Multi-Tree Multicast Traffic Engineering for Software-Defined Networks

Abstract—Although Software-Defined Networking (SDN) enables flexible network resource allocations for traffic engineering, current literature mostly focuses on unicast communications. Compared to traffic engineering for multiple unicast flows, multicast traffic engineering for multiple trees is very challenging not only because minimizing the bandwidth consumption of a single multicast tree by solving the Steiner tree problem is already NP-Hard, but the Steiner tree problem does not considered the link capacity constraint for multicast flows and node capacity constraint to store the forwarding entries in Group Table of OpenFlow. In this paper, therefore, we first study the hardness results of scalable multicast traffic engineering in SDN. We prove that scalable multicast traffic engineering with only the node capacity constraint is NP-Hard and not approximable within \( \delta \), which is the number of destinations in the largest multicast group. We then prove that scalable multicast traffic engineering with both the node and link capacity constraints is NP-Hard and not approximable within any ratio. To solve the problem, we design a \( \delta \)-approximation algorithm, named Multi-Tree Routing and State Assignment Algorithm (MTRSA), for the first case and extend to the general multicast traffic engineering problem. The simulation results demonstrate that the solutions obtained by the proposed algorithm are more bandwidth-efficient and scalable than the shortest-path trees and Steiner trees. Most importantly, MTRSA is computation-efficient and can be deployed in SDN since it can generate the solution on massive networks in a short time.

Index Terms—SDN, multicast, traffic engineering, reliable transmissions

I. INTRODUCTION

Software-defined networking (SDN) provides a new centralized architecture with flexible network resource management to support a huge amount of data transmission [1]. Different from legacy networks, SDN separates the control plane from switches and allows the control plane to be programmable to efficiently optimize the network resources. OpenFlow [2] in SDN includes two major components: controllers (SDN-Cs) and forwarding elements (SDN-FEs). Controllers are in charge of handling the control plane and install forwarding rules based on different policies, while forwarding elements in switches deliver packets according to the rules specified by the controllers. Compared with the current Internet, routing paths no longer need to be the shortest ones, and the paths can be distributed more flexibly inside the network. It has been demonstrated that SDN provides a better overview of network topologies and enables centralized computation for traffic engineering for multiple unicast flows [3], [4], [5]. However, multicast traffic engineering for multiple multicast trees in SDN has attracted much less attention in previous studies.

Compared to unicast, multicast has been shown in empirical studies to be able to effectively reduce overall bandwidth consumption in backbone networks by around 50% [6]. It employs a multicast tree, instead of disjoint unicast paths, from the source to all destinations of a multicast group, in order to avoid unnecessary traffic duplication. The current Internet multicast standard, i.e., PIM-SM [7], employs a shortest-path tree to connect the source and destinations, and traffic engineering is difficult for PIM-SM since the path from the source to each destination is the shortest one. Shortest-path tree tends to loses many good opportunities to reduce the bandwidth consumption by sharing more common edges among the paths to different destinations. In contrast, to minimize the bandwidth consumption, a Steiner tree (ST) [8] in Graph Theory minimizes the number of edges in a multicast tree. Nevertheless, ST only focuses on the routing of a multicast tree, instead of jointly optimizing the resource allocations of all trees. Therefore, when the network is heavily loaded, a link will not be able to support a large number of STs that choose the link. Most importantly, Group Table of an SDN-FE will be insufficient to store the forwarding entries of the STs due to the limited TCAM size [9], [10].

Compared to the shortest-path routing in unicast, unicast traffic engineering in SDN is more difficult to aggregate multiple flows in Flow Table of an SDN-FE, and the scalability has been regarded as a serious issue in the deployment of SDN for a large network [11], [12]. The scalability problem for multicast communications is even more serious since the number of possible multicast group is \( O(2^n) \), where \( n \) is the number of nodes in a network, and the number of possible unicast connections is \( O(n^2) \). To remedy this issue, a promising way is to exploit the branch forwarding technique [13], [14], [15], [16], [17], which stores the multicast forwarding entries in only the branch nodes, instead of every node, of a multicast tree, where a branch node in a tree is the node with at least three incident edges. The branch forwarding technique can remedy the multicast scalability problem since packets are forwarded in a unicast tunnel from the logic port of a branch node in SDN-FE [18] to another branch node. In other words, all nodes in the path exploit unicast forwarding in the tunnel and are no longer necessary to maintain a forwarding entry for the multicast tree. Furthermore, when a branch node is not multicast capable for a tree (ex. Group Table is full in this paper), local unicast tunneling from a nearby multicast capable node has been proposed in MBONE [19], PIM-SM[20] and other multicast standards [21] to allow multiple unicast tunnels to pass through the branch node. Nevertheless, compared to...
multicast, it is envisaged that local unicast tunneling will incur more bandwidth consumption since duplicated packets will be delivered in a link. Therefore, there is a trade-off between the link capacity and node capacity, because each branch node can act as either a branch state node with the corresponding forwarding entry stored in Group Table or a branch stateless node that exploits the unicast tunneling strategy.

In comparison with the ST problem, scalable multicast traffic engineering, which jointly allocates the network resources for multiple trees, is much more challenging because both the link capacity and node capacity constraints are involved in the problem. The link capacity constraint states that the total rate of all multicast trees on each link will not exceed the corresponding link capacity, while the node capacity constraint ensures that Group Table of each node is sufficiently large to support the multicast trees with the node as a branch state node. Moreover, scalable multicast traffic engineering with branching forwarding and unicast tunneling techniques is able to allocate the network resources more flexibly. When Group Table of a node is full, unicast tunneling moves the resource requirement from the node to its incident links, whereas the rerouting of the tree is also promising by exploiting the resources of the nearby nodes and links. Therefore, it is necessary for scalable multicast traffic engineering to carefully examine both the routing and the allocation of the branch state nodes of all multicast trees. In this paper, we explore the Scalable Multicast Traffic Engineering (SMTE) problem for SDNs. Given the data rate requirement of each multicast tree, SMTE aims to minimize the total bandwidth cost of all trees, by finding a tree connecting the source and destinations of each group and assigning the branch state nodes for each tree, such that both the link capacity and node capacity constraints can be sustained.

Figure. ?? presents an illustrative example. Fig. 1(a) is the original network with the unit bandwidth cost specified beside each link. The bandwidth cost of each link is the total bandwidth consumption of the link multiplied by the unit bandwidth cost. The node capacity of each node as 1. The link capacity of edge $e_{sa}$ is 1, and the link capacities of the other edges are $\infty$ in this example. There are two multicast trees with both flow rates as 1. The source of the first tree is $s_1 = s$, and its destination set is $D_1 = \{d_1, d_2, \ldots, d_7\}$. The source of the second tree is $s_2 = s$, and its destination set is $D_2 = \{d'_1, d'_2, \ldots, d'_7\}$. Fig. 1(b) shows the first shortest-path tree (blue) and the second shortest-path tree (red). The branch nodes and branch state nodes of the first tree are $\{c, u, v\}$ and $\{u\}$, respectively. The branch nodes and branch state nodes of the second tree are $\{c, v\}$ and $\{c, v\}$, respectively. Note that $v$ is not assigned as a branch state node of the first tree, and $u$ thus needs to exploit unicast tunneling to $d_6$ and $d_7$ directly. Therefore, traffic of the first tree is duplicated in edge $e_{u,v}$. On the other hand, if $v$ was assigned as a branch state node for the first tree, traffic duplication in $e_{u,v}$ would be more serious for the second tree since $v$ has three downstream nodes $d'_5$, $d'_6$, $d'_7$. The total bandwidth cost of the two shortest-path trees in Fig. 1(b) is 99.

