Online Multicast Traffic Engineering for Software-Defined Networks

Sheng-Hao Chiang*, Jian-Jhii Kuo*, Shan-Hsiang Shen†, De-Nian Yang*, and Wen-Tsuen Chen‡§

*Inst. of Information Science, Academia Sinica, Taipei, Taiwan
†Dept. of Information Science, Academia Sinica, Taipei, Taiwan
‡§Dept. of Computer Science, National Tsing Hua University, Hsinchu, Taiwan

{jiang555,lajacky} @iis.sinica.edu.tw, sshen@csie.ntust.edu.tw, {dynyang.chenwt} @iis.sinica.edu.tw

Abstract—Previous research on SDN traffic engineering mostly focuses on static traffic, whereas dynamic traffic, though more practical, has drawn much less attention. Especially, online SDN multicast that supports IETF dynamic group membership (i.e., any user can join or leave at any time) has not been explored. Different from traditional shortest-path trees (SPT) and graph theoretical Steiner trees (ST), which concentrate on routing one tree at any instant, online SDN multicast traffic engineering is more challenging because it needs to support dynamic group membership and optimize a sequence of correlated trees without the knowledge of future join and leave, whereas the scalability of SDN due to limited TCAM is also crucial. In this paper, therefore, we formulate a new optimization problem, named Online Branch-aware Steiner Tree (OBST), to jointly consider the bandwidth consumption, SDN multicast scalability, and rerouting overhead. We prove that OBST is NP-hard and does not have a \(\frac{D_{\max}}{D_{\max}^*}\)-competitive algorithm for any \(\epsilon > 0\), where \(D_{\max}\) is the largest group size at any time. We design a \(\frac{D_{\max}}{D_{\max}^*}\)-competitive algorithm equipped with the notion of the budget, the deposit, and Reference Tree to achieve the tightest bound. The simulations and implementation on real SDNs with YouTube traffic manifest that the total cost can be reduced by at least 25% compared with SPT and ST, and the computation time is small for massive SDN.

I. INTRODUCTION

Software-defined networking (SDN) provides a promising architecture for flexible network resource allocations to support a massive amount of data transmission [1]. SDN separates the control plane and data plane, allowing the control plane to be programmable to efficiently optimize the network resources. OpenFlow [1] in SDN includes two major components: controllers (SDN-Cs) and forwarding elements (SDN-FEs). Controllers derive and install forwarding rules according to different policies in the control plane, whereas forwarding elements in switches then deliver packets according to the forwarding rules. Compared with the current Internet techniques/multicast, routing paths no longer need to be the shortest ones, and the paths can be embedded more flexibly inside the network. Since SDN provides a better overview of network topologies and enables centralized computation, traditional theoretical studies on SDN traffic engineering mostly focus on static traffic flows. Recently, allocations of online dynamic unicast have drawn increasing attention [2], [3]. However, online multicast traffic engineering that supports IETF dynamic group membership [4] in the current Internet multicast has not been explored for SDN [5]. Dynamic group membership plays a crucial role in practical multicast services because it allows any user to join and leave a group at any time (e.g., for a conference call or video broadcast).

Compared to unicast, multicast has been demonstrated in empirical studies to effectively reduce the overall bandwidth consumption by around 50% [5]. It employs a multicast tree, instead of disjoint unicast paths, from the source to span all destinations of a multicast group. The current Internet multicast standard, i.e., PIM-SM [6], employs a shortest-path tree (SPT), and therefore does not support traffic engineering since the path from the source to each destination is fixed to be the shortest one. SPT tends to lose many good opportunities to reduce the bandwidth consumption by sharing more common edges among the paths to different destinations. In contrast, a Steiner tree (ST) [7] in Graph Theory provides flexible routing to minimize the bandwidth consumption (i.e., total number of edges). Nevertheless, ST is not designed for online multicast with dynamic group membership because 1) re-computing the Steiner tree after any joins or leaves is computationally intensive, especially for a large group, and 2) the overhead (e.g., installing new rules to modify Group Table in OpenFlow) incurred from rerouting the previous Steiner tree to the latest one is not minimized.

Moreover, the scalability problem for SDN multicast is much more serious since for each network with \(n\) nodes, the number of possible multicast groups is \(O(2^n)\), whereas the number of possible unicast connections is only \(O(n^2)\). Therefore, it is more difficult for SDN-FEs to store the forwarding entries of all multicast groups in OpenFlow Group Table due to the limited TCAM size [8], [9]. To remedy this issue, a promising way is to exploit the branch forwarding technique [10], [12], which stores the multicast forwarding entries in branch nodes (with at least three incident edges), interestingly, clones and forwards corresponding packets to the output ports.

1To implement multicast, the SDN controller installs a rule with multiple output ports in its action field into each branch node. Then, the branch node clones and forwards corresponding packets to the output ports.

2SDN switches match more fields in packets that implies that the length of each rule entry is much longer than traditional switches, so SDN switches can support fewer rule entries with the same TCAM size.

This work is supported by Ministry of Science and Technology of Taiwan under contracts No. MOST 105-2622-8-009-008, No. MOST 105-2221-E-001-001- and No. MOST 105-2221-E-001-015-MY3 and by Academia Sinica Thematic Research Grant.
instead of every node of a tree. It can effectively remedy the multicast scalability problem since packets are forwarded in a unicast tunnel from the logic port of a branch node in SDN-FE \(^1\) to another branch node. All nodes in the unicast tunnel are no longer necessary to maintain the forwarding entry of the multicast group. Nevertheless, the number and positions of branch nodes need to be carefully selected, but those important factors are not examined in SPT and ST.

Different from traditional traffic engineering for static traffic, for scalable online multicast in SDN, it is necessary to minimize not only the tree size and branch node number of a tree at any instant but also the rerouting overhead to tailor a sequence of trees over time, while the future user joining and leaving are unknown during the rerouting. In this paper, therefore, we formulate a new optimization problem, named Online Branch-aware Steiner Tree (OBST), for SDNs with dynamic group membership to consider the above factors. We prove that OBST is NP-hard. Note that traditional ST problem is also NP-hard but approximable within 1.39 \(^1\). However, OBST is more challenging because all costs from a sequence of correlated trees need to be carefully examined. Indeed, we prove that for any \(\epsilon > 0\), there is no \(|D_{\text{max}}|^{1-\epsilon}\)-competitive algorithm for OBST unless \(P = \text{NP}\), where \(D_{\text{max}}\) is the largest destination set at any time. For OBST, we design a \(|D_{\text{max}}|\)-competitive algorithm\(^5\) called Online Branch-aware Steiner Tree Algorithm (OBSTA)\(^6\) to achieve the tightest bound. To carefully examine the temporal correlation (which has not been exploited in all static multicast algorithms) of trees, we introduce the notion of \(\text{budget}\) and \(\text{deposit}\), and a larger budget and deposit allows OBSTA to reroute the tree more aggressively for constructing a more bandwidth efficient tree. Moreover, we construct Reference Tree for OBSTA to guide the rerouting process for achieving the performance bound. Simulation and implementation in a real SDN with YouTube traffic manifest that OBSTA can reduce the total cost by 25% compared with SPT and ST.

The rest of this paper is organized as follows. Section II introduces the related work. Sections III and IV formally define OBST and describe the hardness results. We present the algorithm design of OBSTA in Section V and Section VI shows the simulation and implementation results on real topologies. Finally, Section VII concludes this paper.

