Near-Optimal Multicast Tree Construction in Leaf-Spine Data Center Networks
Shouxi Luo*, Huanlai Xing, and Ke Li

Abstract—In this paper, we revisit the optimization problem of constructing multicast trees for cloud applications in the context of leaf-spine data center network (DCN). On one hand, we find that, the maximum multicast rate a network can provide equals to the minimum of the maximum unsplit throughput it can provide to each of the receivers. On the other hand, by taking the characteristic of leaf-spine DCN into account, we prove that the optimal construction of multicast tree in leaf-spine DCNs is equivalent to the well-known problem of minimum set cover, distinguished from the conventional wisdom that models the problem as a general-purpose directed minimum Steiner. Based on these findings, we develop simple yet near-optimal algorithms to construct multicast trees and deal with network failures.

Index Terms—Data center network (DCN), multicast, set cover.

I. INTRODUCTION

In modern data centers, distributed systems commonly trigger one-to-many group communication workloads for tasks, such as file dissemination, data replication, distributed messaging etc. [1]–[3]. For these types of workload, network-layer multicast is widely considered as the best solution since it natively involves less traffic load and server overheads [2]. Luckily, recent advantages of software-defined networking (SDN)-based data center multicast have made this vision a reality. For example, with techniques, such as controller-assisted forwarding rule optimization [3] and source-routed forwarding [2], data center networks (DCNs) can dynamically set up and control a large number of explicit multicast trees for data center workloads on demand.

With this ability of flexible multicast control, the follow-up question is—for a multicast request, how to construct a multicast tree to meet its demands, regarding that applications generally have QoS requirements (e.g., bandwidth) while there are abundant equal cost paths between hosts in modern data centers? Although many previous papers have worked on this topic, they either only consider the case where congestions are allowed and switch and link never fails [1] or employ general models (e.g., directed minimum Steiner) which are sub-optimal because of the ignorance of data center characteristics [4]–[6].

In this paper, we theoretically revisit this optimization problem of constructing multicast trees in the context of leaf-spine DCN and try to answer the following fundamental questions.

Q1. What is the maximum multicast rate that the network can provide?
Q2. Which multicast tree is best for a specified bandwidth requirement?

Q3. How to reconstruct multicast trees efficiently upon switch and link failures?

For the first question we find that the maximum multicast rate a network can provide equals to the minimum of the maximum unsplit throughput it can provide to each of the receivers along (Theorem 1 in Section II-A). Regarding question 2 choosing the tree involving the minimum number of spine switches is preferred, since the amount of both uplink traffic and occupied forwarding table size could be minimized. Although the problem of minimum multicast tree construction is well studied in the general case, we employ a novel model by taking the characteristic of leaf-spine DCN into account, and thus, obtain new results and design a near-optimal algorithm for tree construction (Theorem 2 and Algorithm 1 in Section II-B). Based on these findings, we further develop a failure recovery scheme that could reconstruct multicast trees for admitted requests by providing fairly downgraded performance guarantees upon switch and link failure (Algorithm 2 in Section II-C).

II. TOWARD OPTIMAL MULTICAST TREE

In this section, we give theory-based answers to aforementioned questions and design efficient near-optimal algorithms.

Before looking into the details, let us overview the data center architecture considered in this paper first. To provide very scalable throughput and predictable latencies in a uniform manner, today’s data centers widely adopt the leaf-spine architecture (see Fig. 1) for networking or as its basic building blocks [9], [10]. In this architecture, NR switches are interconnected with each other and to the external network via R leaf- and M spine- switches (denoted by V_L and V_M, respectively), who themselves are connected in a full bipartite graph in turn. For ease of analysis, we consider the simplified two-layered spine-leaf DCN in this paper without loss of generality.
A. Maximum Multicast Rate Computation

Intuitively, the maximum multicast rate that a network can provide from sender \( s \) to a group of receivers \( D \), would not exceed the maximum (unsplit) bandwidth each receiver can achieve along. Indeed, as Theorem 1 says, by selecting the minimum value among all receivers’ maximum possible throughput, we get the optimal solution to question 1.

