A Scalable and Bandwidth-Efficient Multicast Algorithm based on Segment Routing in Software-Defined Networking

Jang-Ping Sheu and Yin-Chen Chen
Department of Computer Science, National Tsing Hua University
Hsinchu, 30013, Taiwan
sheujp@cs.nthu.edu.tw and s103062582@m103.nthu.edu.tw

***Abstract*—Software-Defined Networking (SDN) is an emerging architecture and offers advantages over traditional network architecture, while there exist some scalability challenges. In this paper, we propose a multicast routing algorithm for SDN with segment routing to serve the bandwidth requirement of a multicast routing request. Our algorithm considers the balance of traffic load for network resource of link bandwidth and node flow entries both. Simulation results show that the performance of our algorithm is better than previous works in terms of average network throughput and average rejection rate of routing requests. Besides, the results also show that our multicast architecture improve scalability problem of original SDN model in terms of number of flow entries used.**

Keywords- Multicast routing; software-defined networking; segment routing; traffic engineering

I. INTRODUCTION

In Software-Defined Networking (SDN), network intelligence is logically centralized in software-based SDN control plane that act as the "brain" of the network. Compared with the traditional network routing, SDN architecture enables centralized control plane to maintain a global view of the network and dynamic computation on routing for traffic engineering [1]. That is, with global view of network information, SDN controller can dynamically compute paths for routing requests to optimize network performance and utilize network resource efficiently. As a result, SDN provides a novel centralized architecture with more flexible network resource management for traffic engineering than traditional legacy networks. In research area of traffic engineering, unicast routing has been studied widely, while multicast routing in SDN attracted much less attention. Therefore, we focus our research on multicast routing, which is getting increasingly popular for numerous web-based services.

In traditional networks, multicast routing suffer from numerous limitations because creating and maintaining multicast trees require large number of message exchanges between routers for updating multicast trees[2]. However, SDN based architectures have the opportunity to improve these limitations since controller has the capacity to dominate network resource management and traffic control. SDN transforms distributed routing into centralized mechanism with more flexible traffic management. SDN not only provides flexible and efficient traffic management but also supports fine-grained traffic control since it can decide forwarding behaviors based on combinations of packet header fields, in contrast to traditional coarse-grained destination-based forwarding. However, fine-grained traffic control implies that SDN switch requires larger sized flow table than traditional switch. Flow table is implemented by Ternary Content Addressable Memory (TCAM). TCAM is an extremely expensive and power hungry resources that make SDN face challenges such as network scalability problem [3, 4].

To solve scalability problem for traffic engineering, Segment Routing (SR) is a promising way. It is currently in Internet Engineering Task Force (IETF) draft [5] and is driven by Cisco and supported by many leading telecom companies and opens up a promising alternative network operating model. SR is a network technology that offers new concept to do packet forwarding while minimizing the need for keeping a great amount of network states information, which attributes to the TCAM deficiency problem. As for scalability and flexibility, SR avoids millions of label information to be stored in each network device along the path by encoding routing information into packet header as an ordered list of labels to reduce the amount of forwarding rules in TCAM.

In this paper, we study the multicast traffic engineering issue in SDN with SR. We design a novel multicast routing architecture based on SR technique for solving network scalability challenge. In addition, we propose a heuristic multicast routing algorithm with bandwidth guaranty. The proposed algorithm not only achieve the goal of balance the network traffic load but also reduce the network overhead cost. To reach the goal of bandwidth-efficient, our algorithm considers not only link's residual bandwidth but also *betweenness centrality* which indicate the relative importance of a link or node in a topology graph [6]. Furthermore, due to heavy load of branch nodes and the multicast scalability problem in SDN, it is crucial to minimize the number of branch nodes in a multicast routing tree. Therefore, our algorithm considers not only link cost but also branch node cost for bandwidth-efficient and scalability problem. This is to minimize the possibility of requests being rejected when the network becomes overloaded and upgrade the network performance. We obtain a better network performance compared to other traditional routing algorithms such as Shortest Path Tree (SPT), Widest Shortest Path Tree (WSPT), Steiner Tree (ST), Widest Steiner Tree (WST) and Branch-aware Steiner Tree (BST) [7]. Extensive simulations show that our algorithm can lower the unsatisfied request rate and raise the bandwidth satisfaction rate compared to the previous works.

The rest of this paper is organized as follows. We introduce the related work in Section II. In Section III, we present the proposed multicast architecture with SR and multicast routing algorithm. The performance evaluations are presented in Section IV, and Section V concludes this paper.