II. RELATED WORK

Traffic engineering for unicast in SDN has attracted a wide spectrum of attention. McKeown et al. \(^1\) study the performance of OpenFlow in heterogeneous SDN switches. Jain et al. \(^16\) develop private WAN of Google Inc. with the SDN architecture. Agarwal et al. \(^17\) present unicast traffic engineering in an SDN network with only a few SDN-FEs, while other routers followed a standard routing protocol, such as OSPF. By leveraging SDN and reconfigurable optical devices, Jin et al. \(^18\) design centralized systems to jointly control the network layer and the optical layer, while Jia et al. \(^19\) present approximation algorithms for transfer scheduling. Gay et al. \(^20\) study traffic engineering with segment routing in wide area networks with network failures. Qazi et al. \(^21\) design a new system to effectively control the SDN middleboxes. Cohen et al. \(^22\) maximize the network throughput with limited sizes on forwarding tables. However, the above studies only focus on unicast traffic engineering, and multicast traffic engineering in SDN has attracted much less attention.

For multicast, the current standard IETF PIM-SM \(^6\), which is designed to support the dynamic multicast group membership in IETF IGMP \(^4\), relies on unicast routing protocols to discover the shortest paths from the source to the destinations for building a shortest-path tree (SPT). However, SPT is not designed to support traffic engineering. Although the Steiner tree (ST)\(^7\) minimizes the tree cost, ST is computationally intensive and is not adopted in the current Internet standard. Overlay ST \(^23\), \(^24\), on the other hand, is an alternative to constructing a bandwidth-efficient multicast tree in the P2P environment. However, the path between any two P2P clients is still a shortest path in Internet. Recently, Huang et al. \(^25\) optimized the routing of multicast trees in SDN \(^9\), \(^25\), \(^26\). Zhu et al. \(^27\) study the multicast tree with virtualized software switches. However, the above studies are designed for static multicast trees at any instant, instead of optimizing a sequence of dynamic trees with varying destinations.

For dynamic traffic, dynamic unicast has been explored in \(^2\), \(^28\)–\(^30\) to rerouting the traffic for failure recovery and power saving (i.e., to turn off more links and nodes). Previous literature on dynamic multicast focuses on protocol and architecture design, instead of theoretical study, for other kinds of networks. Baddi et al. \(^31\) revise PIM-SM to support multicast join, leave, and handover in mobile wireless networks. Markowski \(^32\) considers spectrum and modulation setup for dynamic multicast in optical networks. Xing et al. \(^33\) explore dynamic multicast with network coding to minimize the coding cost. However, the dynamic group membership in IETF IGMP has not been considered in any SDN theoretical research for multicast traffic engineering.

III. PROBLEM FORMULATION

In this paper, we formulate a new online multicast problem, namely Online Branch-aware Steiner Tree (OBST). It jointly considers tree routing, rerouting, and scalability in SDNs. More specifically, the network is modeled as a weighted graph \(G = \{V, E\}\) with a source \(s \in V\) and a set of candidate destinations \(D \subseteq V\), where \(w(e)\) is the weight of edge \(e \in E\), respectively. A larger weight can be assigned to a link with the higher delay and higher loss rate (due to congestion) \(^34\). For online multicast, each destination is allowed to join and leave at the beginning of each time
slot $t_i$ and let $D_i$ denote the set of destinations during $t_i$, where $D_i \subseteq D$ and $D_{\text{max}} = \arg \max_{D_i} \{|D_i|\}$. At $t_i$, the SDN controller builds a multicast tree $T_i$ by tailoring $T_{i-1}$ to connect the source $s$ and the destinations in $D_i$, where the future information $D_j$ is unknown for every $j > i$. The tree cost $w(T_i)$ is the sum of edge weights in the multicast tree $T_i$ (i.e., $w(T_i) = \sum_{e \in T_i} w(e)$). To consider the scalability of SDN (i.e., TCAMs), the branch cost $b_T$ in $t_i$ is the total number (or the total weighted number) of branch nodes in $T_i$, where each branch node has at least three neighbors in $T_i$. Moreover, at $t_i$, for the destinations in $t_{i-1}$ that will also stay in this time slot, it is necessary to reroute the tree spanning them because the tree routing $T_{i-1}$ may no longer be bandwidth efficient. To consider the rerouting overhead, we first define the operation pruning of a tree and the symmetric difference of two graphs, and then define the rerouting cost $rc_i$ in $t_i$ according to [37].

**Definition 1.** Given a node set $L \subseteq D$ and a tree $T$ spanning the destinations in $D$, the operation, pruning, denoted by $T \setminus L$, outputs a maximum subtree of $T$, where each leaf is either a destination in $D \setminus L$ or the source of $T$.

**Definition 2.** Given two graphs $G_1$, $G_2$, the symmetric difference $G_1 \triangle G_2$ of the two graphs is the graph with the edges in $G_1 \cup G_2$ but not in $G_1 \cap G_2$.

In other words, $T \setminus L$ considers the case that destination set $L$ leaves the tree, and the path connecting each $d \in L$ to the closest branch node is thereby removed from $T$. Moreover, each edge in $G_1 \triangle G_2$ appears in either $G_1$ or $G_2$.

**Definition 3.** The rerouting cost in $t_i$ is the total edge cost of the symmetric difference $(T_{i-1} \ominus (D_{i-1} \setminus D_i)) \triangle (T_i \ominus (D_i \setminus D_{i-1}))$. In other words, $rc_i = \sum_{e \in (T_{i-1} \ominus (D_{i-1} \setminus D_i)) \triangle (T_i \ominus (D_i \setminus D_{i-1}))} w(e)$.

The first part of the symmetric difference represents the old tree without the leaving destinations, and the second part is the new tree without the joining users. Therefore, the symmetric difference is the tree topology change (i.e., rerouting efforts) for the destinations staying in both time slots.

**Example 1.** Fig. 1 presents an example of the rerouting cost for $d_1, d_2 \in D_{i-1}$ and $d_3 \in D_i$. Figs. 1(a) and 1(d) show $T_{i-1}$ and $T_i$, respectively. Figs. 1(b) and 1(c) show the old tree after $d_1$ leaves and the new tree if $d_3$ was not a destination, respectively. Fig. 1(e) shows the symmetric difference with $rc_i = 23$. It reroutes $d_2$ by removing the old path in Fig. 1(b) and adding the new path in Fig. 1(c).

**Definition 4.** Given $G = \{V,E\}$, a source $s \in V$, and a collection of destination sets, $\{D_1,D_2,\ldots,D_n\}$ for time $t_1, t_2, \ldots, t_n$, the OBST problem finds a set of trees $T = \{T_1, \ldots, T_n\}$ such that 1) $s$ is connected to all destinations in $D_i$ via $T_i$ for every $t_i$, and 2) the total cost $\sum_{i=1}^{n} (w(T_i) + \alpha \times b_T + \beta \times \rho_{ci})$ is minimized, where $\alpha \geq 0$ and $\beta \geq 0$.

In the offline OBST problem, $D_i$ is available at any $t_i$ with $i < j$, but the above information is not available in the online OBST problem studied in this paper.

Therefore, OBST aims to find a series of multicast trees such that the weighted sum of the overall tree costs, the branch costs, and rerouting costs is minimized, where $\alpha$ and $\beta$ are tuning knobs [9], [25], [26] for network operators to differentiate the importance of network load, scalability, and rerouting overhead. Compared with the SPT and ST, OBST not only provides the flexible routing of trees but also carefully examines the practical issues of OpenFlow in dynamic SDNs.