**Theorem 1:** Let \( P_{s,d} \) be the set of all simple paths from source \( s \) to destination \( d \), and \( B_p \) be the available bandwidth on path \( p \); then the maximum multicast rate from sender \( s \) to a group of receivers \( D \), saying \( B_{\max}^{s,D} \), is \( \min_{d\in D} \max_{p\in P_{s,d}} B_p \).

**Proof:** The proof is straightforward. We first show that \( \min_{d\in D} \max_{p\in P_{s,d}} B_p \) is the upper bound of multicast rate the network can provide for \( s \) and \( D \). Since multicast only employs a single path for each receiver, the maximum possible rate from source \( s \) to a destination \( d \), could never exceed the maximum bandwidth among all the available paths between them, i.e., \( \max_{\forall p\in P_{s,d}} B_p \). Moreover, all receivers share the same streaming in multicast; then the maximum possible multicast rate \( B_{\max}^{s,D} \) would never exceed the minimum rate among all receivers, i.e., \( \min_{d\in D} \max_{p\in P_{s,d}} B_p \).

Next, we show how to build a tree to achieve this optimal multicast rate for \( s \) and \( D \). Generally, there exists one or more paths with the available bandwidth of \( \min_{d\in D} \max_{p\in P_{s,d}} B_p \) for each receiver \( d \in D \). We randomly choose one qualified path for each and merge these \( |D| \) selected paths into a directed graph (note that they share the same source node \( s \)). Then, by generating a spanning tree from this generated graph, we get a tree with available multicast bandwidth \( \min_{d\in D} \max_{p\in P_{s,d}} B_p \) from root \( s \) to leaves \( D \).

Suppose that the available bandwidth on link \((u,v)\) is \( B_{u,v} \), and the leaf switch of server \( s \) is \( L[s] \). According to the structure of leaf-spine DCN (see Fig. 1) if sender \( s \) and receiver \( d \) are under the same leaf switch (i.e., \( L[u]=L[d] \)), there is only a simple path (with two hops) between them, and thus, \( \max_{\forall p\in P_{s,d}} = \min(B_{s,L[u]}, B_{L[u],d}) \). Otherwise, there are \( M \) alternative simple paths (with four hops) crossing over the \( M \) spine switches, and thus, \( \max_{\forall p\in P_{s,d}} = \max_{\forall V_M \in \text{V}_M} \min(B_{s,L[u]}, B_{L[u],d}) \), in which \( \text{V}_M \) is the set of spine switches. Obviously, for a leaf-spine data center with \( M \) spines, it is easy to obtain the maximum supported multicast rate from \( s \) to \( D \) within \( O(MD) \) time.

B. Minimum Tree Construction

For a multicast task crossing multiple leaf switches, there might be multiple choices on constructing its tree, due to the abundant equal-cost alternative paths provided by leaf-spine DCN. As an example, consider the case shown in Fig. 2, where an application needs to disseminate a file from the sender belonging to leaf switch \( L_1 \) to three receivers (not shown in figure for simplicity) belonging to leaf switches \( L_2, L_3, \) and \( L_4 \), respectively. Since there are many fabric links that satisfy the bandwidth requirements, the network can multicast the data file by using all available spine switches [see Fig. 2(a)], or just one [see Fig. 2(b)]. To reduce traffic loads and the number of forwarding rules with the power of multicast, the optimal tree is the one involving the minimum number of spine switches.

1) Equivalent to Minimum Set Cover: To make the analysis specific, we denote the multicast request as \( f = (s, D, b) \), which means a multicast tree from \( s \) to a group of receivers \( D \) with bandwidth \( b \) is desired. For server \( s \), let \( \mathbb{S}[s,b] \) denote the set of spines switches, to whom, there are uplinks with available bandwidth \( b \) from server \( s \), i.e., (1). And for spine switch \( x \), let \( \mathbb{S}[x,b] \) denote the set of leaf switches, to whom, there exists downlinks with available bandwidth \( b \) from spine switch \( x \), i.e., (2).