II. RELATED WORK

2.1 Background

The key idea in SR is to break up the routing path and encode route information into segments in order to control routing paths more flexible and improve network utilization. With SR, a node steers a packet through an ordered list of instructions, called segments. There are two basic types of segments: node segment and adjacency segment. A node segment identifies a node of forwarding plane such as switch or router. Node segment can be represented by switch or router ID

that is globally unique within the network domain. In addition, an adjacency segment represents a local interface of a node. Adjacency segment can be represented by an output port ID that identifies specific egress data link to an adjacent node. Therefore, a segment can have a local or global semantic to an SR node within network domain. SR allows enforcing a flow through any topological path and service chain while maintaining per-flow state only at the ingress node since SR leverages the source routing paradigm. For source routing paradigm of SR, SDN technology provides excellent solution with the global view of controller, which enables more flexible and efficient traffic engineering.

We illustrate the overview of SDN-based SR in Fig. 2.1. The alphabet on switch is switch ID and can be represented as node segment. The number next to switch with specific egress data link to an adjacent node is output port ID which can be represented as adjacency segment. First, the SDN controller will compute explicit routing path based on global view of network topology, resource management, and traffic engineering requirements. Then, SDN controller configures the forwarding table of the ingress switch with an ordered list of segments. The ingress switch adds labels with an ordered list of segments to packet header. These labels can be regarded as intermediate destinations. By default, the packet will be routed to intermediate destinations using the shortest path routing paradigm which is implemented in advanced as forwarding rules in each switch's forwarding table. When the packet reaches the intermediate destination, the top label is popped by the intermediate destination and then the packet is routed to the next segment again along the shortest path [8].

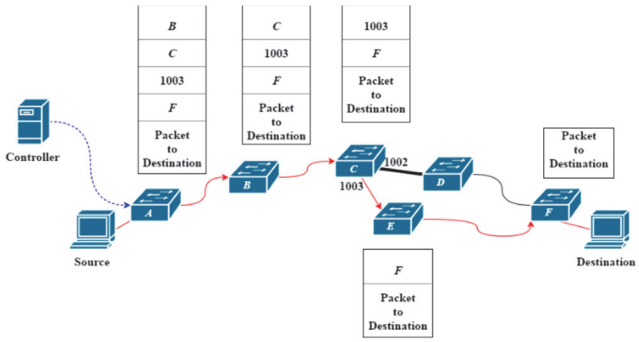

Fig. 2.1 Segment Routing traffic engineering with SDN controller

In SR technology, packets are routed based on the list of segments they carry. It can reduce a great number of forwarding rules in TCAM since there is no need to maintain path state in each switch or router along the path and then improve SDN's scalability problem. Relatively, SDN provides SR a promising source routing paradigm, which makes traffic engineering and resource management more flexible. SDN-based SR is definitely an efficient and agile technology for traffic engineering. However, SR is defined for unicast and the application of the source-route concept to multicast is not in the scope of IETF draft [5]. In Section III, we will design a novel multicast routing architecture by combining SR technique and the concept of branch forwarding in detail.

2.2 Traffic Engineering

Traffic engineering configures the routing scheme to control how traffic is routed across the network. The objective of traffic engineering is to ensure that traffic is managed such that network capacity is efficiently utilized and in a balanced manner. Traditional routing in IP networks is along the shortest paths using link weight as the metric. It has been observed that the shortest path routing can lead to congestion on some links in the network while capacity is available elsewhere in the network. Thus, several techniques are proposed for performing traffic engineering, including adjusting link weights based on traffic patterns, using MPLS (Multi-Protocol Label Switching) to control routing paths and using centralized controllers like an SDN controller to control traffic in a centralized manner, while we use SDN-based SR in the work. Some works show that traffic engineering with SR technology has been implemented and successfully demonstrated in SDN-based testbed [9]. However, our work concentrates on efficient routing algorithm for better network performance.

There are two common transmission in traffic engineering, unicast and multicast. In [10], the authors study unicast traffic engineering in SDN and propose an efficient unicast routing algorithm based on SR technology. Since many previous works have extensively explored the issues on traffic engineering for unicast traffic in SDN, we focus on multicast routing which is more complicated but can relatively save a huge amount of bandwidth than unicast.

2.3 Multicast Tree Algorithms

Here, we introduce several studies in multicast tree algorithm for traffic engineering. One of the simplest algorithm for multicast routing is the Shortest Path Tree (SPT) algorithm [11]. SPT finds the shortest path for each source to destination pair and unions these shortest paths. Another common multicast algorithm is Steiner Tree (ST) algorithm [12]. ST starts from adding source to the tree. For each run, it finds the nearest destination and add the path to tree. ST repeats above step until all destinations added to the multicast tree. The Widest-Shortest Path Tree (WSPT) and Widest-Steiner Tree (WST) algorithms are extension of SPT and ST. The width of a path represents the available bandwidth. Therefore, the objective of WSPT and WST algorithm is to select the path that has the largest amount of residual bandwidth.