Afterward, Fig. 1(c) shows the first Steiner tree (blue) and the second Steiner tree (red), and the branch state nodes of the two trees are also $\{c, u, v\}$ and $\{u\}$, respectively. The total bandwidth cost of the trees in Fig. 1(c) is 103. Finally, Fig. 1(d) presents the first tree (blue) and the second tree (red) in SMTE with the same branch state nodes specified above. The total bandwidth cost of the trees in Fig. 1(c) is 79. Therefore, this example manifests that equipped with branching forwarding and unicast tunneling for scalable SDN, shortest-path trees and Steiner trees incur much more bandwidth consumption and is thereby not suitable to support multicast traffic engineering.

SMTE is very challenging. The ST problem is NP-Hard but can be approximated within the ratio 1.55 [7] and is thus in APX of Complexity Theory. In other words, there exists an approximation algorithm for ST that can find a tree with the total cost at most 1.55 times of the optimal solution. In contrast, we first prove that SMTE-N (i.e., SMTE with only the node capacity constraint, while the link capacity constraint is relaxed) is NP-Hard but cannot be approximated within $\delta$, which denotes the number of destinations of the largest multicast group. Afterward, we prove the SMTE (i.e., with both the link and node capacity) cannot be approximated within any ratio. To solve SMTE-N, we propose a $\delta$-approximation algorithm, named Multi-Tree Routing and State Assignment Algorithm (MTRSA), that can be deployed in SDN-C. MTRSA includes two phases: Multi-Tree Routing Phase and State-Node Assignment Phase, to effectively minimize the total bandwidth cost of all trees according to the node capacity constraint. We first focus on the node capacity (i.e., SMTE-N), instead of the link capacity, because the scalability in Group Table is unique in SDN and has not been explored in previous studies of multicast tree routing for other networks. Since no $(\delta^{1-\epsilon})$-approximation algorithm exists in SMTE-N for arbitrarily small $\epsilon > 0$, MTRSA achieves the best approximation ratio. Afterward, we extend MTRSA to support SMTE with the link capacity constraint.
The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 and 4 formulate SMTE with Integer Programming and describe the hardness results. We present the algorithm design of MTRSA in Section 5, and Section 6 shows the simulation and implementation results. Finally, Section 7 concludes this paper.

II. RELATED WORK

The issues of traffic engineering for unicast traffic in SDN have attracted a wide spectrum of attention in the literature. Sushant et al. [?] developed private WAN of Google Inc. with the SDN architecture. Mueller et al. [?] proposed a cross-layer framework and an adaptive network management system with a novel integrated dynamic traffic engineering approach. Qazi et al. [?] designed a new system in SDN to control the middleboxes, and Mckeown et al. [?] studied the performance of OpenFlow in heterogeneous SDN switches. Agarwal et al. [?] presented unicast traffic engineering in an SDN network with only a few SDN-FEs, while the other routers in the network followed a standard routing protocol, such as OSPF. However, the above studies focused on only unicast traffic engineering, and multicast traffic engineering for multiple multicast trees in SDN has attracted much less attention. To support the multicast communications, the current multicast routing standard PIM-SM [?] 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) [?] minimizes the tree cost and the volume of traffic in a network, ST is computationally intensive and is thus not adopted in the current Internet standard. Overlay ST [?], [?], on the other hand, presents an alternative way to construct a bandwidth-efficient multicast tree in the P2P environment. However, the path between any two P2P clients is still a shortest path in Internet, and it is therefore difficult to optimize the routing of the P2P tree. Most importantly, both SPT and ST are designed to find the routing of a tree, instead of jointly optimizing the resource allocation of multiple trees.

Flow table scalability is crucial to support large-scale SDN networks due to the limited TCAM size. Kanizo at al. [?], who first showed that the major bottleneck in SDN is the restricted table sizes, proposed a framework called Patette to decompose a large SDN table and distribute its entries across a network. Leng et al. [?] proposed a flow table reduction scheme (FTRS) to reduce flow table usage with omnipotent controller functions. DIFANE [?] distributed the flow entries in multiple SDN switches as follows. When an ingress switch receives a flow, it redirects the flow to the switch who is in charge of the flow. Thus, more policies can be supported in the switches. Nevertheless, the above studies were not designed for multicast traffic engineering with multiple trees in SDN.

III. PROBLEM FORMULATION

In this paper, we explore the Scalable Multicast Traffic Engineering (SMTE) problem for SDN. Given the data rate requirement of each multicast group, SMTE aims to minimize the total bandwidth consumption of all multicast groups in the network, by finding a tree connecting the source and destinations of each group, and assigning the branch state nodes for each tree, such that the number of multicast forwarding states will not exceed the size of Group Table in each node, and the total multicast flows on each edge will not exceed the link capacity. Note that a branch node can only facilitate unicast tunneling for a multicast group if it is not assigned as a branch state node in the corresponding multicast tree.

More specifically, given a network $G(V, E)$, where $V$ and $E$ denote the set of nodes and directed edges, respectively, let $b_u$ denote the maximal number of branch state nodes that can be maintained by node $u$. Let $N^+_v$ ($N^-_v$) denote the set of out-neighbor (in-neighbor) nodes of $v$ in $G$. Node $u$ is in $N^+_v$ ($N^-_v$) if $e_{c,u}$ ($e_{u,v}$) is a directed edge from $v$ to $u$ (from $u$ to $v$) in $E$, and $c_{u,v}$ is the capacity of $e_{u,v}$, while $k_{u,v}$ is the unit bandwidth cost of $e_{u,v}$. Let $T = (T_1, T_2, \ldots, T_t)$ denote the set of multicast trees, while $s_i$ act as the root of tree $T_i \in T$, i.e., the source with data rate $f_i$, and the destination set $D_i$ contains the set of destinations in $T_i \in T$. In the following, we first formally define SMTE, while the derivations of the bandwidth consumption will be explained later in this section in the proposed Integer Programming formulation.

Definition 1: For network $G(V, E)$ and multicast groups $T$, SMTE is to find the routing of each tree $T_i$ in $T$ spanning $s_i$ and $D_i$ and assign the branch state nodes in $T_i$, such that each node $u$ acts as the branch state nodes of at most $b_u$ trees, and total multicast bandwidth consumption in each edge $e_{u,v}$ is at most $c_{u,v}$.

