Short Papers

# 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).

### A. Novelty of the Proposed Algorithms

The optimization of multicast tree construction is the well-known NP-hard *directed Steiner problem* [5], [6]. A large number of papers have worked on developing algorithms with good approximation ratios by using either sophisticated greedy designs [5], [7] or Lasserre relaxations of linear program (LP) [6]. Basically, these general-purpose algorithms are either too sophisticated in implementation thus are time-inefficient to deal with the massive number of multicast requests in large-scale data centers [6], or inapplicable in DCN scenes, since all alternative paths are with the equivalent cost (e.g., these shortest path-based heuristics would degenerate to simple random constructions [5], [7]). Distinguished from the prior wisdom, we focus on constructing optimized multicast trees for today's *de facto* leaf-spine DCNs. We prove that finding the optimal multicast tree in leaf-spine DCN is equivalent to the well-investigated NP-hard problem of *minimum set cover* (Theorem 2). Motivated by this, we propose Algorithms 1 and 2, two simple yet near-optimal algorithms, for the construction and reconstruction of trees, based on the greedy set covering [8], respectively.

## 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, $NR$ severs are interconnected with each other and to the external network via $R$ leaf- and $M$ spine- switches (denoted by $V_R$ 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.

Manuscript received January 1, 2019; revised April 3, 2019; accepted May 16, 2019. This work was supported by the Fundamental Research Funds for the Central Universities under Grant 2682019CX61. *(Corresponding author: Shouxi Luo.)*

The authors are with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China (e-mail: sxluo@swjtu.edu.cn; hxx@swjtu.edu.cn; keli@swjtu.edu.cn).

Digital Object Identifier 10.1109/JSYST.2019.2918446

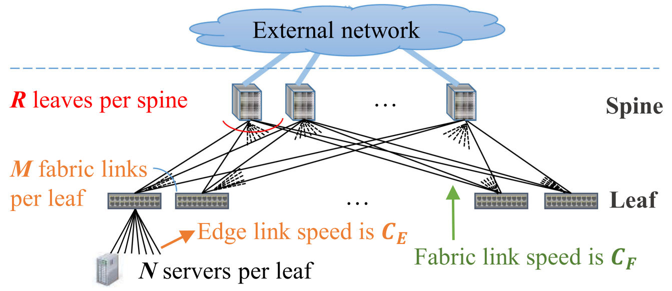


Fig. 1. Leaf-spine DCN architecture. $R$ leaf switches and $M$ spine switches are connected in a full bipartite graph, and each leaf switch is attached with $N$ host servers.

### 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 that 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., $\max_{p\in P_{s,d}} B_p$. 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., $L[s]=L[d]$), there is only a simple path (with two hops) between them, and thus, $\max_{p\in P_{s,d}} = \min(B_{s,L[s]}, B_{L[d],d})$. Otherwise, there are $M$ alternative simple paths (with four hops) crossing over the $M$ spine switches, and thus, $\max_{p\in P_{s,d}} = \max_{o\in V_M}\min(B_{s,L[s]}, B_{L[s],o}, B_{o,L[d]}, B_{L[d],d})$, in which $V_M$ 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 $O(M|D|)$ 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 $L2$, $L3$, and $L4$, 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*.

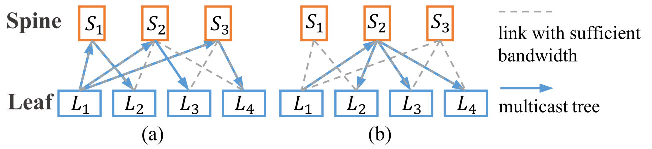


Fig. 2. Example of constructing multicast tree: (b) is better than (a).

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

$$\overline{S}[s,b] = \{j\in V_M : B_{L[s],j}\geq b\} \tag{1}$$

$$\underline{S}[x,b] = \{y\in V_R : B_{x,y}\geq b\} \tag{2}$$

$$L_f = \{L[d] : d\in D \wedge L[d]\neq L[s]\}. \tag{3}$$

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 $\overline{S}[s,b]$. By using a binary variable $x_j$ 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

$$\text{minimize} \quad \sum_{j\in V_M} x_j \tag{4a}$$

$$\text{subject to} \quad x_j\in\{0,1\}, \quad \forall j\in V_M \tag{4b}$$

$$\sum_{j\in\overline{S}[s,b]:k\in\underline{S}[j,b]} x_j \geq 1, \quad \forall k\in L_f. \tag{4c}$$

