph.D. degrees from the University of Electronic Science and Technology, Chengdu, China, in 1995 and 1998, respectively.

He is currently a Professor and the Director of the Switching and Intelligent Control Research Center, Beijing University of Posts and Telecommunications, Beijing, China. His research interests include distributed computing, network virtualization, and cloud security.

Jiangwei Yu is an undergraduate student with Tsinghua University, Beijing, China.