In [7], the authors exploit the branch forwarding technique and propose a new multicast tree for SDN, named Branch-aware Steiner Tree (BST). The branch forwarding technique stores the entries in only the branch nodes, instead of every node, of a multicast tree. The concept of branch forwarding is similar to SR. Therefore, we combine them together in our architecture design. The BST problem is difficult since it needs to minimize the number of the edges and the branch nodes in a tree. Nevertheless, considering the nodes together with edges is important for SDN-based traffic engineering because of scalability problem in SDN networks. If we can minimize the number of nodes maintaining forwarding rules, we can improve scalability limitations. Here, we consider BST problem and extend it to minimize the cost of the edges and the branch nodes in our multicast tree algorithm.

III. System Design and Routing Algorithm

3.1 Problem Definition and System Design

Our problem is to decide a routing paradigm for multicast routing in SDN-based networks and ensure that the traffic is routed over feasible paths with specific service guarantees. The two goals of our proposed multicast routing algorithm is bandwidth-efficient and scalability. For the former, our algorithm needs to compute an explicit bandwidth satisfying multicast tree for traffic request of multicast group due to the QoS requirement. For the latter, we proposed a novel multicast routing architecture with SR strategy and the concept of branch forwarding to improve scalability limitation. However, the number of the branch nodes is still a bottleneck since heavy load for branch nodes to store multicast states and process packets by actions such as duplicating and forwarding to different output ports. Therefore, we not only need to consider bandwidth utilization of links for bandwidth-efficient but also need to

concern the loading of branch nodes for load balancing and scalability.

Here, we present a novel SDN-based multicast routing architecture with SR strategy as shown in Fig. 3.1. For the traffic demand of multicast group with a source and multiple destinations, SDN controller will compute an explicit multicast routing tree by the routing module as our proposed algorithm presented in Section 3.2. Then, it will configure the forwarding tables of the ingress or branch switches with an ordered list of segments, which indicate the routing path. When the packets routed to these ingress or branch switches, the switches will modify the packet by encoding the segments as MPLS label stacks to packet header. That is, along the routing path tree, the packet header will be modified afresh when arriving the branch switches. When a packet is arriving to the branch switches, it means that the packet will be directed to multiple output ports for different destinations. In this moment, the packet will be duplicated and sent to different ports, so the branch switch needs to modify packet header with SR technique and route the packet to different paths. Thus, the whole multicast routing tree can be cut by the branch nodes and regarded as separation of SRs for the architecture in Fig. 3.1.

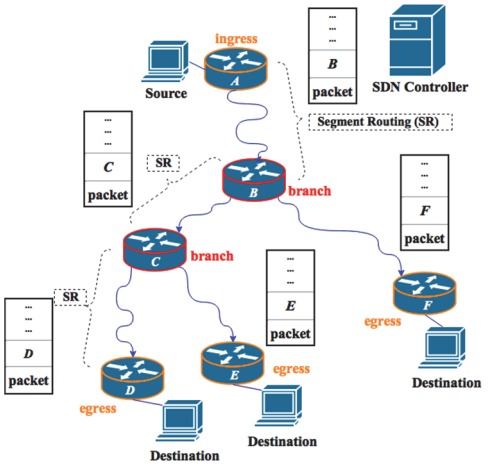

Fig. 3.1 SDN-based multicast routing architecture with SR strategy

For multicast with SR, there is only ingress and branch nodes should be concerned in the multicast routing. In other words, SDN controller needs to compute the routing tree for multicast traffic demands and configuring the forwarding rules for the ingress and branch nodes. Then, the multicast packet will be definitely sent to destinations using SR technology with labels in header and default forwarding rules in switches along the path. The proposed SDN-based multicast routing architecture with SR strategy offers new concept to do packet forwarding while reducing the amount of network states information, which attributes to the TCAM deficiency problem and really improve SDN's scalability problem especially for multicast cases.

3.2 Routing Algorithm

Before presenting the detail of our Bandwidth-efficient Branch-aware Segment Routing Tree (BBSRT) algorithm, we define the notations used in the paper. An SDN network topology can be represented as a graph $G = (V, E)$, where V is the set of vertex as switch nodes and E is the set of edges as data links. In routing, the request of traffic for multicast group can be represented as $MG = (S, D)$, where S is the source node and D is the list of destination nodes. We define two weight functions on graph G, the weight function on each edge $e \in E$ called link weight $W_L(e)$ and the weight function on each vertex $v \in V$ called node weight $W_N(v)$. In our algorithm, we use these weight functions for computing the weight for nodes and edges. The weighted graph can be constructed by these weight functions, which imply the network states. We then construct the multicast tree for traffic request by our algorithm to realize scalable and bandwidth-efficient multicast routing.