*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 $j\in\overline{S}[s,b]$, we let $H_j=\underline{S}[j,b]\cap L_f$, denoting the subset of receiver's leaf switches, to whom this spine switch has *b-capacity* downlinks. Then, $\{j\in\overline{S}[s,b] : k\in\underline{S}[j,b]\}$, 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 $\mathbb{H}=\{H_j : \forall j\in\overline{S}[s,b]\}$, the

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

```
    Constants: $V_M$, $V_N$, $V_R$, $L[]$ (the map from $V_N$ to $V_R$)
    Inputs: request $f : \langle s, D, b\rangle$, $B_{u,v}$ for each link
    Outputs: $T$ the constructed tree for $f$
 1: $L_f \leftarrow \{L[d] : d \in D, L[d] \neq L[s]\}$
 2: $\overline{S}[s,b] \leftarrow \{j \in V_M : B_{L[s],j} \geq b\}$
 3: $\mathbb{H} \leftarrow \emptyset$, $\mathbb{H}^* \leftarrow \emptyset$
 4: for all $j \in \overline{S}[s,b]$ do
 5:     $H_j \leftarrow \{k \in L_f : B_{j,k} \geq b\}$
 6:     $\mathbb{H} \leftarrow \mathbb{H} \cup \{H_j\}$
 7: while $L_f \neq \emptyset$ do                    ▷ Greedy Set Cover [8]
 8:     select an $H_j \in \mathbb{H}$ that maximizes $L_f \cap H_j$
 9:     $\pi[j] \leftarrow L_f \cap H_j$
10:     $L_f \leftarrow L_f \setminus H_j$
11:     $\mathbb{H}^* \leftarrow \mathbb{H}^* \cup \{H_j\}$
12:     $\mathbb{H} \leftarrow \mathbb{H} \setminus \{H_j\}$
13: $T \leftarrow \{(s, L[s])\} \cup \{(L[d], d) : d \in D\}$
14: for all $j \in \{j : H_j \in \mathbb{H}^*\}$ do
15:     $T \leftarrow T \cup \{(L[s], j)\} \cup \{(j,k) : k \in \pi[j]\}$
16: return $T$
```

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

$$\text{minimize} \quad \sum_{j:H_j\in\mathbb{H}} x_j \tag{5a}$$

$$\text{subject to} \quad x_j \in \{0,1\}\,, \quad \forall H_j \in \mathbb{H} \tag{5b}$$

$$\sum_{j:k\in H_j} x_j \geq 1, \quad \forall k \in L_f. \tag{5c}$$

While NP-complete, a greedy $O(\sum_{H_j\in\mathbb{H}} |H_j|)$ algorithm provides the good approximation of $H(\max_{H_j\in\mathbb{H}} |H_j|)$ for *minimum set cover*, where $H(n)$ is the $n$th harmonic number, i.e., $H(n) = \sum_{k=1}^{n} \frac{1}{k} \leq \ln n + 1$ [8]. Suppose that the optimal cover involves *opt* spine switches. By repeatedly selecting the spine that contains the largest number of uncovered receivers until all of them get covered, we find a set cover involving $\min(|V_M|, \ln(\max_{H_j\in\mathbb{H}} |H_j|) + 1) \times$ *opt* spines at most. Motivated by this, we develop mscTree to achieve near-optimal tree generation for leaf-spine DCN, thus answering question 2.

*2) Near-Optimal Algorithm:* The procedure of mscTree is as Algorithm 1 shows. Basically, to construct the multicast tree for $f$, mscTree first extracts the qualified spine switches and gets the set of leaf switches of receivers that each of these spine switches can cover (Lines 1–6). Then it employs the *greedy set cover algorithm* [8] to find the near-optimal spine switches (Lines 7–12). During the greedy selection, mscTree also records the set of leaves covered by spine $j$ in $\pi[j]$ (Line 9). Finally, it collects all the selected links to generate the near-optimal multicast tree for $f$ (Lines 13–15).

### C. Tree Reconstruction Upon Network Failure

To be reliable, once switch or link failures occur, the DCN controller must reconstruct trees for these affected multicast requests. The straightforward design is to rerun mscTree for them in turn. However, in case the network is in its high load, these requests might not be satisfied any more. To deal with this, it is reasonable to provide only downgraded multicast bandwidths by scaling their multicast rates down fairly. Indeed, based on Theorem 1 and the proposed mscTree, it is easy to reconstruct trees for affected multicast requests.