Recall that, the routes between intra-leaf servers are fixed. Thus, for ease of description, we directly focus on constructing a tree for receivers that do not share the leaf switch with the sender and only consider the bandwidth limits of fabric links during our analysis without loss of generality. For request \( f \), let \( L_f \) denote the set of receivers’ leaf switches distinct from that of the sender, i.e., (3). Obviously, the spine switches that could be employed in \( f \)’s multicast tree, must belong to set \( \mathbb{S}[s,b] \). By using a binary variable \( x \), to indicate whether spine switch \( j \) is involved in the optimized multicast tree, this minimum spine switch selection problem can be formulated as the following integer LP (ILP), which is equivalent to the well-known NP-hard problem of minimum set cover as Theorem 2 says.

\[
\begin{align*}
\text{minimize} & \quad \sum_{j \in \mathbb{V}_M} x_j \\
\text{subject to} & \quad x_j \in \{0,1\}, \quad \forall j \in \mathbb{V}_M \\
& \quad \sum_{j \in \mathbb{S}_j[b]} x_j \geq 1, \quad \forall k \in L_f.
\end{align*}
\]

**Theorem 2:** Finding the multicast tree involving the minimum number of spine switches in leaf-spine DCNs, is equivalent to the problem of minimum set cover [8].

**Proof:** For each spine switch \( j \in \mathbb{S}[s,b] \), let \( H_j = \mathbb{S}[j,b] \) and \( L_f \), denoting the subset of receiver’s leaf switches, to whom this spine switch has \( b \)-capacity downlinks. Then, \( \{j \in \mathbb{S}[s,b] : k \in H_j \} \), the set of spines that have sufficient uplink and downlink bandwidth to setup \( b \)-capacity paths from sender \( s \) to receiver \( d \), can be described as \( \{j : d \in H_j\} \). By further letting \( \mathbb{H} = \{H_j : \forall j \in \mathbb{S}[s,b]\} \), the
where $n$ is the number of uncovered receivers until all of them get covered, we find a switch. By repeatedly selecting the spine that contains the largest selection, mscTree also records the set of leaves covered by spine. mscTree first extracts the qualified spine switches and gets the set of leaves covered by spine. Then, for any request $f$, mscTree runs for it to find its maximum multicast rate that the network can provide, i.e., $B_{f\max}$.

Let $F$ be the set of multicast requests broken by the failure, in which the $i$th request is denoted by $f_i: (s_i, D_i, b_i)$; and suppose the remaining bandwidth on link $(u, v)$ is $B_{u,v}$; then our reconstruction processes are in four steps as Algorithm 2 shows. Following these, broken multicast trees are reconstructed with the support of downgraded rates. One may notice that, if all links are already fully used, there does not exist feasible links for tree reconstruction. To avoid this in practice, a simple yet useful design is to let links have capacity slacks, 5% for instance, when accepting multicast requests.

### III. Evaluation

As mscTree is the key of both the construction and reconstruction of multicast tree, in this part, we verify its benefits by employing it to construct trees for synthesized requests.

#### A. Methodology

To highlight the comparison, we consider the capacity limits on fabric links and focus on the optimized selection of spine switches in constructing multicast trees. For multicast request $f: (s, D, b)$, the task is to construct a tree with available multicast bandwidth $b$, from its sender’s leaf switch $s$ to its receivers’ leaf switches $D$. For each request, we first check whether the maximum multicast rate that the network could provide to it (Theorem 1), satisfies its requirements or not. If so, we admit it and build up its tree via mscTree. For the construction of multicast tree, we use the most widely deployed multicast routing protocol, protocol independent multicast (PIM), as the baseline. On admitting a request, PIM builds a shortest unicast path tree to multicast. However, since all alternative paths are with the equivalent cost in leaf-spine DCNs, the process of PIM (and other shortest path-based heuristics [5], [7]) is essentially to select a random spine for each receiver; thus we refer it as random. Besides, we also consider the least-load-spine-firsts (LLSF) greedy which chooses the spine involving the least used downlink bandwidth to $d$ as $d$’s parent node to construct trees.