In our algorithm, we consider network information such as links residual bandwidth, node's flow entry, and betweenness centrality together to compute a routing tree for multicast. To consider the network information in our algorithm, we generate the weighted graph before constructing multicast tree. We will compute the weight for each node and link based on their weight functions we defined. The weight for links usually used in routing is easily understood for considering network resource when selecting paths and constructing multicast tree. Additionally, we add the weight for nodes in our algorithm for considering the loading of nodes. We state for scalability issue and set the goal to minimize the number of branch nodes previously. However, the number of flow entries in a switch node is limited. There may exist many multicast groups in a network at the same time, so we need to distribute network resource usages in order and cannot to always select the same branch node for lots of multicast groups to prevent critical network resource from being exhausted early. Thus, we combine the number of branch nodes and loading of branch nodes together, and then define the node weight function.

The weight function is defined by two parts, betweenness centrality and congestion index. Centrality is often used for determining the relative importance of a link or node in a topology graph. Link or node with a high betweenness centrality are usually key players in a social network or a bottleneck in a communication network. The betweenness centrality of a node v stands for the average ratio that a node s reach a node t via the shortest path in the network that needs to pass through a specific node v. The betweenness centrality of a specific node v, $BC_N(v)$ is given by equation (1), where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of shortest paths between s and t that pass through node v. In the same way, the betweenness centrality of a specific link e, $BC_L(e)$ is given by equation (2). The σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(e)$ is the number of shortest paths between s and t that pass through link e [6].

$$BC_N(v) = \sum_{s \neq v \neq t} \sigma_{st}(v)/\sigma_{st} \quad (1)$$

$$BC_L(e) = \sum_{s \neq e \neq t} \sigma_{st}(e)/\sigma_{st} \quad (2)$$

High betweenness centrality means that the node or link has the high opportunity for traffic to go through. If we assign these high betweenness centrality forwarding devices for traffic request all the time for the shortest path routing, these devices will load heavily and stuck. Since a node or link with higher betweenness centrality have higher influence in the network and thus is prone to become the bottleneck, we should include betweenness centrality in the weight functions to prevent the bottleneck in the multicast routing.

Congestion index which stands for the current loading status, is another index for defining the weight functions. Since current loading is also an important factor for the routing performance, we also consider congestion index of each node and link. For a data link e, the congestion index of the link e is denoted as $C_L(e) = F(e)/B(e)$, where $F(e)$ denotes the total amount of traffic flows carried by the link e and $B(e)$ denotes the residual bandwidth of the link e. The definition of the link congestion index $C_L(e)$ is an increasing and convex function [13, 14]. When the amount of traffic flows passed through the link e approaches to its capacity, the function $C_L(e)$ will increase quickly. In other words, heavy loaded links will get larger congestion index computed by $C_L(e)$. Since the main purpose of using congestion index in the weight functions is to make good use of the link resource and load balancing, we define and let $C_L(e)$ increase

rapidly as link utilization grows beyond a threshold. As a result, we can track the state of congestion based on information of link utilization. As for a switch node, the congestion index of the node v is denoted as $C_N(v) = R(v)/E(v)$, where $R(v)$ denotes the total amount of traffic rules on flow table of the node v and $E(v)$ denotes the residual flow entry of the node v. The definition of the node congestion index $C_N(v)$ is also an increasing and convex function for the similar reason as the links.

Betweenness centrality can help to defer loading on highly critical links and prevent critical network resource from being exhausted early. Congestion Index can help to track the current utilization and prevent congestion in networks. We combine them together to define our weight functions for the algorithm in order to choose the nodes or links that can balance the loads across the network. We then use appropriate parameters α and β to combine the two indices together to compose the weight for link and node as equation (3) and (4).