In the following, we present the Integer Programming (IP) formulation for SMTE. SMTE includes the following binary decision variables to find the routing of each multicast tree and the assignment of branch state nodes. Let binary variable $x_{i,d,u,v}$ denote if edge $e_{u,v}$ is in the path from $s_i$ to a destination node $d$ in $D_i$ in $T_i$. Let integer variable $\epsilon_{i,u,v}$ denote the number of times that each packet of $T_i$ is sent in edge $e_{u,v}$ via multicast (once) or unicast tunneling (multiple times). Let binary variable $\beta_{i,u,v}$ denote if $v$ is a branch state node in $T_i$. Intuitively, when we are able to find the path from $s_i$ to each destination node $d$ of $T_i$ with $x_{i,d,u,v} = 1$ on every edge $e_{u,v}$ in the path, together with the set of branch nodes $\beta_{i,u}$, the routing of the tree (the set of edges $e_{u,v}$ with $\epsilon_{i,u,v} \geq 1$) can be constructed according to the paths from $s_i$ to all destination nodes in $D_i$.

The objective function of the IP formulation for SMTE is as follows.

$$\min \sum_{1 \leq i \leq t} \sum_{e_{u,v} \in E} f_i \times k_{u,v} \times \epsilon_{i,u,v}.$$  

The objective function minimizes the total bandwidth cost of all multicast trees. For each tree $T_i$, the following constraints first describe the routing assignment (i.e., $x_{i,d,u,v}$) for the path connecting the source $s_i$ and each destination in $D_i$. Afterwards, we assign the branch nodes (i.e., $\beta_{i,u}$) in

\footnote{In the following, we first assume that the memory size allocated in Group Table to maintain the branch state node of each multicast tree is the same, and later we extend it to the general scenario that supports different memory sizes for different multicast trees [?] (ex. according to the degree of the node in the trees) in Section V C.}
different nodes and then derive the bandwidth consumption (i.e., $\varepsilon_{i,u,v}$) of $T_i$ via multicast and unicast tunneling.

$$\sum_{v \in N_i^+} \pi_{i,d,s,v} - \sum_{v \in N_i^-} \pi_{i,d,v,s} = 1, \forall 1 \leq i \leq t, d \in D_i,$$

$$\sum_{u \in N_i^+} \pi_{i,d,u,d} - \sum_{u \in N_i^-} \pi_{i,d,d,u} = 1, \forall 1 \leq i \leq t, d \in D_i,$$

$$\sum_{v \in N_i^+} \pi_{i,d,u,v} = \sum_{v \in N_i^-} \pi_{i,d,u,v},$$

$$\forall 1 \leq i \leq t, d \in D_i, u \in V, u \neq d, u \neq s_i,$$

$$\pi_{i,d,u,v} \leq \varepsilon_{i,u,v}, \forall 1 \leq i \leq t, d \in D_i, \forall u \in E,$$

$$-|D_i| \times \beta_{i,u} + \sum_{v \in N_i^+} \varepsilon_{i,u,v} \leq \sum_{v \in N_i^-} \varepsilon_{i,u,v},$$

$$\forall 1 \leq i \leq t, u \in V, u \neq s_i,$$

$$\beta_{i,u} \leq b_u, \forall u \in V,$$

$$\sum_{1 \leq i \leq t} f_i \times \varepsilon_{i,u,v} \leq c_{u,v}, \forall \varepsilon_{u,v} \in E.$$

The first three constraints, i.e., (1), (2), and (3), are the flow-continuity constraints for each tree $T_i$ to find the path from $s_i$ to every destination node $d$ in $D_i$. More specifically, $s_i$ is the source node, and constraint (1) states that the net outgoing flow from $s_i$ is one, implying that at least one edge $e_{i,s_i,v}$ from $s_i$ to any neighbor node $v$ needs to be selected with $\pi_{i,d,s,v} = 1$. Note that here decision variables $\pi_{i,d,s,v}$ and $\pi_{i,d,v,s}$ are two different variables because the flow is directed. On the other hand, every destination node $d$ is the flow destination, and constraint (2) ensures that the net incoming flow to $d$ is one, implying that at least one edge $e_{i,u,d}$ from any neighbor node $u$ to $d$ must be selected with $\pi_{i,d,u,d} = 1$. For every other node $u$, constraint (3) guarantees that $u$ is either located in the path or not. If $u$ is located in the path, both the incoming flow and outgoing flow for $u$ are at least one, indicating that at least one binary variable $\pi_{i,d,u,v}$ is 1 for the incoming flow, and at least one binary variable $\pi_{i,d,v,u}$ is 1 for the outgoing flow. Otherwise, both $\pi_{i,d,u,v}$ and $\pi_{i,d,v,u}$ are 0. Note that the objective function will ensure that $\pi_{i,d,v,u} = 1$ for at most one neighbor node $v$ to achieve the minimum bandwidth consumption. In other words, both the incoming flow and outgoing flow among $u$ and $v$ cannot exceed 1.

Constraints (4) and (5) are formulated to find the routing of the tree and its corresponding branch state nodes, i.e., $\varepsilon_{i,u,v}$ and $\beta_{i,u}$. Constraint (4) states that $\varepsilon_{i,u,v}$ is at least 1 if edge $e_{u,v}$ is included in the path from $s_i$ to at least one $d$, i.e., $\pi_{i,d,u,v} = 1$. The tree $T_i$ is the union of the paths from $s_i$ to all destination nodes in $D_i$. Constraint (5) is the most crucial one. For each node $u$ in $T_i$, if it is not a branch state node, i.e., $\beta_{i,u} = 0$, $u$ does not maintain a forwarding entry of $T_i$ in Group Table and thereby facilitates unicast tunneling. In this case, constraint (5) and the objective function guarantee that the number of packets received from an incoming link $e_{v,u}$ must be the summation of the number of packets sent to every outgoing link $e_{v,u}$. By constraint, when $\beta_{i,u} = 1$, constraint (5) becomes redundant because the Left-Hand-Side (LHS) is smaller than 0 and thereby imposes no restriction on the Right-Hand-Side (RHS). In this case, constraint (4) ensures that $\varepsilon_{i,u,v} = 1$ for every incident edge $e_{v,u}$ with $\pi_{i,d,u,v}$ as 1. Therefore, $u$ is multicast capable for $T_i$, and each packet is delivered at most once in every incident link.

The last two constraints are capacity constraints. Constraint (6) states that each node $u$ can act as a branch state node of at most $b_u$ trees in $T$, while constraint (7) describes that the total multicast bandwidth consumption of in each directed edge $e_{v,u}$ cannot exceed $c_{u,v}$.

### IV. Hardness Results

In the following, we first show that SMTE-N is very challenging in Complexity Theory by proving that it is NP-hard and not able to be approximated within $\delta^c$ for every $c < 1$, where $\delta = \max_{1 \leq i \leq t} |D_i|$. Afterward, we prove that SMTE cannot be approximated within any ratio.