Fig. 2 presents an example to compare SPT, ST, and OBST with one source $s$ and two destinations $d_1$ and $d_2$ in Fig. 2(a), where $D_1 = \{d_1\}$, $D_2 = \{d_1,d_2\}$, $\alpha = 1$, and $\beta = 0.2$. SPT, ST, and OBST are the same in $t_1$ with the total cost as $7.5 + 2.5 + 1 \times 1 + 0.2 \times 0 = 11$ in Fig. 2(b). However, after $d_2$ joins, SPT becomes $(7.5+2.5+6.2+4)+1 \times 1 + 0.2 \times 0 = 21.2$ in Fig. 2(c), and ST becomes $(6.2+4+4)+1 \times 2 + 0.2 \times (7.5+2.5+6.2+4) = 20.24$ in Fig. 2(d). In contrast, OBST only incurs $(7.5+2.5+2+4)+1 \times 2 + 0.2 \times 0 = 18$ in Fig. 2(e). Note that OBST does not choose an alternative $(7.5+2.5+6)+1 \times 1 + 0.2 \times 0 = 17$ in Fig. 2(f) because if $d_1$ leaves the tree, $d_2$ is inclined to induce dramatic rerouting.

**IV. HARDNESS RESULTS**

The OBST problem is more challenging than the Steiner tree problem since ST does not consider the SDN scalability and examine the temporal correlation of trees. In the following, we first show that OBST is very challenging in Complexity Theory by proving that the offline OBST problem does not

---

**Table in OpenFlow.**
have a $|D_{\text{max}}|^{1-\epsilon}$-approximation algorithm for any $\epsilon > 0$ with a gap-introducing reduction from the Hamiltonian Path (HP) problem, where $D_{\text{max}}$ is the largest destination set at any time. Afterward, we show that the online OBST problem does not have a $|D_{\text{max}}|^{1-\epsilon}$-competitive algorithm.

**Definition 5.** Given any graph $G_H = \{V_H, E_H\}$ and a vertex $y \in V_H$, the HP problem finds a path starting from $y$ to visit every other vertex exactly once.

**Theorem 1.** For any $\epsilon > 0$, no $|D_{\text{max}}|^{1-\epsilon}$-approximation algorithm exists for the offline OBST problem unless P = NP.

**Proof.** We prove the theorem with a gap-introducing reduction from the HP problem. For any instance $G_H$, we construct an OBST instance $G$ with the following goals: 1) if $G_H$ has a Hamiltonian path starting at $y$, $\text{OPT}(G) \leq 2\alpha m(m^{p+1} + 1)$; 2) otherwise, $\text{OPT}(G) > 2\alpha m(m^{p+1})|D_{\text{max}}|^{1-\epsilon}$, where $m$ is the number of nodes in $G_H$ and $\text{OPT}(G)$ is the optimal OBST. Specifically, we first create a source node $s$ and $m$ clones of $G_H$, namely $G_{H,1}, G_{H,2}, ..., G_{H,m}$, and add them into $G$, where $p$ denotes the smallest integer greater than $\frac{1}{1+\log_m 2}$. Each clone node of $y$ is then connected to $s$. The cost of every edge in $G$ is 1, and $\alpha$ and $\beta$ are set to be any number no smaller than $\frac{m^p(m^{p+1}+1)}{2}$ and 0, respectively.

Every node (except $s$) in $G$ joins and leaves the multicast group as follows. We first generate an arbitrary permutation $\mathcal{R}$ for all nodes in $G_H$. Let $\mathcal{R}[j]$ denote the $j$-th node in $\mathcal{R}$, where $1 \leq j \leq m$. Then, in each $t_i$ with $1 \leq i \leq m^{p+1}$, the clone node in $G_{H,i/m}$ of $\mathcal{R}[(i-1) \mod m] + 1$ will join the multicast group. In $t_{mp+1}$, all nodes are in the multicast group, and each node then leaves the group one-by-one in each subsequent time slot after $t_{mp+1+1}$ in a reverse order (i.e., first-in-last-out). The total number of time slots is $2m^{p+1}$. We first prove the first goal (i.e., the sufficient condition). If $G_H$ has a Hamiltonian path starting at $y$, there is a path starting from $s$ that visits every destination in $D_m = G_{H,1}$ exactly once in $t_m$, and the source must be in the tree such that $b_{t_1} \geq 1$. That is, the cost of the above path is at most $m + \alpha$. Since $\beta = 0$, the smallest cost in $t_m$ is at most $m + \alpha$. Similarly, the smallest costs in $t_{2m+1} - t_{3m+1}$, ..., $t_{mp+1}$-th are at most $2m + \alpha, 3m + \alpha, ..., m^{p+1} + \alpha$, respectively. Moreover, because the group size grows until $t_{mp+1}$, the smallest cost in $t_i$ must be no greater than that of $t_{i/m \times m}$, where $1 \leq i \leq m^{p+1}$. Thus, let $\text{OPT}(G_i)$ denote the smallest cost of $t_i$, $\text{OPT}(G) = \sum_{i=1}^{m^{p+1}} \text{OPT}(G_i) = 2 \sum_{i=1}^{m^{p+1}} \text{OPT}(G_i) 
\leq 2[1 + 2 + ... + m^p]m^2 + m^{p+1}\alpha 
= m^{p+1}(m^{p+1} + 1) + 2\alpha m^{p+1} \leq 2\alpha m^{p+1}$.

We then prove the second goal (i.e., the necessary condition). If $G_H$ does not have a Hamiltonian path starting at $y$, for each time $t_i$ with $1 \leq i < m$, the cost is at least $i + \alpha$. In $t_m$, there exists at least one branch node, and $\text{OPT}(G_m) \geq m + 2\alpha$. Similarly, the smallest cost in $t_i$ must be at least $i + (\lfloor i/m \rfloor + 1) \times \alpha$, where $1 \leq i \leq m^{p+1}$. Therefore, $\text{OPT}(G) = 2 \sum_{i=1}^{m^{p+1}} (\text{OPT}(G_i)) 
\geq 2 \sum_{i=1}^{m^{p+1}} (i + (\lfloor i/m \rfloor + 1) \times \alpha) 
\geq 2m^p \frac{m(m^{p}+1)}{2} + \alpha m + m^{p+1}(m^{p+1} - 1)(m + \alpha) 
> \alpha m^{p+1}(m^{p+1} + 1) + 2\alpha m^{p+1} + 1)m^{p+1} \log_m 2 
= 2\alpha m^{p+1} + 1)m^{p+1} \log_m 2 
\geq 2\alpha m^{p+1} + 1)m^{p+1}(1-\epsilon) = 2\alpha m^{p+1} + 1)|D_{\text{max}}|^{1-\epsilon}$.

Since $\epsilon$ can be arbitrarily small, for any $\epsilon > 0$, there is no $|D_{\text{max}}|^{1-\epsilon}$-approximation algorithm unless P = NP.

Since any $c$-approximation algorithm for an online problem is also a $c$-approximation algorithm for the offline problem [15], we have the following corollary.

**Corollary 1.** For any $\epsilon > 0$, no $|D_{\text{max}}|^{1-\epsilon}$-competitive algorithm exists for the online OBST problem unless P = NP.