$$W_L(e) = C_L(e)\ \alpha + BC_L(e)\ (1-\alpha),\ 0< \alpha <1 \qquad (3)$$

$$W_N(v) = C_N(v)\ \beta + BC_N(v)\ (1-\beta),\ 0< \beta <1 \qquad (4)$$

After defining the weight functions, SDN controller with global view can collect network status and generate the weighted graph G. In our algorithm, we will find the shortest paths for each source destination pair (s, d) at first where s is a source node and $d \in D$ is a destination node in a multicast group MG. The shortest path here indicates the path with minimum total cost concerning only the link weight in equation (3). The minimum link weight path guarantees bandwidth efficiency since the weight function is concerned with betweenness centrality and congestion index explained before. In our algorithm, we want to find the first k shortest paths rather than only one shortest path [15]. The first k shortest paths reserve the agility for multicast routing. If we select only one the shortest path for each source destination pair to construct multicast tree, the tree will be limited to the links with the most bandwidth-efficient path. Therefore, we broaden the range for selection with the k shortest paths at first. Then, we will concern for the scalability problem and load balancing of switch nodes by considering branch node weight. In the beginning, we sort the destinations by cost of their shortest paths for each source destination pair. The original list of destination nodes D is sorted as D_{sort} in an increasing order. We then add these destination nodes into multicast tree in order. In our algorithm, we will check the k shortest paths for each source destination pair, select one of them, and add it into multicast tree.

For each destination, there are k shortest paths for us to select. We will compare each path by computing extra cost with branch node weight then select the least cost one. In the SR technology, the branch nodes are responsible for specific SR rules to push ordered list of labels to packets and forward them to different output ports. The routing policy in the other nodes is based on default rules. That is, only branch nodes should be concerned with loading and scalability problem in our architecture. Let $P_k(d)$ denote the k-th shortest path of destination d and $S_k(x, d)$ denote the corresponding sub-path from the constructed multicast tree T to d with the intersection node x. We compute extra cost $EC_k(x, d)$ for the k-th shortest path, which is computed as the sub-path cost $SC_k(x, d)$ plus the branch node weight $W_N(x)$ of intersection node x. Note that, we only include the new branch node weight in computing extra cost. We give non-negative parameters w_1 and w_2 to conduct the weight between link cost and branch node cost. Thus, the extra cost is denoted as $EC_k(x, d) = SC_k(x, d)\ w_1 + W_N(x)\ w_2$.

The example in Fig. 3.2 shows the algorithm of selecting path for destination $D3$ among the first k (= 2) shortest paths when constructing multicast tree T. The left-part tree represents the already constructed multicast tree including source S and two destinations $D1$ and $D2$ with branch node a in Fig. 3.2(a). We then add the path for destination $D3$ into multicast tree. The first two shortest paths from S to $D3$ are $P_1(D3)$ = /*S~b-d-e-f-D3*/ and $P_2(D3)$ = /*S~b~a-c-e-f-D3*/. We will choose one of them based on computing extra link weight and branch node weight. In Fig. 3.2, the number next to the link represents the link weight.

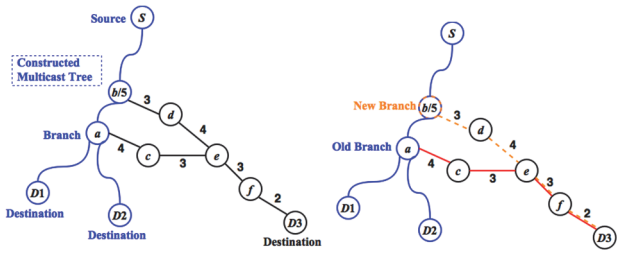

Fig. 3.2 Example of selecting path among the first k (=2) shortest paths for a destination

In Fig. 3.2(b), for the first shortest path $P_1(D3)$, we find the sub-path from destination $D3$ to tree T as $S_1(b, D3)$ = /*b-d-e-f-D3*/ and the cost is $SC_1(b, D3) = 3+4+3+2 = 12$ with intersection node b. If we choose this sub-path, node b will become a new branch node then cause more branch nodes and loading. Therefore, we should include the new branch node weight in computing extra cost. Assume the node weight of the new branch node b is $W_N(b) = 5$ which is computed before using weight function in equation (4). The extra cost is then computed as $EC_1(b, D3) = SC_1(b, D3)\ w_1 + W_N(b)\ w_2 = 12+5 = 17$, where we set $w_1 = w_2 = 1$ for balancing consideration of scalability and bandwidth-efficient both. For the second path $P_2(D3)$, we find the sub-path $S_2(a, D3)$ = /*a-c-e-f-D3*/ and the cost is $SC_2(a, D3) = 4+3+3+2 = 12$ with intersection node a. Since the node a is already an old branch node, we will not compute the weight cost repeatedly. The extra cost is then computed as $EC_2(a, D3) = SC_2(a, D3)\ w_1 = 12$. We then compare the cost of the k paths and choose one sub-path $S_{min}(D3)$ with minimum extra cost $EC_{min}(D3)$. Here, $S_{min}(D3) = S_2(a, D3)$ = /*a-c-e-f-D3*/ with minimum extra cost $EC_{min}(D3) = 12$. Therefore, we add the sub-path /*a-c-e-f-D3*/ into multicast tree T and go on to next destination if there exists. For each destination d in the sorted list D_{sort}, we add the minimum extra cost sub-path $S_{min}(d)$ to multicast routing tree T in order until every destination in multicast group MG is added into the multicast tree. The proposed BBSRT algorithm is summarized as follows.