The Steiner tree problem is a special case of SMTE-N. However, SMTE-N is much more challenging than the Steiner tree problem because the Steiner tree problem can be approximated within ratio 1.55 and is thus in APX in complexity theory, but we find out that SMTE is much more difficult to be approximated. The following theorem proves that SMTE-N cannot be approximated within $\delta^c$ for every $c < 1$.

**Theorem 1:** For any $\epsilon > 0$, there exists no $(\delta^1-\epsilon)$-approximation algorithm for SMTE, where $\delta = \max_{1 \leq i \leq t} |D_i|$, assuming $P \neq \text{NP}$.

**Proof:** We prove the theorem with the gap-introducing reduction from the 3SAT problem.

The 3SAT problem is a simplification of the regular SAT problem. An instance of 3SAT is a conjunctive normal form (CNF) in which each clause contains exactly three variables. The 3SAT problem is to determine, given a Boolean expression $\phi$ in CNF such that each clause contains exactly three variables, whether $\phi$ is satisfiable.

Given a directed graph $G$ and destination sets $D_1, D_2$, let $\text{OPT}(G)$ denote the optimal solution of $G$ for SMTE. (We only consider two destination sets, more destination sets are similar.)

For an instance $\phi$ of the 3SAT problem, we build an instance $G(V, E)$ of SMTE such that

1) if $\phi$ is satisfiable then $\text{OPT}(G) \leq 5p^{q+1}$, and
2) if $\phi$ is not satisfiable then $\text{OPT}(G) > (5p^{q+1}) \times \left(\max\{|D_1|, |D_2|\}\right)^{1-\epsilon}$,

where $n$ is the number of Boolean variables in $\phi$, $m$ is the number of clauses in $\phi$, $p = \max\{m, n\}$ and $q$ is a large number determine later.

We first detail how to build the instance of SMTE from an instance of the 3SAT problem. Given an instance $\phi$ of 3SAT with $n$ Boolean variables $x_1, \ldots, x_n$ and $m$ clauses $C_1, \ldots, C_m$, we construct a directed graph $G(V, E)$ such that

1) $\text{OPT}(G) \leq 5p^{q+1}$,
2) the node set $V$ is partitioned into five node sets \{s\}, \{W, U, D_1, D_2\};
3) $W$ contains $n$ nodes $w_1, w_2, \ldots, w_n$, and for each $i$ with $1 \leq i \leq n$, there is a directed edge $(s, w_i)$;
4) $U$ contains $2n$ nodes $u_1, u_2, \ldots, u_n, \overline{u}_n$ (nodes $u_i$ and $\overline{u}_i$ are corresponding to the Boolean variable $x_i$), and for each $i$ with $1 \leq i \leq n$, there are directed edges $(w_i, u_i)$ and $(w_i, \overline{u}_i)$;
5) $D_1$ contains $mp^q$ nodes $d_j^k$, where $1 \leq j \leq m$ and $1 \leq k \leq p^q$ (nodes $d_j^k$ are corresponding to $p^q$ copies of the clause $C_j$), and there exists a directed edge $(u_i, d_j^k)$ if and only if the variable $x_i$ appears in $C_j$;
6) $D_2$ contains $np^q$ nodes $w_i^k$, where $1 \leq i \leq n$ and $1 \leq k \leq p^q$, and there is a directed edge $(w_i, w_j^k)$ for each $i, j$ with $1 \leq i \leq n$ and $1 \leq k \leq p^q$;
7) $G$ only has the directed edges described above;

\[ p = \max(m, n) \] and $q$ is the smallest integer such that $q \geq (3 + \log_2 5) / \epsilon$.

The cost of each edge from $s$ to $W$ is set to be $p^q$, and the cost of the other edges are set to be 1. The capacity of each node is set to be 1.

Let $s$ and $D_1$ be the source node and destination set of $T_1$ respectively, and let $s$ and $D_2$ be the source node and destination set of $T_2$ respectively.

If $\phi$ is satisfiable, there is a truth assignment to $x_i$ such that $x_i$ is assigned to be true. Consider the tree $T_1$ rooted at $s$ which includes 1) the edges between $s$ and $W$, 2) the edges between $W$ and $A$, and 3) the edges between $d_j^k$ and one of its neighbor in $A$ (the existence of its neighbor in $A$ comes from that $\phi$ is satisfiable). Consider the tree $T_2$ which includes 1) the edges between $s$ and $W$, 2) the edges between $W$ and $D_2$. Then $(T_1, T_2)$ is a feasible solution of SMTE and it can act as an upper bound of SMTE in $G$. The basic cost of $T_1$ is $np^q + mp^q$ and the basic cost of $T_2$ is $np^q + mp^q$. Since each node's capacity is enough, the total cost of this feasible solution is $3np^q + n + mp^q \leq 5p^{q+1}$. Hence $\text{OPT}(G) \leq 5p^{q+1}$.

On the other hand, if $\phi$ is not satisfiable, and let $(T_1, T_2)$ be any feasible solution. For $1 \leq k \leq p^q$, let $I_k$ be the set consisting of all $i$ with $1 \leq i \leq n$ such that $u_i$ and $u_i$ are adjacent to some nodes in $d_j^k$ along edges in $T_1$. Since $\phi$ is not satisfiable, we have $I_k$ is nonempty for each $k$ with $1 \leq k \leq p^q$. By pigeonhole principle, there exists at least one $i^*, 1 \leq i^* \leq n$, such that $i^* \in I_k$ for at least $\frac{p_q}{n}$ of $I_k$'s. Thus $(w_l^k, u_i)$ and $(w_l^k, u_i)$ are edges in $T_1$ (hence $w_l^k$ is a branch node in $T_1$), and $w_i$ has at least $\frac{p^q}{n} \geq p^{q-1}$ descendants which are destination nodes of $T_1$. Thus the capacity of node $w_i$ is not enough, and the total cost is at least $p^{2q-1}$. So the total cost of the optimal solution is greater than $p^{2q-1}$. Hence, $\text{OPT}(G) > p^{2q-1} = (5p^{q+1})(p^{q-2} + \log_2 5) = (5p^{q+1})(p^{q-1} + 1 + \epsilon) = (5p^{q+1})(5p^{q+1} + 1 - \epsilon) > (5p^{q+1})(5p^{q+1})^{1-\epsilon} \geq (5p^{q+1})(5p^{q+1})^{1-\epsilon}$. Since $\epsilon$ can be arbitrarily small, there is no $(\max\{D_i, D_j\})^{1-\epsilon}$ approximation algorithm for SMTE, assuming $P \neq NP$.

Theorem 2: For any polynomial time computable function $f$, SMTE cannot be approximated within a factor of $f(|V|)$, unless $P = NP$. In other words, for arbitrary positive integer $k$, SMTE cannot be approximated within $|V|^k$.

Proof: Assume, for a contradiction, that there is a factor $f(|V|)$ polynomial time approximation algorithm, $A$, for

Our-problem. We will show that $A$ can be used for deciding the 3SAT problem in polynomial time, thus implying $P = NP$. The 3SAT problem is a simplification of the regular SAT problem. An instance of 3SAT is a Boolean expression in which each clause contains exactly three variables. The problem, given a Boolean expression in CNF such each clause contains exactly three variables, determines whether it is satisfiable. Given a graph $G$, let $\text{OPT}(G)$ denote the optimal solution of $G$ for Our-problem.