V. ALGORITHM DESIGN OF OBSTA

In the following, we propose a $|D_{\text{max}}|$-competitive algorithm, called Online Branch-aware Steiner Tree Algorithm (OBSTA), for OBST. Different from SPT and ST that focus on a static tree, we introduce the notion of budget $B_t$ and deposit $dep_t$ for each $t_t$ to properly optimize a sequence of correlated trees over time. Intuitively, with a larger budget and deposit, OBSTA is able to reroute the tree more aggressively for constructing a more bandwidth efficient tree in $t_{i+1}$. Nevertheless, the rerouting cost also grows in this case, and it is thus necessary to carefully set a proper budget in order to achieve the performance bound (i.e., competitive ratio). Also, the remaining budget $B_t$ can be saved to deposit $dep_t$, so that $B_{t+1} + dep_t$ can be exploited to reroute the tree in $t_{i+1}$. More specifically, to achieve the competitive ratio, the lower bound of the optimal solution is exploited to derive the budget. The lower bound is the longest shortest path among all source-destination pairs plus the cost of one branch node (proved later). Thus, we set the budget in $t_i$ as $B_i = \sum_{d \in D_i} dist(G(s,d) + \alpha \times |D_i|$. On the other hand, the deposit represents the remaining unused budget, i.e., $dep_t = dep_{t-1} + B_t - w(T_t) - \alpha \times b_T - \beta \times rc_t$. Also, we derive a potential rerouting cost $prec_i$ at each $t_i$ for OBSTA to estimate the worst-case rerouting effort in the future. In the worst case, when all the other destinations in the same subtree of $d$ leave the group, $d$ will connect to $s$ via the shortest path because another alternate zigzag incurs a high cost. Therefore, $prec_i = \sum_{e \in U_{d} D_i} (P_A(s,d) \Delta P_G(s,d)) w(e)$, where $P_A(s,d)$ denotes the shortest path between $s$ and $d$ in the graph $A$. OBSTA includes a policy to save a sufficient deposit $dep_t \geq \beta \times prec_t$ for the future. Moreover, to reduce the rerouting cost, our idea is to organize stable destinations (i.e.,
the destinations staying with longer time in a bandwidth-efficient stable tree, called Reference Tree (RT), and other destinations are then rerouted and attached to RT in OBSTA. Later, we show that the budget, deposition, potential rerouting cost, and RT are the cornerstones of OBSTA to achieve the tightest performance bound. Section 7 also manifests that OBSTA outperforms SPT and ST because temporal information is not defined and exploited in SPT and ST.

A. Algorithm Description

For each time \( t_i \), OBSTA includes the following four phases: 1) Reference Tree (RT) Construction, 2) Candidate Deployed Tree (DT) Generation, 3) Candidate DT Patching, and 4) Final DT Selection. More specifically, RT Construction first builds a bandwidth-efficient tree for stable destinations as a reference, whereas DT is the real multicast tree deployed in SDNs. Candidate DT Generate creates several candidate DTs, and based on RT, Candidate DT Patch reroutes and attaches different remaining destinations to create several candidate final solutions, in order to effectively avoid unnecessary zigzag paths in the tree. The final phase then chooses the one with the minimum cost. To achieve the performance bound, it is important for Candidate DT Patching to ensure that the current deposit is sufficient to support rerouting in the future. The pseudo-code is presented in Appendix A.

1) Reference Tree (RT) Construction: At the beginning of each \( t_i \), RT Construction identifies a set of stable destinations to construct a reference tree \( RT_i \). Let \( \tau_i \) denote the number of stable destinations in \( RT_i \). In \( t_i \) with no destination, only \( s \) is included in \( RT_i \) with \( \tau_i = 0 \). Afterward, for each \( t_i \) with \( i > 1 \), to achieve the performance bound, OBSTA selects \( \tau_i \) stable destinations from [38] and then builds \( RT_i \) rooted at \( s \) with a subtree of \( SPT(D) \) that spans only the \( \tau_i \) destinations, where \( SPT(D) \) is the shortest path tree spanning all candidate destinations in \( D \). More specifically, to evaluate the degree of stability for each destination, the Stability Index (SI) [38] of a node \( d \in D \) is defined as follows,

\[
SI(d) = \begin{cases} 
2h_d & \text{if } h_d \leq \frac{n-a_d}{2}, \\
1 & \text{otherwise},
\end{cases}
\]

where \( n, a_d \), and \( h_d \) denote the number of time slots for the multicast, the first arrival time slot of \( d \), and the duration in the group of \( d \), respectively. A destination \( d \) with an \( SI(d) > H \) will be selected as the stable destination, where \( H \) is a threshold tuned by the network operators. Thus, \( \tau_i = |\{d \in D : SI(d) > H\}| \).

2) Candidate Deployed Tree (DT) Generation: In contrast to RT \( RT_i \), DT \( T_i \) is the solution tree to be deployed in SDNs at \( t_i \). Therefore, for the destinations leaving in the beginning of \( t_i \), OBSTA first removes them from \( T_i - 1 \) by pruning the edges not on a path between the source and any staying destination, in order to create an initial DT \( T_i,leave = T_i - 1 \ominus (D_i - 1 \setminus D_i) \) in \( t_i \). Afterward, OBSTA sorts the destinations in \( T_i,leave \) in a non-decreasing order of the distance between \( s \) and each destination to find the sorted list \( (q_1, q_2, q_3, \ldots, q_k) \), where \( k = |D_i - 1 \cap D_i| \). Finally, the candidate DT set is \( \{T_i^1, T_i^2, \ldots, T_i^k\} \), where each \( T_i^l \) includes the \( l \) closest destinations from the source, (i.e., \( (q_1, \ldots, q_l) \)). For each candidate DT, other destinations not in the tree will be rerouted and attached to the tree in the next phase as much as possible. In other words, OBSTA is designed to explore different intermediate trees with various rerouting strategies to achieve the performance bound.

More specifically, OBSTA derives the routing of each candidate DT by spraying, i.e., \( T_i^k = T_i,leave \otimes (q_1, q_2, \ldots, q_k) \) and \( T_i^0 = s \), defined as follows.

Definition 6. For any tree \( A = \{V_A, E_A\} \) with \( V_A \) and \( E_A \) as the vertex and edge sets, respectively, let \( P_A(s, t) \) denote the path from \( s \) to \( t \) in \( A \). Given an arbitrary sequence \( (a_1, a_2, \ldots, a_k) \) of nodes in any subset of \( V_A \), where \( k \leq |V_A| \), spraying \( A \otimes (a_1, a_2, \ldots, a_k) \) outputs a collection of subtrees of \( A \), denoted by \( (A_1, \ldots, A_k) \), where each subtree \( A_k = \bigcup_{l \leq k} P_A(s, a_l) \), and \( A_1 \subseteq A_2 \subseteq \ldots \subseteq A_k \subseteq A \).

Example 2. Fig. 3 presents an example of spraying. Fig. 3(a) shows \( A \) with a sequence of nodes \( (a_1, a_2, a_3) \). The bold trees in Figs. 3(b), 3(c) and 3(d) are subtrees \( A_1, A_2 \) and \( A_3 \), respectively, where \( A_1 = P_A(s, a_1) \), \( A_2 = P_A(s, a_1) \cup P_A(s, a_2) \) and \( A_3 = P_A(s, a_1) \cup P_A(s, a_2) \cup P_A(s, a_3) \). Fig. 4 presents an example of Candidate DT Generation, where Fig. 4(a) shows \( T_i - 1 \). After removing the leaving destinations (red nodes), Fig. 4(b) shows \( T_i,leave \), and the six candidate DTs (i.e., \( T_i^0, T_i^1, T_i^2, T_i^3, T_i^4, T_i^5 \)) are generated, where \( T_i^3 \) is shown in Fig. 4(c).