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

```
1: For each involved request $f_i \in F$, find its maximum
   multicast rate that the network can provided, i.e., $B^{\max}_{s_i,D_i}$;
2: For request $f_i \in F$, reconstruct its multicast tree based on $T_i$,
   which is the results of mscTree running for demand
   $\langle s_i, D_i, \min(b_i, B^{\max}_{s_i,D_i})\rangle$;
3: For each congested link $(u,v)$, compute the scale factor $\lambda_{u,v}$
   that could remove its congestion; obversely,
   $\lambda_{u,v} = \min(1, B_{u,v}/\sum_{i:f_i\in F\wedge(u,v)\in T_i} b_i)$;
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.
```

Let $F$ be the set of multicast requests broken by the failure, in which the $i$th request is denoted by $f_i : \langle s_i, D_i, b_i\rangle$; 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.

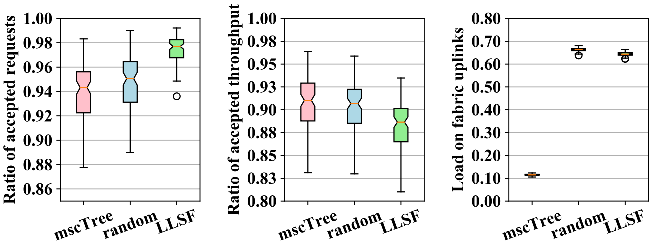
## III. Evaluation

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 $f : \langle s, D, b\rangle$, the task is to construct a tree with available multicast bandwidth $b$, from its sender's leaf switch $s$ to its receives' 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 $b$ 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-first* (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 $\{\alpha 2^i : 0 \leq i \leq n_r\}$ with the probability of $P(b = \alpha 2^i) = \frac{2^{n_r}}{2^{n_r+1}-1}2^{-i}$: $0 < \alpha \ll 1$ is a small positive constant. Besides, the sender's leaf $s$ and receiver's leaves $D$ are randomly chosen, in which the size of $D$ follows the uniform distribution of $U[2, r_{\max}]$. Therefore, the mean multicast throughput of all requests is $(1 + 0.5r_{\max})b_E$, where $b_E = \sum_{i=0}^{n_r} \alpha 2^i * \frac{2^{n_r}}{2^{n_r+1}-1}2^{-i} = \frac{\alpha(n_r+1)2^{n_r}}{2^{n_r+1}-1}$. 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 $\frac{MR}{(1+0.5r_{\max})b_E}$ requests to fulfill the network, then calculate the number of accepted requests, their multicast throughputs, and the corresponding loads on fabric uplinks.



(a) (b) (c)

Fig. 3. mscTree provides larger multicast throughput by accepting more wider requests, and reduces uplink loads greatly. (a) Accpeted 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 multicasts. 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

[1] Z. Guo *et al.*, "On-line multicast scheduling with bounded congestion in fat-tree data center networks," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 1, pp. 102–115, Jan. 2014.

[2] M. Shahbaz *et al.*, "Elmo: Source-routed multicast for cloud services," 2018, arXiv: 1802.09815.

[3] X. Li *et al.*, "Scaling IP multicast on datacenter topologies," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, 2013, pp. 61–72.

[4] J. Cao *et al.*, "Datacast: A scalable and efficient reliable group data delivery service for data centers," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2632–2645, Dec. 2013.

[5] M. Charikar *et al.*, "Approximation algorithms for directed Steiner problems," *J. Algorithms*, vol. 33, no. 1, pp. 73–91, 1999.

[6] T. Rothvoß, "Directed Steiner tree and the Lasserre hierarchy," 2011, arXiv: 1111.5473.

[7] D. Watel *et al.*, "A practical greedy approximation for the directed Steiner tree problem," *J. Combinatorial Optim.*, vol. 32, no. 4, pp. 1327–1370, Nov. 2016.

[8] T. H. Cormen *et al.*, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[9] M. Alizadeh *et al.*, "On the data path performance of leaf-spine datacenter fabrics," in *Proc. IEEE 21st Annu. Symp. High-Perform. Interconnects*, Aug. 2013, pp. 71–74.

[10] A. Singh *et al.*, "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 183–197.