For an instance of the 3SAT problem, we build an instance of Our-problem on $G(V, E)$, such that
- if $\phi$ is satisfiable then $\text{OPT}(G) \leq m + 3n$, and
- if $\phi$ is not satisfiable then $\text{OPT}(G) > (m + 3n) \times f(|V|)$, where $n$ is the number of Boolean variables, $m$ is the number of clauses, and $f(t)$ is a polynomial time computable function. Hence there is no $f(|V|)$ approximation algorithm for Our-problem unless $P = NP$.

We first detail how to build the instance of Our-problem from an instance of the 3SAT problem. Given an instance $\phi$ of 3SAT with $n$ Boolean variables $x_1, \ldots, x_n$ and $m$ clauses $C_1, \ldots, C_m$, we construct a directed graph $G(V, E)$ such that
- the node set $V$ is partitioned into four node sets $s, U, D_1$, and $D_2$;
- $U$ contains $2n$ nodes $u_1, u_2, \ldots, u_n$, $u_{n+1}, \ldots, u_{2n}$ (nodes $u_i$ and $u_i$ correspond to the Boolean variable $x_i$), and for each $i$ with $1 \leq i \leq n$, there are directed edges $(s, u_i)$ and $(s, u_i)$;
- $D_1$ contains $m$ nodes $d_1, \ldots, d_m$ (nodes $d_j$ are corresponding to the clause $C_j$), and there exists a directed edge $(u_i, d_j)$ if and only if the variable $x_i$ is assigned to be false. Consider the tree $T_1$ rooted at $s$ which includes 1) the edges between $s$ and $W$ and 2) the edges between $d_j$ and one of its neighbor in $W$. The existence
of its neighbor in $A$ comes from that is satisfiable). Consider the tree $T_2$ which includes 1) the edges between $U \setminus W$ and 2) the edges between $U \setminus W$ and $D_2$. Then $(T_1, T_2)$ is a feasible solution of Our-problem and it can act as an upper bound of Our-problem in $G$. The basic-cost of $T_1$ is $m + n$ and the basic-cost of $T_2$ is $2n$. The total-cost of this feasible solution is $m + 3n$. Hence $OPT(G) \leq m + 3n$.

On the other hand, if $\phi$ is not satisfiable, and let $(T_1, T_2)$ be any feasible solution. Since $\phi$ is not satisfiable, there is an $i$ such that $u_i$ and $w_i$ belong to $T_1$. Since for each directed edge with capacity 1, the directed edge $(s, d_i')$ must belong to $T_2$. Thus the total-cost of this feasible solution is greater than $(m + 3n) \times f(|V|)$. Therefore Our-problem can not be approximated within a factor of $f(|V|)$, unless $P = NP$. ■

V. ALGORITHM DESIGN

In the follows, we first propose a $\delta$-approximation algorithm, named Multi-Tree Routing and State Assignment Algorithm (MTRSA), for SMTE-N, where $\delta = \max_{1 \leq i \leq |D|} |D_i|$. Note that we first focus on the node capacity, instead of the link capacity, because the the scalability in Group Table Assignment is unique in SDN and has not been explored in previous studies of multicast tree routing for other networks. Since Theorem 1 proves that there is no $(D^{1-\epsilon})$-approximation algorithm of SMTE-N for any $\epsilon > 0$, MTRSA achieves the best approximation ratio. Afterward, we extended it to support SMTE. Due to the space constraint, the pseudo-code is presented in [CORR].

A. Algorithm Description

MTRSA includes two phases: 1) Multi-Tree Routing Phase and 2) State-Node Assignment Phase. Multi-Tree Routing Phase first constructs an initial multicast tree for each multicast group to minimize the total bandwidth consumption and balance the distribution of branch nodes in different trees. Afterward, State-Node Assignment Phase finds the branch state nodes for each multicast tree to follow the node constraint.

1) Multi-Tree Routing Phase: Multi-Tree Routing Phase first constructs a shortest-path tree with source $s_i$ and destination set $D_i$ for each tree $T_i \in T$. A node $u$ is full if it acts as a branch node for $b_u$ trees. By contrast, $u$ is overloaded if it acts as a branch node for more than $b_u$ multicast trees, and $u$ needs to act as a branch stateless node for some trees and thereby will incur more bandwidth consumption. To address this issue, after finding the shortest-path trees, if there is an overloaded node, we adjust the local tree routing nearby the overloaded node to move the branch node to another node that has not been full, in order to balance the distribution of branch nodes among different multicast trees.

More specifically, if any node $u$ is an overloaded node and a branch node in any tree $T_i$, MTRSA chooses a node $v$ of $T_i$ such that: 1) $v$ is a downstream of $u$ in $T_i$, 2) $v$ is a branch node or a destination node of $T_i$, and 3) there is no other branch node or destination node in the path from $u$ to $v$ in $T_i$, MTRSA reroutes the path from $u$ to $v$ in $T_i$ as follows. Let $\ell$ denote the total cost of the path from $u$ to $v$ in $T_i$. We find a new path from $v$ to a node in $T_i$ such that: 1) the total cost of the new path is at most $\ell$, 2) the new path does not pass through any exiting node in $T_i$, and 3) this new path ends at a new node $v'$ which is not a leaf node of $T_i$ and not full or overloaded. We update tree $T_i$ by substituting the old path from $u$ to $v$ in $T_i$ with the new path from $v$ to $v'$, and the overload situation in $u$ can be alleviated accordingly. Afterward, we process $u$ for another tree $T_i$ iteratively until $u$ is no longer overloaded.

Example. Consider the following example in Fig. ??.

Let $G(V, E)$ be the network with two multicast trees $T_1$ and $T_2$. The number on each edge is the cost of this edge, and the node capacity of every node is 1. The source $s_1$ of the first tree $T_1$ is $s$ with the corresponding destination set $D_1 = \{d_1, d_2, \ldots, d_8\}$, while the source $s_2$ of $T_2$ is also $s$, but the destination set is $D_2 = \{d_1', d_2', \ldots, d_6'\}$. In Multi-Tree Routing Phase, we first find the blue and red shortest-path trees $T_1$ and $T_2$ in Fig. ???. Afterward, we adjust the multicast trees for overloaded nodes. Specifically, the node capacity of $a$ is 1, but $a$ is a branch node of both $T_1$ and $T_2$. Therefore, $a$ is an overloaded node, and MTRSA examines nodes $d_1, d_2, v, c$, which are downstream nodes of $a$ in $T_1$, and the path from $a$ to each of the above nodes does not pass through any branch or destination nodes. MTRSA thereby adjusts the path $a, b, c$ in tree $T_1$. Since node $y$ is overloaded, node $c$ cannot be rerouted to node $y$. Since node $v$ is not a leaf node in tree $T_1$ and neither full nor overloaded in $T_1$ and $T_2$, MTRSA reroutes the node $c$ to node $v$ as shown in Fig. ???. In Fig. ?? MTRSA also tries to adjust the path $a, b, c$ in tree $T_2$, but node $y$ and $a$ are both overloaded. Therefore, node $c$ is not able be adjusted in $T_2$, and Fig. ?? presents the two multicast trees after Multi-Tree Routing Phase.