3) Candidate Deployed Tree (DT) Patching: To generate candidate final solutions, this phase adds new joining destinations and reroutes the destinations (not selected in the previous phase) to the candidate DTs according to RT, whereas the sufficient deposit is required to be followed. For ease of presentation, we first define the operation, grafting, as follows.

Definition 7. Given two trees \( A \) and \( B \) rooted at \( s \), grafting \( A \oplus B \) outputs a tree spanning all destinations in \( A \) and \( B \) as follows: 1) For each destination \( d \in B \) and each shortest path
the stable nodes in $T$ denote the set of staying destinations that are not included in $P_B(u,d)$. We find the smallest one $P_B(u,d,d)$ such that every node between $d$ and $u$ in $P_B(u,d,d)$ is not in $A$. 2) $A \oplus B = A \cup \bigcup_{d \in D_B} P_B(u,d,d)$, where $D_B$ denotes the destination set in $B$.

**Example 3.** Fig. 5 presents an example of grafting. Fig. 5(a) shows tree $A$, and Fig. 5(b) is $B$ with six destinations. We first find every path $P_B(u,d,d)$ for each destination $d$ in $B$ as shown in Fig. 5(c). Nodes $d_1$ and $d_2$ both connect to $u_3$ in $A$, and nodes $d_3$ and $d_4$ both connect to $u_2$ in $A$. Since node $d_5$ is in $A \cap B$, the path contains only $d_5$. Node $d_6$ connects to node $u_3$. Fig. 5(d) shows $A \oplus B$, and $A \subseteq A \oplus B$.

Then, let $D_i,join$ denote the set of destinations joining in the beginning of $i$. For each candidate DT $T_i$, let $D_i,rest$ denote the set of staying destinations that are not included in $T_i$ in the previous phase. Let $D_i,attach$ denote the destination set containing $D_i,join$ and $D_i,rest$. Let $D_i,stable$ represent the stable nodes in $D_i,attach$. In the following, we first attach the destinations in $D_i,stable$ to form a new source $T_i$. Let $RT_i(D_i,stable)$ denote the minimum-cost subtree of $RT_i$ to span $D_i,stable$. OBSTA first sorts the nodes in $D_i,stable$ in a non-decreasing order of SI (see RT Construction phase) as $(q_1, q_2, ..., q_k)$, where $k = |D_i,stable|$. Then, to patch each candidate DT $T_i$, OBSTA iteratively reroutes and attaches the closest stable destinations according RT to generate $(T_i,stable), ..., T_{i,k} = (RT_i(D_i,stable) \otimes (q_1, q_2, ..., q_k)) \oplus T_i$. In other words, OBSTA attaches (i.e., grafts) the destinations $(q_1, q_2, ..., q_k)$ to each candidate tree $T_i$ according to the sprouting of reference tree $RT_i$ and $D_i,stable$. Afterward, for those $T_i,stable$ having sufficient deposit, OBSTA selects the one with the maximum $m$ and sets the patched candidate DT $T_i,stable$ as $T_i,stable$. Note that $T_i,stable = T_i$ if $m = 0$.

**Example 4.** Fig. 6 presents an example of the Candidate DT Patching for stable destination $d_1$, where Figs. 6(a) and 6(b) are RT and the candidate DT, respectively. OBSTA reroutes $d_1$ to the DT with the path in RT shown in Fig. 6(d).

Afterward, we attach the rest $D_i,rest$ of the destinations (i.e., $D_i - D_i,stable$) to candidate DT $T_i,stable$ with operation contraction defined as follows.

**Definition 8.** Given a graph $A = \{V_A, E_A\}$ with a connected subgraph $B = \{V_B, E_B\}$, contraction $A \cdot B$ outputs a graph $A'$ by contracting (i.e., merging) the nodes in $B$ into one vertex $u$. For each neighbor $v \in V_A \cap V_B$ of $u$, $A \cdot B$ includes only the edge with $\min_{e \in E_{u,v}} \{ w(e) \}$, where $E_{u,v}$ denotes the set of edges between $u$ and $v$.

**Example 5.** Fig. 7 shows the A with a connected subgraph B (the bold-line tree). Fig. 7(b) shows the contracted graph $A'$ with all vertices in $B$ merged to $s'$.

OBSTA first builds a complete graph $G'$ consisting of the nodes in $D_i,rest$ and a new source $s'$, by contracting the nodes of $T_i,stable$ in $G$ to form a new source $s'$ and assigning each edge weight $e_{u,v}$ in $G'$ as the shortest path length between nodes $u$ and $v$ in the contracted G. Then, OBSTA constructs a minimum spanning tree $MST(G')$ rooted at $s'$ to span all destinations in $D_i,rest$. Then, OBSTA grafts $SPT(D_i,rest)$ on $T_i,stable$ (i.e., $T_i,stable \oplus SPT(D_i,rest)$) to find the complete tree $T_i,rest\cdot SPT$. In this case, if $T_i,rest\cdot SPT$ has sufficient deposit, OBSTA picks the candidate DT $T_i$ as $T_i,rest\cdot SPT$. Otherwise, OBSTA constructs another tree $T_i,rest\cdot SPT$ rooted at $s$ and extracted from $SPT(D)$ to span all destinations in $D_i,rest$. Then, OBSTA grafts $SPT(D_i,rest)$ on $T_i,stable$ (i.e., $T_i,stable \oplus SPT(D_i,rest)$) to find the complete tree $T_i,rest\cdot SPT$. In this case, if $T_i,rest\cdot SPT$ has sufficient deposit, OBSTA picks the candidate DT $T_i$ as $T_i,rest\cdot SPT$. Otherwise, OBSTA discards the candidate DT $T_i$.

**Example 6.** Fig. 7 presents an example of Candidate DT Patching for $d_2, d_3, d_4$ that have not been included in $T_i$ generated in Candidate DT Generation. Fig. 7(a) shows $G$ and the current candidate DT $T_i,stable$ including $d_1, d_5, d_6$. OBSTA first contracts $T_i,stable$ to $s'$ in Fig. 7(b) and constructs the complete graph $G'$ in Fig. 7(c). The constructed spanning tree $d_2, d_3, d_4$ (with bold lines) in $G'$ is in Fig. 7(c). Then, OBSTA reverts $G'$ to $G$ to obtain $T_i,stable\cdot MST$ in Fig. 7(d). If $T_i,stable\cdot MST$ does not have sufficient deposit, OBSTA constructs another tree $T_i,rest\cdot SPT$ by $T_i,stable \oplus SPT(D_i,rest)$.
Thus, there is at least one candidate DT with the sufficient deposit.

To prove by induction, we assume that the deposit at node $i$ is non-negative and prove that the deposit at node $i$ is non-negative (i.e., $d_{0,i} \geq 0$), where $i \geq 1$.