We assume that the multicast rate $b$ follows a power-law alike distribution, where each request’s rate is randomly selected from a pre-defined set $\{\alpha^2: 0 \leq \alpha \leq n_r\}$ with the probability of $P(b = \alpha^2) = \frac{\alpha^2}{\sum_{i=0}^{n_r} \alpha^2}$. Moreover, we assume that the links are randomly chosen, in which the size of $D$ follows the uniform distribution of $U[2, \max]$. Therefore, the mean multicast throughput of all requests is $\frac{1}{n_r} \sum_{i=0}^{n_r} \frac{\alpha^2}{\sum_{i=0}^{n_r} \alpha^2} = \frac{\alpha^2}{\sum_{i=0}^{n_r+1} \alpha^2}$. We consider a cluster containing $M$ spines and $R$ leaves, and assume that each fabric link has the normalized bidirectional bandwidth of 1. In each test, we generate $\frac{M}{n_r+1}B_{u,v}$ requests to fulfill the network, then calculate the number of accepted requests, their multicast throughputs, and the corresponding loads on fabric uplinks.

#### Algorithm 1: mscTree: Minimum Set Cover Tree Construction.

<table>
<thead>
<tr>
<th>Algorithm 1: mscTree: Minimum Set Cover Tree Construction.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constants:</strong> $V_M, V_N, V_R, L</td>
</tr>
<tr>
<td><strong>Inputs:</strong> request $f: (s, D, b)$, $B_{u,v}$ for each link</td>
</tr>
<tr>
<td><strong>Outputs:</strong> $T$: the constructed tree for $f$</td>
</tr>
<tr>
<td>1: $L_f \leftarrow {L[d]: d \in D, L[d] \neq L[s]}$</td>
</tr>
<tr>
<td>2: $S[b, l] \leftarrow {j \in V_M: B_{l,s,j} \geq b}$</td>
</tr>
<tr>
<td>3: $\emptyset \leftarrow H, H^+ \leftarrow \emptyset$</td>
</tr>
<tr>
<td>4: <strong>for all</strong> $j \in S[b, l]$ <strong>do</strong></td>
</tr>
<tr>
<td>5: $H_j \leftarrow {k \in L_f: B_{j,k} \geq b}$</td>
</tr>
<tr>
<td>6: $\emptyset \leftarrow H \leftarrow {H_j}$</td>
</tr>
<tr>
<td>7: <strong>while</strong> $L_f \neq \emptyset$ <strong>do</strong></td>
</tr>
<tr>
<td>8: select an $H_j \in H$ that maximizes $L_f \cap H_j$</td>
</tr>
<tr>
<td>9: $\pi[j] \leftarrow L_f \cap H_j$</td>
</tr>
<tr>
<td>10: $L_f \leftarrow L_f \setminus H_j$</td>
</tr>
<tr>
<td>11: $H^+ \leftarrow H^+ \cup {H_j}$</td>
</tr>
<tr>
<td>12: $H \leftarrow H^+ \setminus {H_j}$</td>
</tr>
<tr>
<td>13: $T \leftarrow ({S[l,s]} \cup {(L[d], d): d \in D}$</td>
</tr>
<tr>
<td>14: <strong>for all</strong> $j \in {j: H_j \in H^+}$ <strong>do</strong></td>
</tr>
<tr>
<td>15: $T \leftarrow T \cup {L[s,j]} \cup {(j, k): k \in \pi[j]}$</td>
</tr>
<tr>
<td>16: <strong>return</strong> $T$</td>
</tr>
</tbody>
</table>