2) State-Node Assignment Phase: It is worth noting when the network is heavy-loaded, the first phase may not be able to ensure that every overloaded node can be successfully adjusted to balance the distribution of branch nodes in different trees, and State-Node Assignment Phase is crucial in this case to minimize the increment of bandwidth consumption due to multi-unicast from branch stateless node. More specifically, State-Node Assignment Phase include two stages: 1) Greedy Assigning Stage, and 2) Local Search Stage. Greedy Assigning Stage assigns the branch state nodes by iteratively maximize the reduction of the number of branch state nodes, and later in Section V-B, we prove that the the number of branch state nodes reduced by the Greedy Assigning Stage is at least half of the number of branch state nodes reduced by an optimal strategy. Local Search Stage then improves the solution by alleviating the assignment on overloaded nodes. We detail the two stages as follows.

For each multicast tree $T_i$ obtained in Multi-Tree Routing Phase, let $W_i$ denote the set of branch nodes in $T_i$, where each node in $W_i$ has not been full, and $W = \cup_{1 \leq i \leq 5} W_i$. By contrast, let $A_i$ be the set of branch state nodes in $T_i$ to be decided in this phase, and $A_i$ thereby is a subset of $W_i$. Let $c(T_i, A_i)$ denote the total bandwidth cost of $T_i$ with the set of branch state nodes as $A_i$. More precisely, $c(T_i, A_i) = \sum_{v \in A_i \cup D_i} \phi(P_v)$, where $P_v$ is the path from the closest upstream node in $A_i$ or the source to $v$, such that all internal nodes of $P_v$ are not in $A_i$, and $\phi(P_v)$ is the cost of all edges in $P_v$. In other words, if there is no branch stateless node in $P_v$, every packet is delivered only
once on every link of $P_i$. By contrast, if $P_i$ include a branch stateless node $u$, each packet is sent multiple times on the link from the parent node of $u$ to $u$, corresponding to the unicast tunneling case.

An assignment $A$ of branch state nodes can be defined as follows: $A$ is a 0,1-matrix with the rows indexed by $\{1, \ldots, t\}$ and columns indexed by $W$, such that 1) the 1’s in row $i$ are all with the column index in $W_i$, and 2) the number of 1’s in column $w \in W$ is no more than the capacity of $w$. We assign a branch state node $w \in W$ to tree $T_i$ if and only if the $(i, w)$ entry of $A$ is 1. In other words, the first condition ensures that a branch state node can only be assigned to a branch node of $T_i$, while the second condition is the node capacity constraint. Given an assignment $A$ of branch state nodes, let $A_i = \{w \in W : \text{the (i, w) entry of } A \text{ is } 1\}$ denote the set of branch state node for $T_i$, and the total bandwidth cost for the set $T$ of all multicast trees with the state-node assignment $A$ is $c(T, A) = \sum_{1 \leq i \leq t} c(T_i, A_i)$. Since a branch state node assignment $A$ can also be regarded as a subset of $N$, where $N = \{1, \ldots, t\} \times W$, let $M$ be the family of all feasible assignments of branch state nodes to $T$, and we use $c(T, \emptyset)$ to denote the total bandwidth cost of $T$ without assigning any branch state node. Now let the set function $z : M \rightarrow \mathbb{R}$ such that $z(A)$ represents the cost reduced by assignment $A$. More formally, $z(A) = c(T, \emptyset) - c(T, A)$ for each $A \in M$.

The Greedy Assignment Stage starts from a branch state node assignment $\emptyset$ and cost $c(T, \emptyset)$, and iteratively assign one branch state node of a node to a tree in $T$ until no more assigning is possible. More precisely, in each iteration, if the present branch state node assignment is $A \in M$, then we choose an element $x$ in $N - A$ such that: 1) $A \cup \{x\}$ is in $M$, and 2) $z(A \cup \{x\}) = \max_{y \in (N - A)} z(A \cup \{y\})$ (the first condition guarantees that the new assignment is feasible, and the second condition is the greedy policy).

Afterward, Local Search Stage adjusts the assignment of branch state nodes for overloaded nodes iteratively. In each iteration, we first extract an overloaded node $u$ and then compute the total bandwidth costs of $T$ with every feasible state-node assignment on $u$ over different trees in $T$, assuming that the state-node assignment of other nodes is not changed. Afterward, this phase chooses the assignment of $u$ with the minimum total bandwidth cost. This stage is repeated until all overloaded nodes are carefully examined. Finally, this phase reroutes the paths from the branch stateless nodes of a tree in order to find a smaller tree with the same assignment of branch state nodes. More specifically, for any branch stateless node $u$ in tree $T_i$, we choose nodes $v$ and $w$ of $T_i$ in the same way as the Multi-Tree Routing Phase in order to find a new path from $w$ to $v$, and $w$ here cannot be an overloaded node.

**Example.** In Greedy Assignment Stage of the State-Node Assignment Phase, MTRSA first assigns a branch state node on $v$ to tree $T_1$ with the cost reduced by 84 (the maximum). It then assigns a branch state node on $y$ to $T_2$ with cost reduced by 45. Afterward, node $c$ is assigned as a branch state node for $T_2$ with the cost reduced by 32, and node $a$ is assigned as a branch state node for $T_2$ with cost reduced by 27. In Local Search Stage, there are three overloaded nodes $a$, $c$, and $y$. For overloaded node $a$, this phase removes the branch state node on $a$ without changing the branch state nodes of the other nodes. If we assign a branch state node on $a$ to $T_1$, it becomes possible to reduce the cost by 27. In contrast, if we assign a branch state node on $a$ to $T_2$, we are able reduce the cost by 18. Therefore, MTRSA changes to assign the branch state node on $a$ to tree $T_1$. Nodes $c$ and $y$ are then processed similarly. Finally, in Fig. ??, since node $a$ is overloaded and node $y$ is a branch state node, node $c$ can be re-routed to node $y$ in $T_2$ to reduce the total bandwidth consumption from ?????? to ?????.

**B. Approximation Ratio and Time Complexity**

In the following, we examine the quality of assignment for branch state nodes in the second phase. We prove that $(N, M)$ is a matroid and the set function $z : M \rightarrow \mathbb{R}$ is a nondecreasing submodular set function. Therefore, according to the theorem on matroid for maximizing submodular set function [ADD CITATION], we have the following theorem.

**Theorem 3:** The number of branch state nodes reduced by the Greedy Assignment Stage is at least one half of the branch state nodes reduced by the optimal assignment of branch state nodes.