Algorithm 1: *Generate the Weighted Graph G*

1. Compute the link betweenness centrality $BC_L(e)$.
2. Compute the node betweenness centrality $BC_N(v)$.
3. For each edge $e \in E$
4. Compute the link congestion index: $C_L(e) = F(e)/B(e)$;
5. Compute the link weight: $W_L(e) = C_L(e)\ \alpha + BC_L(e)\ (1-\alpha),\ 0< \alpha <1$;
6. For each node $v \in V$
7. Compute the node congestion index: $C_N(v) = R(v)/E(v)$;
8. Compute the node weight: $W_N(v) = C_N(v)\ \beta + BC_N(v)\ (1-\beta),\ 0< \beta <1$;
9. Return G

Algorithm 2: *Find a minimum weight multicast tree for a multicast group MG = (S, D) with weighted graph G*

1. **for** destination $d \in D$ **do**
   /* only considering link weight here */
2. Find the first k shortest paths and their path cost from source s to d by algorithm [15].

3.**end for**
4.**sort** destinations D as D_{sort} according to their shortest path cost.
/* Constructing multicast tree T */
5.add source s to T;
6.**for** each destination $d \in D_{sort}$ in increasing order **do**
7. **for** $i = 1$ to k **do**
8.find the k-th shortest path $P_i(d)$ and its corresponding sub-path $S_i(x, d)$ from d to T with intersection node x;
/* compute extra cost with branch node weight */
9. $EC_i(x, d) \leftarrow SC_i(x, d)\ w_1$
10. **if** x is a new branch node in T **then**
11. $EC_i(x, d) \leftarrow EC_i(x, d) + W_N(x)\ w_2$
12. **end if**
13. **end for**
14. find the sub-path $S_{min}(d)$ with the minimum extra cost $EC_{min}(d)$
15. add the sub-path $S_{min}(d)$ to T
16. **end for**
17. Return T and update the residual link bandwidth and node flow entry for the network graph

IV. Performance Evaluation

In this Section, we compare the performance of our proposed multicast routing tree algorithm BBSRT with other traditional algorithms Shortest Path Tree (SPT), Widest Shortest Path Tree (WSPT), Steiner Tree (ST), Widest Steiner Tree (WST) and Branch-aware Steiner Tree (BST) [7] in terms of number of branch nodes, rejection rate and network throughput. We implement our algorithm using JAVA language and model different network environments and traffic flows for simulations. The experiments are conducted under different network sizes and traffic requests. We create the network topology for experiments using Waxman method [16]. In the simulations, we randomly create the network topology and generate multicast groups of traffic requests for the experiments.

For each multicast group, there is a source and multiple destinations and source needs to send packet to every destination with flow size ranges from 10MB/s to 100MB/s. The default link capacity is 1GB/s. We generate some basis traffic and let critical nodes and links with higher *betweenness* centrality have more opportunity to consume more network resource in order to create a more realistic network environment.

In the following simulations, we set k to 5 for the k shortest paths. We also set w_1 and w_2 equal to one when computing extra cost to balance the weight for link and node. From the weight functions defined in equations (3) and (4), the smaller value of α and β means that we focus on critical links or nodes in network topology to avoid critical network resource to be exhausted early. In contrast, the larger value of α and β means that we put more emphasis on current loading of network resource to prevent congestion. In the following simulations, we then set the parameters α and β equal to median value 0.5 in order to balance the consideration between rejection rate and extra consumption of bandwidth and delay.

One of our goals is to minimize the need for keeping a great amount of network states information, which attributes to the TCAM deficiency and scalability problem in SDN. Branch nodes are responsible for specific SR rules to push routing labels and forward packets to different output ports. Therefore, the number of branch nodes imply the cost to maintain SR rules for routing in our novel multicast architecture. Fig. 4.1 shows the number of branch nodes of SPT, WSPT, ST, WST, BST and our proposed algorithm BBSRT under different multicast group size with fixed network size $N = 400$. SPT performs the worst to generate the multicast tree with the highest number of branch nodes. Since SPT constructs the multicast tree by union every shortest path from source to each destination, the topology of the multicast tree can be expanded larger than ST, which guarantees the minimum spanning tree. Since ST constructs the multicast tree by adding the nearest destination to tree, there is probability for some destinations to share the same branch node. The simulation results show that the number of branch nodes for SPT is more than that of ST.