Candidate DT Generation creates at least one candidate DT $T_0^i$ with only the source $s$ and deposit $d_{0,i} = d_{i-1} + B_0^i - w(T_0^i) - \alpha \times b_{T_i} - \beta \times r_{c_i}$. Note that $B_0^i = \alpha$, $w(T_0^i) = 0$, $b_{T_i} = 1$, and $r_{c_i} = 0$. Thus, $d_{0,i} = d_{i-1}$. Moreover, because only $s$ is in $T_0^i$, there is no destination, and $prc_i = 0$.

Thus, $d_{0,i} = d_{i-1} \geq 0 = \beta \times \frac{r_{c_i}}{prc_i}$. Next, the first part of Candidate DT Patching can always generate a candidate DT ($T_{i,stable}^0$) with the sufficient deposit because at least one candidate DT (e.g., $T^0_i$) has sufficient deposit by Candidate DT Generation.

Subsequently, we prove (by contradiction) that the candidate DT $T_{i,stable}^0$ (i.e., $T_{i,rest\rightarrow SPT}$) generated by the second part of Candidate DT Patching must have the sufficient deposit. Assume that candidate DT $T_{i,stable}^0$ (i.e., $T_{i,stable}^0$) does not have the sufficient deposit. That is, there exists a set $D_{i,rest}^0$ such that $T_{i,stable}^0 \oplus SPT(D_{i,rest}^0)$ has the insufficient deposit. Since $T_{i,stable}^0$ has the sufficient deposit, we have two following inequalities representing the deposits before and after $T_{i,stable}^0 \oplus SPT(D_{i,rest}^0)$, respectively. Note that we use hat symbol to mark the one after $T_{i,stable}^0 \oplus SPT(D_{i,rest}^0)$.

- $d_{0,i} = d_{i-1} + B_1^i - w(T_{i,stable}^0) - \alpha \times b_{T_i} - \beta \times r_{c_i} \geq \beta \times \frac{r_{c_i}}{prc_i}$ (1)
- $d_{0,i} = d_{i-1} + B_1^i - w(T_{i,rest\rightarrow SPT}^0) - \alpha \times b_{T_i} - \beta \times r_{c_i} < \beta \times \frac{r_{c_i}}{prc_i}$ (2)

Since the budget increases by $\alpha \times |D_{i,rest}^0| + \sum_{x \in D_{i,rest}^0} dist_G(s, x)$, the tree cost grows by at most $\sum_{x \in D_{i,rest}^0} dist_G(s, x)$, and the branch cost increases by at most $|D_{i,rest}^0|$. Thus, $\beta \frac{r_{c_i}}{prc_i} + \frac{r_{c_i}}{prc_i} > \beta \frac{r_{c_i}}{prc_i} + r_{c_i}$. We examine the generated cost for each node $d \in D_i$ as follows.

1) Node $d$ is in $T_{i,stable}^0$. Both RT and $SPT(D_{i,rest}^0)$ are sub-trees rooted at $s$ in $SPT(D)$. Thus, $T_{i,stable}^0 \oplus SPT(D_{i,rest}^0)$ is also a sub-tree in $SPT(D)$. Since node $d$ is added according to RT, the potential rerouting costs for $d$ before and after $T_{i,stable}^0 \oplus SPT(D_{i,rest}^0)$ (i.e., $prc_i^0$ and $\frac{r_{c_i}}{prc_i}$) are zero. The rerouting cost for $d$ remains the same after $T_{i,stable}^0 \oplus SPT(D_{i,rest}^0)$ because the routing path of $d$ does not change, leading to a contradiction.

2) Node $d$ is in $D_{i,rest}^0$. After $T_{i,stable}^0 \oplus SPT(D_{i,rest}^0)$, $\frac{r_{c_i}}{prc_i}$ becomes zero.

a) Node $d$ is in $D_{i,stable} \cap D_i$. For $d$, $prc_i$ is no smaller than rerouting cost $r_{c_i}$ since $t_{i-1}$ finds $prc_i$ for $d$ from the symmetric difference between the path from $s$ to $d$ in $T_{i-1}$ and the shortest path from $s$ to $d$ in $G$, leading to a contradiction.

b) Node $d$ is not in $D_{i,rest}$. For $d$, $r_{c_i} = r_{c_i} = 0$ in $T_i$. It leads to a contradiction since $prc_i$ is non-negative. Thus, at least one candidate DT with the sufficient deposit.

Since all candidate DTs from Candidate DT Patching have the sufficient deposit, the one chosen by Final DT Selection also follows and generates a feasible tree.

Lemma 2. $\sum_i B_i$ is no smaller than OBSTA($G$)

Proof. By Lemma 1 the sufficient deposit guarantees $d_{i} = \sum_i B_i - \sum_i (w(T_i) + \alpha \times b_{T_i} + \beta \times r_{c_i}) \geq \beta \times prc_i$. Because $prc_i$ is non-negative in each $T_i$,

$$\sum_i B_i \geq \sum_i (w(T_i) + \alpha \times b_{T_i} + \beta \times r_{c_i})$$

Therefore, $\sum_i B_i \geq OBSTA(G)$.

Theorem 2. OBSTA is a $|D_{max}|$-competitive algorithm.

Proof. Let $\max_{d_i \in D_i}\{\text{dist}_G(s, d_i)\}$ denote the maximum of all shortest paths from $s$ to $D_i$.

$$\sum_i B_i = \sum_i (\alpha \times |D_i| + \sum_{d \in D_i} \text{dist}_G(s, d)) \leq \sum_i |D_i| (\alpha + \max_{d \in D_i}\{\text{dist}_G(s, d_i)\})$$

Then, $\max_{d_i \in D_i}\{\text{dist}_T(s, d_i)\} \geq \max_{d_i \in D_i}\{\text{dist}_G(s, d_i)\}$ holds, where $T_i^*$ is the optimal tree at $t_i$, and

$$\sum_i B_i \leq \sum_i |D_i| (\alpha + \max_{d \in D_i}\{\text{dist}_G(s, d_i)\})$$

$$\leq \sum_i |D_i| (\alpha + \max_{d \in D_i}\{\text{dist}_T(s, d_i)\})$$

$$\leq |D_{max}| \sum_i (\alpha + w(T_i^*))$$

$$\leq |D_{max}| \sum_i (w(T_i^*) + \alpha \times b_{T_i} + \beta \times r_{c_i})$$

since $b_{T_i} \geq 1$. Also, $w(T_i^*) + \alpha \times b_{T_i} + \beta \times r_{c_i}$ is the optimal cost at $t_i$, we have $\sum_i B_i \leq |D_{max}| \times OPT$.

Time complexity. Note that OBSTA takes time $O(|V|^2)$ to update the budget, deposit, budget, rerouting cost, and potential
rerouting cost, and examine whether the deposit is sufficient at each time slot. RT Construction takes time $O(|D|)$ to update the SI values of all nodes and then select the stable nodes (i.e., $SI(d) > H$) among them. Candidate DT Generation takes time $O(|V||D|)$, since it constructs at most $O(|D|)$ candidate trees and each of them needs time $O(|V|)$. For each candidate DT, Candidate DT Patching needs time $O(|V|^2|D|)$, since it iteratively examines whether the deposit is sufficient after adding each of $O(|D|)$ stable nodes, where each addition and examination require time $O(|V|)$ and $O(|V|^2)$, respectively, and it then takes time $O(|V||D|+|D|^2)$, $O(|D|^2)$, $O(|V||D|)$, $O(|V|^2)$, and $O(|V|^2)$ on graph construction, MST construction, graph reversion, SPT construction, and deposit update. Therefore, Candidate DT Patching takes time $O(|D||V|^2|D|^2)$. Finally, Final DT Selection selects the best one among $O(|D|)$ candidate DTs, and thus the total complexity is $O(|V|^2|D|^2)$. Note that the above result is worst-case analysis, while the next section indicates that OBSTA is very efficient for large networks with large groups (see Table I).