**Proof:** First we prove that $M$ is a matroid. $M$ is the family of subsets of $N = \{1, \ldots, t\} \times W$ (we put elements of $N$ in a $t \times |W|$ array) such that the elements in the $i$-th row are in the columns indexed by $W_i$ and the number of elements in the column indexed by $w$ is at most the capacity of $w$. Hence, by definition, we have: 1) $\emptyset \in M$, 2) If $A \subseteq B \in M$, then $A \in M$, and 3) If $A, B \in M$ with $|A| < |B|$, then there is an element $b \in B$ such that $A \cup \{b\} \in M$. Therefore $M$ is a matroid.

Now we prove that the set function $z : M \subseteq 2^N \rightarrow \mathbb{R}$ is submodular and nondecreasing. Let $A, B \in M$ with $A \subseteq B$ and $c \in N$ be the element in row $i$ and column $w$ such that $A \cup \{c\}, B \cup \{c\} \in M$, since $z(A \cup \{c\}) - z(A)$ is the cost reduced by assigning a branch state node in node $w$ to tree $T_i$ with branch state node assignment $A$ and $B$ is the cost reduced by assigning a branch state node in node $w$ to tree $T_i$ with branch state node assignment $B$. Therefore, we have $z(A \cup \{c\}) - z(A) \geq z(B \cup \{c\}) - z(B)$, hence $z$ is submodular. Let $A, B \in M$ with $A \subseteq B$, by definition of $z$, we have $z(A) \leq z(B)$, hence $z$ is nondecreasing.

Let $Z_{opt}$ be $\max\{z(A) : A \in M\}$ and $Z_G$ be the result from greedy algorithm. By a result on maximizing submodular set function on matroid [M.L. Fisher, G.L. Nemhauser, and L.A. Wolsey, An Analysis of Approximations for Maximizing Set Functions-II, Mathematical Programming Study 8 (1978) 73-87], we have $\frac{Z_{opt} - Z_G}{Z_{opt}} \leq \frac{1}{e}$, hence $Z_G \geq \frac{1}{e}Z_{opt}$.

The theorem follows.

In the following, we prove that MTRSA is a $\delta$-approximation algorithm for Our-problem, where $\delta$ is the maximum size of the destination sets. Since Theorem 4 proves that there is no approximation algorithm with ratio $\delta^{1-\epsilon}$ for any $\epsilon > 0$, the following theorem shows that MTRSA achieves the best approximation ratio.

**Theorem 4:** MTRSA is a $\delta$-approximation algorithm for SMTE-N, where $\delta = \max_{1 \leq i \leq t} |D_i|$.
Proof: Let the tree structure $T^* = (T_1^*, \ldots, T_t^*)$ with the branch state node assignment $A^*$ be the optimal solution to Our-problem, $W^* = \bigcup_{i=1}^t W_i^*$, where $W_i^*$ is the set of branch nodes of $T_i^*$, and $A_i^* = \{w \in W_i^* : (i, w) \text{ entry of } A \text{ is } 1\}$ be the node set where $T_i^*$ has branch state nodes, hence the optimal cost is $c(T^*, A^*) = \sum_{i=1}^t c(T_i^*, A_i^*)$. Suppose that in the Routing Phase, we first construct the tree structure by shortest-path tree to get the tree structure $T(1) = (T_1^{(1)}, \ldots, T_t^{(1)})$, and MTRSA finally get tree structure $T(2)$ and branch state node assignment $A(2)$. Because of the constraints in the Multi-Tree Routing Phase, MTRSA updates the tree structure only when the cost does not increase, and the branch state nodes can only decrease the cost, so we have $c(T^{(2)}, A^{(2)}) \leq c(T^{(2)}, \emptyset) \leq c(T^{(1)}, \emptyset)$. Since the path $P_{s_i, d}$ from the source $s_i$ to node $d$ in $T_i$ is the shortest path from $s_i$ to $d$ in $G$, and $T_i^*$ has a path from $s_i$ to $d$, we have $c(P_i^{(1)}) \leq c(T_i^*)$ for each $i$ and each $d$. Therefore $c(T^{(2)}, A^{(2)}) \leq c(T^{(1)}, \emptyset) = \sum_{i=1}^t c(T_i^{(1)}, \emptyset) \leq \sum_{i=1}^t \sum_{d \in D_i} c(P_{s_i, d}) \leq \sum_{i=1}^t \sum_{d \in D_i} c(T_i^*, A_i^*) = \sum_{i=1}^t |D_i| \times c(T_i^*, A_i^*) \leq D(\sum_{i=1}^t c(T_i^*, A_i^*)) = D \times c(T^*, A^*)$. The theorem follows.

Time Complexity. We first find the shortest path between any two nodes in $G$ with Johnson’s algorithm in $O(|V||E| + |V|^2 \log |V|)$ time as pre-processing procedure for quickly look up afterwards. The advantage is that this preprocessing only needs to be performed once but can be exploited during the construction of the shortest-path trees and the reroutings of the tree structure in the Multi-Tree Routing Phase.

In Multi-Tree Routing Phase, first we construct the shortest-path tree for each source $s_i$ and its corresponding destination set $D_i$, MTRSA compares the distance from a destination node to all other nodes in $O(|V|)$ time, and processing all $d \in D_i$ requires $O(|V||D_i|) = O(|D|)$ time, hence processing these $t$ shortest-path trees requires $O(t|D|)$ time. After constructing the shortest-path trees, MTRSA reroutes the routes from each overloaded node to some of its descents in one tree of the tree structure. Since there are at most $t\delta$ branches in the tree structure, and we reroute at most $\delta$ descents of each branches, and each rerouting requires comparison of at most $|V|$ distances of paths, hence it takes $t\delta V \log |V|$ time, then the Routing Phase requires $O(t\delta^2 |V| \log |V|)$ time.

In State-Node Assignment Phase, first we run the Greedy Assigning Stage. In the stage, there are at most $t|V|$ branch state nodes needed to be assigned, so there are at most $t|V|$ iteration. In each iteration, we find the minimum cost reduced by assigning one branch state node of node $v$ to one tree $T_i$, this requires $O(t|V|)$ time, and we update the cost reduced by assigning each non-assigned branch state node to the tree $T_i$ according to the new branch state node assignment, this takes $O(|T_i||E|) = O(|V||E|)$ time, therefore this stage requires $O(t|V|^2(t + |E|)) = O(t|V|^2(t + |E|))$ time. In the Local Search Stage, we consider overloaded nodes in each iteration, hence there are at most $|V|$ iterations. In each iteration, we remove the branch state node assignment of the overloaded node we are considering without changing other branch state node assignment, and calculate the cost reduced by assigning a branch state node to a tree, this requires $O(|E|)$ time, and this stage takes $O(t|V||E|)$ time. Therefore the State-Node Assignment Phase requires $O(t|V|^2(t + |E|))$ time to allocate the branch state nodes and $O(t\delta^2 |V| \log |V|)$ time for rerouting. Hence MTRSA requires $O(t\delta^2 |V|^2 \log |V||(t + |E|)))$ time.