WSPT and WST are the extension of SPT and ST, respectively. Widest multicast tree algorithms select the widest path, which contains more bandwidth resource to avoid congestion. However, the number of branch nodes for widest multicast tree algorithms such as WSPT and WST is larger than their original SPT and ST. BST is the algorithm for minimizing the edge and branch node numbers, and there is some optimization phase to adjust the topology of tree to share more branches in order to decrease the branch node numbers. The result shows that BST performs the best with the fewest branch nodes. However, our proposed algorithm not only consider the number of branch nodes for scalability issue but also concern for the network resource for branch nodes. Therefore, the simulation result of average rejection rate and throughput shows that our proposed algorithm is outstanding than BST as shown in the following figures.

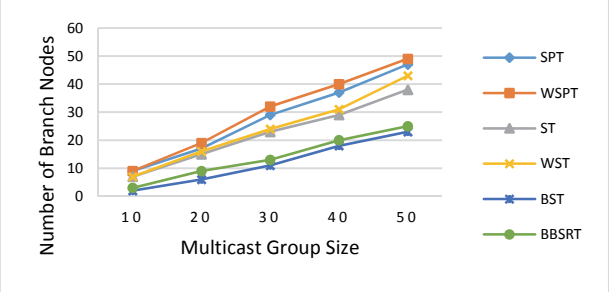

Fig. 4.1 Number of branch nodes vs. multicast group size

Fig. 4.2 shows the average rejection rate of six algorithms with fixed network size $N = 400$ and randomly generated multicast group size ranges from 20 to 40 for every request. The abscissa of Fig. 4.2 represents the number of total requests in the whole simulation. SPT and ST perform the worst because of always selecting the shortest path with the minimum hop count without considering network bandwidth and switch node resource. WSPT and WST performs better than SPT and ST by selecting the widest path with more bandwidth resource to prevent network congestion. BST and our proposed algorithm perform better than other algorithms because of considering the factor of bandwidth and node together. However, our proposed algorithm makes more effort on load balancing of branch nodes. Therefore, the proposed algorithm outstands in load balancing of branch nodes and success in reducing the rejection rate due to insufficient flow entries.

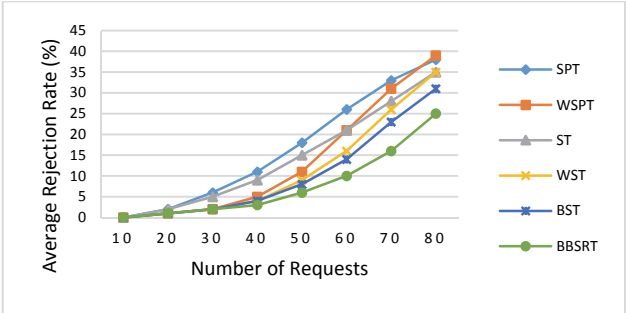

Fig. 4.2 Average rejection rate vs. number of requests

Fig. 4.3 shows the average network throughput of the six algorithms with fixed network size N = 400 and randomly generated multicast group size ranges from 20 to 40 for every request. Network throughput is the amount of satisfied bandwidth routed successfully as a function of the total amount of requested bandwidth arrived at the network. In general, the average network throughput increases when the requested bandwidth increases. We can observe that the trend of average rejection rate and average network throughput is similar. Our method gets the highest network throughput and this indicates that our algorithm performs better than all the other benchmark algorithms. Fig. 4.4 shows the comparison between various schemes with incremental network sizes. The algorithms have the similar behaviors to Fig. 4.3 on different network size. In conclusion, our proposed algorithm performs the best performance no matter the size of the network.

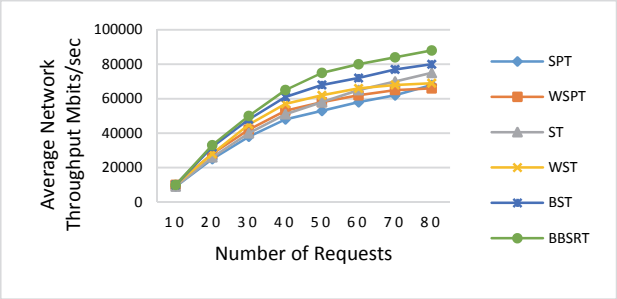

Fig. 4.3 Average network throughput vs. number of requests

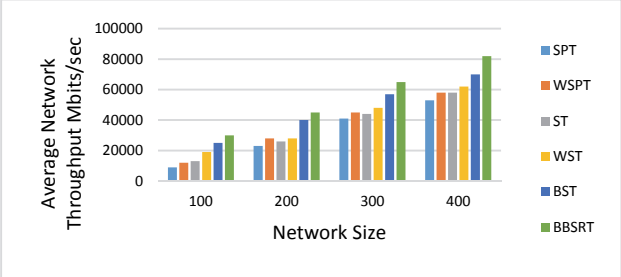

Fig. 4.4 Average network throughput under different network sizes

We also conduct our experiment in real network from the Internet Topology Zoo [17]. The Internet Topology Zoo is a store of network data created from the information that network operators make public. It is the most accurate large-scale collection of network topologies available. We choose the Kentucky Datalink (Kdl) network for simulation. Kdl is a large-scale network topology with 754 nodes and 895 links. The simulation result shows that our proposed algorithm outperforms other algorithms in real network with the best network throughput as shown in Fig. 4.5.