VI. PERFORMANCE EVALUATION

We evaluate OBSTA by the simulation on real topologies with real multicast traces and implementation on an experimental SDN with HP SDN switches and YouTube traffic.

A. Simulation Setup

We compare OBSTA with SPT and ST[1] in two real networks, i.e., TataNld with 145 nodes/193 links and UsSignal from the Internet Topology Zoo with 63 nodes/78 links, as well as large synthetic networks with thousands of nodes to test the scalability. In the simulation, a link with the higher delay and higher loss rate (due to congestion) is assigned a larger weight [34]. We set the packet loss rate of each link from 1% to 10% and link delay from 10 ms to 100 ms. Multicast join and leave dynamics are extracted from the trace file of SNDlib over 24 hours and divided into 195 time slots. α and β are set to 0.1 and 0.6, respectively. We also change the duration of the time slots, the weight β for rerouting, and the network size |V|. The performance metrics include the 1) total cost, 2) tree cost, 3) branch cost, and 4) rerouting cost defined in Section II. We implement all algorithms in an HP DL580 server with four Intel Xeon E7-4870 2.4 GHz CPUs and 128 GB RAM. Each simulation result is averaged over 100 samples.

B. Small Real Networks

Fig. 8 compares OBSTA, SPT, and ST, and the tree costs (i.e., bandwidth consumption) of all approaches gradually increase since the joining events are much more than the leaving events. ST generates the smallest tree cost, which is much smaller than SPT. The tree cost of OBSTA is very close to ST, but the total rerouting costs of OBSTA are only 8.8% of ST in TataNld and 9.3% of ST in UsSignal, respectively, whereas SPT maintains shortest paths and thereby does not reroute and thereby is not shown in the figures. A larger rerouting cost incurs a higher volume of control messages and requires more CPU resources at both controller and switch sides [13]. Moreover, the branch cost of OBSTA is much smaller than ST (at least 33% reduction but not shown as a figure due to the space constraint) and thereby is more scalable for SDN. Therefore, OBSTA outperforms SPT and ST because it optimizes not only the tree cost and branch cost but also the rerouting cost at different time.

C. Large Synthetic Networks

Fig. 9 compares OBSTA, ST, and SPT in larger synthetic networks, where |V| ranges from 1000 to 5000. The total cost of OBSTA is 25% less than SPT and ST, and the rerouting cost is only 9.8% of ST since the temporal information is carefully examined in OBSTA. Compared with Fig. 8, the advantage of OBSTA is more significant here, especially for rerouting costs. Minimizing the rerouting cost is more critical in a large SDN because the latency between a controller and switches increases, and high latency may deteriorate user QoE in video services. With a higher β in Fig. 9(d), the rerouting cost can be further reduced for various network sizes. We also change the duration of time slots from 10 to 40 events. Because OBSTA adjusts multicast tree less frequently with longer time slots, the tree costs are slightly increased by 5.5%. Table I summarizes the running time of OBSTA and ST (in brackets) with different |D|. In smaller networks (|V| = 1000 and |V| = 2000), the running time for OBSTA is less than 1 second. Although larger |D| and |T| indeed increase the computation time, OBSTA is still much more efficient than ST. Therefore, the above results manifest that OBSTA is more practical to be deployed in SDN controllers.
D. Implementation

We compare different algorithms in an experimental SDN with 20 HP Procurve 5406zl OpenFlow switches and 40 links, whereas OBSTA is implemented in an OpenDaylight SDN controller. A 191-second YouTube 4K video with 34.3 Mbps is first streaming to a proxy (since YouTube does not support multicast) and then multicast to 15 destinations. Table II manifests that ST induces more rerouting and incurs much larger control overheads (20 times of OBSTA). A larger rerouting cost indeed increases the latency (more than 3 seconds) to install new rules into all involved switches for ST to complete the route setup. It may cause video stalling and deteriorate QoE. In contrast, the rerouting in OBSTA only induces 0.154 seconds. On the other hand, SPT produces larger multicast trees and consumes more network bandwidth. The result manifests that OBSTA is promising to support video services in SDN.

VII. Conclusion

Online multicast traffic engineering supporting IETF dynamic group membership is more practical but has not been studied for SDN. In this paper, we formulate Online Branch-n-ary Steiner Tree (OBST), to jointly minimize the bandwidth consumption, SDN multicast scalability, and rerouting overhead. We prove that OBST is NP-hard and does not have a $D_{max}\left(1-\epsilon\right)$-competitive algorithm for any $\epsilon > 0$, and the proposed $D_{max}\left(1-\epsilon\right)$-competitive algorithm exploits the notion of the budget, the deposit and Reference Tree to achieve the tightest bound. The simulation and implementation on real SDNs with YouTube traffic manifest that OBSTA can effectively reduce the total cost by 25% compared with SPT and ST.

TABLE I

| $|V|$  | $|V|=1K$  | $|V|=2K$  | $|V|=3K$  | $|V|=4K$  | $|V|=5K$  |
|------|----------|----------|----------|----------|----------|
| 50   | 0.060(0.7)| 0.27(2.8)| 0.52(6)  | 0.95(11.1)| 1.62(18.2)|
| 100  | 0.13(1.4) | 0.55(5.9)| 1.05(12.7)| 1.91(23.3)| 3.27(38.2)|
| 200  | 0.26(3)   | 1.09(12.9)| 2.07(27.9)| 3.81(51.2)| 6.5(84)|

TABLE II

| BW Consumption | OBSTA | ST | SPT |
|----------------|-------|----|-----|
| 3591 MB        | 3375 MB | 4563 MB |