C. Extension to SMTE

For SMTE, we first present the concept of weak edge constraints. Let $\varepsilon_{t,i,u,v} = 1$ if $\varepsilon_{t,i,u,v} > 0$, and 0 otherwise. The link capacity constraints (7) of the linear programming is changed to $\sum_{t=1}^{\infty} f_t \times \varepsilon_{t,i,u,v} \leq c_{i,u,v}, \forall c_{i,u,v} \in E$. That is, we consider the set of trees passing through the edge, and the sum of data rates of those trees is at most the capacity of this edge (without multiplicity of any one tree).

To extend MTRST for SMTE, it is necessary to adjust MTRST as follows.

1) Before Multi-Tree Routing Phase, we sort the multicast trees in $T$ according to their data rate from large to small, so $f_1 \geq f_2 \geq \cdots \geq f_t$.

2) In Multi-Tree Routing Phase, we find the shortest-path tree for the first $T_1 \in T$, then we decrease the capacity of the edges in $T_1$ by its flow rate $f_1$ to update the network. Then we do the same procedure for the new network for the rest multicast trees in $T$ until all multicast trees in $T$ are handled.

3) In Rerouting Stage of Multi-Tree Routing Phase, we reroute according to the weak edge constraints.

4) In Greedy Assignment Stage, we change to find an element $x = (i, u)$ in $N - A$ according to $z(A \cup \{(i, u)\}) - z(A) = \max \{z(A \cup \{(i', u')\}) - z(A) \colon A \cup \{(i', u')\} \in M\}$.

5) In Local Search Stage, optimizing the branch node assignment of one node is the same as the knapsack problem, and we use dynamic programming to get optimization in pseudo-polynomial time.

6) In the Rerouting Stage of State-Node Assignment Phase, rerouting of routes has to satisfy the constraints of link capacities.

VI. PERFORMANCE EVALUATION

We evaluate MTRSA in real networks in this section by simulation. In addition, we also deploy our algorithm in a small experimental SDN network with HP switches to evaluate the video performance with DASH traffic.

A. Simulation Setup

We simulate our algorithm in two real networks, which are VlWavenet2011 and Columbus [2]. VlWavenet2011 includes 91 nodes and 96 links, while Columbus has 70 nodes and 85 links. Our simulation is divided into small and large scales. The numbers of trees in the small scale ones are fewer than 100, and there are more than 2000 trees in the large scale cases. Link capacity in the topologies is set to the level that the maximal bottleneck link utilization reaches 100% [2]. We vary the number of multicast tree, the number of destinations, and node capacity. The source and destinations are chosen randomly from each network.
We compare MTRSA with the following algorithms: 1) the shortest-path tree algorithm (SPT), 2) the Steiner tree (ST) algorithm \cite{1}, and 3) CPLEX \cite{2}, which finds the optimal solution of RST problem by solving the MILP formulation in Section ???. In SPT and ST, each node is randomly assigned to the trees who share the node as a branch node. We change the number of trees, the number of destinations \(k\), and the node capacity. The performance metrics we use is total cost (the summation of all tree costs), which means the amount of link capacity is used by all multicast trees. A lower cost implies that the link resources can support more traffic other than the multicast traffic, and resource utilization can be improved.

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 Scale Evaluation

In the small scale cases, we compare the total tree costs of MTRSA, PT, ST, and the optimal solution generated by CPLEX with different number of trees \(|T|\), different node capacity \(b_u\), and different number of destinations \(|D|\). Each node in the networks can be assign to any tree going through it as a branch node before using out its capacity. Since the scalable multicast traffic engineering is NP-Hard, CPLEX is able to find the optimal solutions for only smaller instances, and we thus only find an optimal solution for both VtlWavenet2011 and VtlWavenet2008. As shown in Fig. 3 and Fig. 4, MTRSA generates a solution with the costs that are very close to the optimal solution. Although SPT provides the shortest path to destinations, it doesn’t reroute its paths to use node capacity efficiently, so its cost is higher than MTRSA. Compared with SPT, the distance between the source and a destination in ST is usually higher since the path needs to be deviated from the shortest one in order to aggregate with another path, so the cost of ST is higher than SPT. With the growing number of trees, the cost is also higher, because more traffic is in the networks as Fig. 5. In Fig. 4 the cost drop a little when node capacity is increased; when the number of destination increases, the cost is also increase because of larger trees.

### C. Large Scale Evaluation

We also evaluate MTRSA, ST, and SPT in larger scale, where the number of multicast tree is from 2000 to 10000, the number of destinations is from 5 to 25, and node capacity is from 50 to 250. Compared with smaller scale cases, the advantage of MTRSA is more significant in larger scale cases. As shown in Fig. 6 the total cost increases with the number of trees. For a larger network, the source and any destination are inclined to be located with more hops away, but there is also a higher chance to find a node with enough capacity as a branch node to reduce cost. In average, MTRSA limits the total cost by 66% and 59%, respectively, compared to ST and SPT. Fig. 7 shows that cost drops when we increase node capacity, and MTRSA reaches its lowest cost when the node capacity is over 100. The cost grows with the number of destinations, because trees spans more nodes as Fig. 6. We also observe resource utilization, which is used node capacity divided by total node capacity, increases slowly with the number of destinations, because larger trees consumes more node capacity.

Table II summarizes the running time of MTRSA with different \(|T|\) and \(|D|\). With a smaller input, such as 2000 trees and 5 destinations, the running time for RAREA is around 1 second. As \(|T|\) and \(|D|\) increase, the running time grows, but RAERA only spends around 73 seconds in the largest case. Note that according to our online algorithm \cite{2}, it is not necessary to re-compute the whole tree when a tree or a destination joins or leaves the network after the trees are initialized with MTRSA. It is envisaged that our algorithm...
is practical to be deployed in SDN networks.

D. Implementation

To evaluate RAERA in real environments, we implement it in our experimental SDN network including HP Procurve 5406zl switches, which are Openflow-enabled switches. We use Floodlight as our Openflow controller to install the traffic routing in SDN-FEs. MTRSA is running on the top of Floodlight. Since YouTube has not supported multicast, a YouTube proxy for multicast is implemented.

Our testbed includes 12 nodes and 24 links as shown in Fig. 10 and we randomly select 10 nodes as video servers as tree roots. For each servers, we randomly assign 10 destinations. We configure each HP switches as multiple SDN-FEs by assigning each slice of the switch as a logical node of the SDN network. The servers send videos along multicast trees, and the test video is in 460s long. Current videos are encode as Variable bitrate (VBR) that cause various bit rate during video playback. We want to observe the total bandwidth consumption of multicast traffic in a real case. Fig. 11 shows the total bandwidth consumption during playback, and we average the bandwidth every 40s. In average, MTRSA uses the bandwidth 46% and 35% lower than ST and SPT, respectively. In addition, the sources in ST traverse longer path to destinations, so ST consumes more bandwidth than SPT. Therefore, MTRSA can reduce bandwidth consumption to use network resources more efficiently.

VII. CONCLUSION
(a) VllW wenet2011
(b) Columbus

Resource Utilization (%)
#Trees (k)