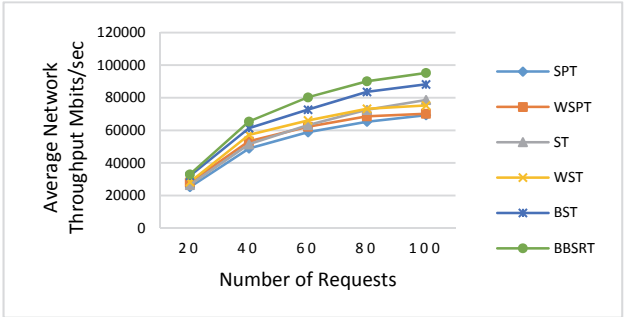

Fig. 4.5 Kdl network: Average network throughput vs. number of requests

V. Conclusion

In this paper, we focus our study on multicast traffic engineering issues in SDN with SR. One of our goals is to construct a bandwidth-efficient multicast routing tree for the traffic requests of multicast group to minimize the possibility of rejecting traffic demands and increase the total network throughput. Another goal is to solve scalability problem for traffic engineering in SDN. Therefore, we propose an efficient heuristic multicast routing algorithm considering not only link criticality and residual bandwidth but also node loading and scalability. The simulation results show that our method outperforms other routing algorithms in terms of average rejection rate and network throughput under different network sizes.

References

[1] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," *Proceedings of IEEE INFOCOM,* pp. 2211-2219, Turin, April 2013.

[2] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," *IEEE Network,* vol. 14, no. 1, pp. 78-88, 2000.

[3] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine,* vol. 51, no. 2, pp. 136-141, February 2013.

[4] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," *Proceedings of IEEE INFOCOM,* pp. 545-549, Turin, April 2013.

[5] *Segment Routing Architecture*. Available: https://tools.ietf.org/html/draft-ietf-spring-segment-routing-08

[6] O. Green, R. McColl, and D. A. Bader, "A Fast Algorithm for Streaming Betweenness Centrality," *International Confernece on Social Computing,* pp. 11-20, Amsterdam, September 2012.

[7] L. H. Huang, H. J. Hung, C. C. Lin, and D. N. Yang, "Scalable and bandwidth-efficient multicast for software-defined networks," *IEEE Global Communications Conference,* pp. 1890-1896, Austin, TX, December 2014.

[8] R. Bhatia, F. Hao, M. Kodialam, and T. V. Lakshman, "Optimized network traffic engineering using segment routing," *IEEE Conference on Computer Communications,* pp. 657-665, Kowloon, April 2015.

[9] L. Davoli, L. Veltri, P. L. Ventre, G. Siracusano, and S. Salsano, "Traffic Engineering with Segment Routing: SDN-Based Architectural Design and Open Source Implementation," *Fourth European Workshop on Software Defined Networks,* pp. 111-112, Bilbao, 2015.

[10] M. C. Lee and J. P. Sheu, "An efficient routing algorithm based on segment routing in software-defined networking," *Computer Networks,* vol. 103, no. 5, pp. 44-55, July 2016.

[11] P. Narvaez, S. Kai-Yeung, and T. Hong-Yi, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Transactions on Networking,* vol. 8, no. 6, pp. 734-746, December 2000.

[12] E. Aharoni and R. Cohen, "Restricted dynamic Steiner trees for scalable multicast in datagram networks," *Proceedings of IEEE INFOCOM,* vol. 2, pp. 876-883, Kobe, April 1997.

[13] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," *Proceedings of IEEE INFOCOM,* vol. 2, pp. 519-528, Tel Aviv, March 2000.

[14] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE Journal on Selected Areas in Communications,* vol. 20, no. 4, pp. 756-767, May 2002.

[15] J. Y. Yen, "Finding the K Shortest Loopless Paths in a Network," *Management Science,* vol. 17, no. 11, pp. 712-716, July 1971.

[16] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications,* vol. 6, no. 9, pp. 1617-1622, December 1988.

[17] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications,* vol. 29, no. 9, pp. 1765-1775, October 2011.