REFERENCES

[1] N. McKeown et al., “Openflow: Enabling innovation in campus networks,” SIGCOMM Comput. Commun. Rev., vol. 38, pp. 69–74, 2008.
[2] M. Huang et al., “Dynamic routing for network throughput maximization in software-defined networks,” in IEEE INFOCOM, 2016.
[3] R. Bhatia et al., “Optimized network traffic engineering using segment routing,” in IEEE INFOCOM, 2015.
[4] B. Cain et al., “Internet group management protocol, version 3,” IETF RFC 3376, 2002.
[5] R. Malli, X. Zhang, and C. Qiao, “Benefits of multicasting in all-optical networks,” in Proceedings of the SPIE, vol. 3531, 1998, pp. 209–220.
[6] B. Fenner et al., “Protocol independent multicast - sparse mode (PIM- SM): Protocol specification,” IETF RFC 4601, 2006.
[7] F. K. Hwang and D. S. Richards, “Steiner tree problems,” Networks, vol. 22, pp. 55–89, 1992.
[8] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, “Scalable flow-based networking with DIFANE,” in ACM SIGCOMM, 2010.
[9] L.-H. Huang, H.-C. Hsu, S.-H. Shen, D.-N. Yang, and W.-T. Chen, “Multicast traffic engineering for software-defined networks,” in IEEE INFOCOM, 2016.
[10] D. N. Yang and W. J. Liao, “Optimal state allocation for multicast communications with explicit multicast forwarding,” IEEE Trans. Parallel Distrib. Syst., vol. 19, pp. 476–488, 2008.
[11] D.-N. Yang and W. Liao, “Protocol design for scalable and adaptive multicast for group communications,” in IEEE ICNP, 2008.
[12] I. Stoica, T. S. E. Ng, and H. Zhang, “REUNITE: a recursive unicast approach to multicast,” in IEEE INFOCOM, 2000.
[13] A. R. Curtis et al., “DevoFlow: Scaling flow management for high-performance networks,” in ACM SIGCOMM, 2011.
[14] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità, “An improved LP-based approximation for Steiner tree,” in ACM STOC, 2010.
[15] A. Borodin and R. El-Yaniv, Online Computation and Competitive Analysis. New York, NY, USA: Cambridge University Press, 1998.
[16] S. Jain et al., “B4: Experience with a globally-deployed software defined wan,” in IEEE SIGCOMM, 2013.
[17] S. Agarwal, M. Kodialam, and T. V. Lakshman, “Traffic engineering in software defined networks,” in IEEE INFOCOM, 2013.
[18] X. Jin et al., “Optimizing bulk transfers with software-defined optical WAN,” in IEEE INFOCOM, 2016.
[19] S. Jia et al., “Competitive analysis for online scheduling in software defined optical WAN,” in IEEE INFOCOM, 2017.
[20] S. Gay, R. Hartert, and S. Vissicchio, “Expect the unexpected: Sub second optimization for segment routing,” in IEEE INFOCOM, 2017.
[21] O. Cohen et al., “Simple-tying middlebox policy enforcement using SDN,” in IEEE SIGCOMM, 2013.
[22] R. Cohen et al., “On the effect of forwarding table size on SDN network utilization,” in IEEE INFOCOM, 2014.
[23] D.-N. Yang and W. Liao, “On bandwidth-efficient overlay multicast,” IEEE Trans. Parallel Distrib. Syst., vol. 18, pp. 1503–1515, 2007.
[24] S. Jia et al., “Routing for software-defined networks,” in IEEE INFOCOM, 2015.
[25] R. Zhu, D. Niu, B. Liy, and Z. Li, “Optimal multicast in virtualized datacenter networks with software switches,” in IEEE INFOCOM, 2017.
[26] Awerbuch, Y. Azar, and S. Plotkin, “Throughput-competitive on-line routing,” in IEEE FOCS, 1993.
[27] M. Mohan, T. Truong-Huu, and M. Gurusamy, “TCAM-aware local rerouting for fast and efficient failure recovery in software defined networks,” in IEEE GLOBECOM, 2015.
[28] S. Jia et al., “B4: Experience with a globally-deployed software defined wan,” in IEEE INFOCOM, 2016.
APPENDIX A

PSEUDO-CODE

Procedure 1 Sprouting \( A \otimes (a_1, a_2, ..., a_n) \)

Input: \( A, (a_1, a_2, ..., a_n) \) and \( s \)
Output: \( (A_1, A_2, ..., A_n) \)
1: \( A_0 = \phi \)
2: for all \( 1 \leq i \leq n \) do
3: \( A_i = A_{i-1} \cup P_A(s, a_i) \)
4: end for

Procedure 2 Grafting \( A \oplus B \)

Input: \( A, B, s \) and \( d_1, d_2, ..., d_n \)
Output: \( C \)
1: \( C \leftarrow A \)
2: for all \( 1 \leq i \leq n \) do
3: \( u \leftarrow d_i \)
4: while \( u \notin A \) do
5: \( u \leftarrow u \)'s parent in \( B \).
6: end while
7: \( C \leftarrow C \cup P_B(u, d_i) \)
8: end for

Procedure 3 Reference Tree Construction

Input: \( G = (V, E), s, n, D, D_1 \) to \( D_i \) and \( H \)
Output: \( \tau_i \)
1: for all \( d \in D \) do
2: \( a_d \leftarrow \) the first arrival time slot of \( d \)
3: \( h_d \leftarrow \) the duration in the group of \( d \)
4: if \( h_d \leq \frac{a_d - a_d}{2} \) then
5: \( SI(d) = \frac{2h_d}{n-1} \)
6: else if \( h_d > \frac{n-a_d}{2} \) then
7: \( SI(d) = 1 \)
8: end if
9: end for
10: \( \tau_i = \lfloor d|d \in D \land SI(d) > H \rfloor \)

Procedure 4 Candidate Deployed Tree (DT) Generation

Input: \( G = (V, E), s, D_i, D_{i-1}, D_i, T_{i-1} \)
Output: \( T_i, D^i_{i,join} \)
1: \( D_{i,join} \leftarrow D_i \setminus D_{i-1} \)
2: \( T_{i,leave} \leftarrow \bigcup_{d \in D_i \cap D_{i-1}} P_{T_{i-1}}(s, d) \)
3: \( T_i \leftarrow s \)
4: \( (q_1, q_2, q_3, ..., q_k) \leftarrow \) the sorted sequence of destinations in a non-decreasing order of the between \( s \) and the destination in \( D_i \cap D_{i-1} \) in \( T_{i-1} \).
5: \( T_i^{l_1, ..., l_m} \leftarrow T_{i,leave} \otimes (q_1, q_2, q_3, ..., q_k) \)
6: \( D^i_{i,join} \leftarrow D_{i,join} \cup \bigcup_{j=m+1}^{k} T_j \)

Procedure 5 Candidate Deployed Tree (DT) Patching

Input: \( G = (V, E), s, D_i, D_{i-1}, D_i, T_{i-1}, D^i_{i,stable} \)
Output: \( T_i^l \)
1: \( k \leftarrow |D_{i,stable}^l| \)
2: \( (q_1, q_2, q_3, ..., q_k) \leftarrow \) the sorted sequence of nodes \( D_{i,stable}^l \) in a non-decreasing order of SI.
3: \( (T_{i,stable}^l, ..., T_{i,stable}^k) \leftarrow (RT_i(D_{i,stable}^l) \otimes (q_1, ..., q_k)) \otimes T_i^l \)
4: \( T_{i,stable}^l \leftarrow T_i^l \)
5: for all \( 1 \leq m \leq k \) do
6: if \( T_{i,stable}^m \) has a sufficient deposit then
7: \( T^l_{i,stable} \leftarrow T^l_{i,stable} \)
8: end if
9: end for
10: \( D_{i,rest} \leftarrow D_i \setminus D_{i,stable} \)
11: Build \( T_{i,rest-MST}^l \)
12: if \( T_{i,rest-MST}^l \) has a sufficient deposit then
13: \( T_i^l \leftarrow T_{i,rest-MST}^l \)
14: else
15: Build \( T_{i,rest-SPT}^l \)
16: if \( T_{i,rest-SPT}^l \) has a sufficient deposit then
17: \( T_i^l \leftarrow T_{i,rest-SPT}^l \)
18: else
19: Discards the candidate DT \( T_i^l \)
20: end if
21: end if

Procedure 6 Final Deployed Tree (DT) Selection

Input: \( T_i^l, dep_i^l, prc_i^l \) and \( \gamma \)
Output: \( T_i \)
1: \( g(T_i^l) \leftarrow \gamma \times \text{dep}_i^l + (1 - \gamma) \times \text{prc}_i^l \)
2: \( T_i \leftarrow \arg \max_{\gamma^l} \{g(T_i^l)\} \)