#### Algorithm 2: Tree Reconstruction Upon Network Failure.

<table>
<thead>
<tr>
<th>Algorithm 2: Tree Reconstruction Upon Network Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: For each involved request $f_i \in F$, find its maximum multicast rate that the network can provide, i.e., $B_{f_{i\max}}$;</td>
</tr>
<tr>
<td>2: For request $f_i \in F$, reconstruct its multicast tree based on $T_i$, which is the results of mscTree running for demand $(s_i, D_i, b_i, B_{f_{i\max}})$;</td>
</tr>
<tr>
<td>3: For each congested link $(u, v)$, compute the scale factor $\lambda_{u,v}$ that could remove its congestion; otherwise, $\lambda_{u,v} = \min(1, B_{u,v}/\sum_{f_i \in F(u,v)} \gamma_i b_i)$;</td>
</tr>
<tr>
<td>4: For request $f_i \in F$, scale its multicast rate from $b_i$ down to $b_i \times \min_{(u,v) \in T_i} \lambda_{u,v}$, such that congestions are eliminated.</td>
</tr>
</tbody>
</table>

The aforementioned ILP (4) can be reformulated as follows, which is exactly the well-known problem of minimum set cover [8]:

\[
\begin{align*}
\text{minimize} & \quad \sum_{j \in H_j \in H} x_j \\
\text{subject to} & \quad x_j \in \{0, 1\}, \quad \forall H_j \in H \\
& \quad \sum_{j \in H_j \in H} x_j \geq 1, \quad \forall k \in L_f.
\end{align*}
\]
Fig. 3. mscTree provides larger multicast throughput by accepting more wider requests, and reduces uplink loads greatly. (a) Accepted requests. (b) Accepted throughput. (c) Load on fabric uplinks.

B. Results

We feed the tests with different parameters and repeat the simulation 100 times for each parameter setting. Extensive results imply consistent conclusions. Due to the limits of space, we showcase the results of instances where $M = 16$, $R = 48$, $n_r = 5$, $\alpha = 0.01$, and $r_{max} = \lfloor \frac{R - 3}{3} + 1 \rfloor = 16$. Basically, the network has a total throughput of 768. And we try to fulfill it with 2799 generated multicast requests.

As Fig. 3(a) and (b) shows, mscTree is able to provide a higher multicast throughput with a fewer accepted requests. For instance, it accepts about 2633 requests in average, which occupy about 0.940 of the total multicast requests, and are about 0.007 and 0.034 less than these of the random and LLSF strategies. However, regarding the total provided multicast throughput, in most cases, mscTree outperforms the random and LLSF strategies with the average value 0.005 and 0.024, respectively. Recall that, for each request, its multicast throughput, $b|D|$, is a fixed value that cannot be optimized through tree construction, representing the total fabric downlink bandwidth it needs. Thus, the simulation results imply that, mscTree provides more multicast throughput by accepting multicast requests that have more receivers and making more efficient use of downlink bandwidth.

The most important, as Fig. 3(c) shows, with theory-based optimization of tree construction, mscTree is able to reduce the redundant traffic on uplinks greatly. Those saved bandwidth can be employed by external transfers to accelerate their completions. The absolute values depend on the parameter settings of both the network size and multicaists. And in this case, mscTree can reduce the load on fabric uplink to 0.11, which is about six times better than that of both the random and LLSF strategies. For each tree construction strategy, we also compute the ratio of all accepted requests’ total loads on fabric uplinks and their total multicasting rates. Such a ratio indicates the number of spine switches a multicast request involves on average. The results of mscTree, random, and LLSF are 1.1, 6.6, and 6.5, respectively. That is to say, the multicast tree constructed by mscTree requires a few forwarding rules for multicasting since most requests involves only one spine.

IV. CONCLUSION

In this paper, we studied how to compute the maximum possible multicast tree for a one-to-many transfer and investigated how to construct and recover multicast trees in leaf-spine DCN. By using the characteristic of leaf-spine architecture, we proved that the optimization of multicast tree in such a context equals to the minimum set cover problem, and proposed simple algorithms that could greatly reduce the load on uplinks.

REFERENCES