Constructing Multiple Steiner Trees for Software-Defined Networking Multicast

Jehn-Ruey Jiang
National Central University
Taoyuan City, Taiwan
jrjiang@csie.ncu.edu.tw

Szu-Yuan Chen
National Central University
Taoyuan City, Taiwan
j35682368@yahoo.com.tw

ABSTRACT
A heuristic algorithm is proposed in this paper to construct a set of Steiner trees for multicast with multiple sources under the software-defined networking (SDN) architecture, where each source is associated with a group of receivers. It is based on the extended Dijkstra's shortest path algorithm and the modified selective closest terminal first Steiner tree algorithm. The former algorithm considers not only the edge weights but also the node weights to form data routing path with the shortest delay. The latter algorithm regards nodes with shorter paths to the source to have higher priority in the priority queue for growing Steiner trees. In this way, the number of Steiner tree edges is reduced and thus the bandwidth consumption is cut down. The proposed algorithm is simulated by the EstiNet emulator along with a Ryu controller for different multicast scenarios. The simulation results are compared with those of related algorithms in terms of the source-to-receiver delay and the bandwidth consumption to show the advantage of the proposed algorithm.

Categories and Subject Descriptions
C.2 [Computer-communication networks]: Network Architecture and Design.

Keywords
Software-Defined Networking; Multicast; Dijkstra's Shortest Path Algorithm; Steiner Tree

1. INTRODUCTION
Software-Defined Networking (SDN) is a concept advocated by the Open Network Foundation (ONF) [1] to decouple the control plane and the data plane of network devices. SDN switches are responsible for forwarding the packets on the data plane, while an SDN controller or a set of SDN controllers is responsible of collecting network information from switches and configuring switches' forwarding tables (also called flow tables) which all switches base on to process data packets. In this manner, SDN users can composite application programs running on top of the controller to monitor and manage the whole network in a centralized and timely way.

CFI'16, June 15-17, 2016, Nanjing, China.
 ISBN 978-1-4503-4181-3/16/06$15.00
DOI: http://dx.doi.org/10.1145/ 2935663.2935665

Google has applied an SDN architecture to its private WAN called B4 [2] for improving the network performance. Consequently, the network link utilization is driven from 30-40% to near 100% by centralized traffic engineering (TE) based on the architecture. Besides the application in Google, many SDN-based applications [3][4] have been proposed, such as load balancing, access control, and multicast.

Multicast is a fundamental communication function, in which a data packet is sent by a *source*, replicated at intermediate devices for forwarding to multiple outgoing links, and eventually delivered to all *receivers* of a *multicast group*. It can be applied to applications like IPTV, the video conference, and live streaming, etc. Multicast routing can be regarded as constructing a multicast tree that is rooted at the source and spans all receivers. The PIM-SM protocol [5] finds a shortest path from the source to every receiver and puts all paths jointly to have a *shortest path tree* as the multicast tree. However, the shortest path tree may contain many links which consume a large volume of bandwidth. On the contrary, the *minimum Steiner tree* [6] contains the smallest number of links to span all receivers; it is therefore a good alternative to be the multicast tree that may consume less bandwidth.

This paper proposes a heuristic algorithm to construct a set of multicast trees by combining the shortest path trees and the minimum Steiner trees. The constructed trees are then applied to achieve multicast with multiple sources under the SDN architecture. The proposed algorithm takes advantage of both the extended Dijkstra's algorithm [7] and a Steiner tree forming algorithm called the Selective Closest Terminal First (SCTF) algorithm [8]. Modeling the network as a directed graph, the extended Dijkstra's algorithm considers not only the edge weight but also the node weight to obtain the shortest path from a single source node to every other node. The SCTF algorithm uses a heuristic to build a Steiner tree with approximately the minimum total cost (weight). This paper modifies the heuristic to fit for the SDN multicast scenarios. The proposed algorithm is simulated by the EstiNet emulator [9] along with a Ryu controller [10] for different multicast scenarios. The simulation results are compared with those of related algorithms in terms of the source-to-receiver delay and the total bandwidth consumption to show the advantage of the proposed algorithm.

The rest of this paper is organized as follows. Section 2 describes some related work. The proposed algorithm is detailed in Section 3. Section 4 demonstrates simulation results of the proposed algorithm. And finally, Section 5 concludes the paper.

2. RELATED WORK
2.1 SDN
SDN is the concept to separate the control plane and the data plane, so that underlying data-plane switching devices (called *switches*)

are controlled by a centralized control-plan device (called *controllers*) on top of which different software entities (called *applications*) can be developed and run. Figure 1 depicts the logical view of the SDN architecture [11]. SDN allows the network administrator to write applications to cooperate with the controller through the *northbound interface* to interact with switches through the *southbound interface* for providing network services, including routing, access control, load balancing, multicast, and other traffic engineering tasks.

OpenFlow [12] is the most well-known southbound interface protocol used between the controller and the switch. As depicted in Figure 2, a switch has one or more *flow tables* and/or *group tables*, and a flow table entry consists of match fields, counters, instructions, and so on. A controller can update, add and delete flow entries in the flow table both reactively and proactively.

On receiving a packet, a switch first matches its header with match fields of every entry in the flow table(s). The matching process begins in the first table and continues subsequently to additional tables. It is prioritized; that is, the first matched entry is returned and the matching process stops. If a matched entry is found, the counter associated with the entry is updated for the purpose of traffic statistics, and associated instructions are executed to complete specific actions, such as forwarding the packet to another switch via an outgoing port, and dropping the packet, etc. If no match is found in any flow table, then the packet may be dropped or forwarded to the controller.

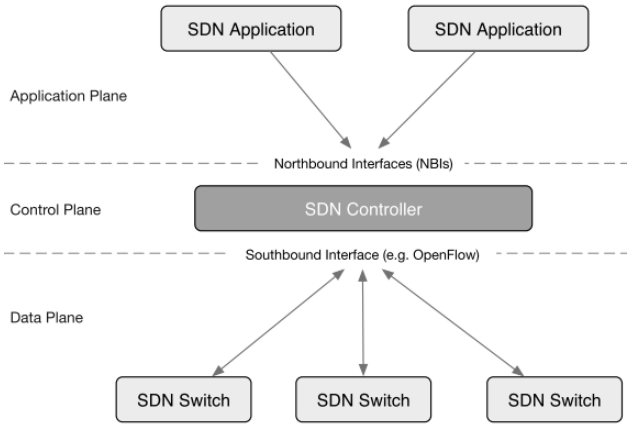
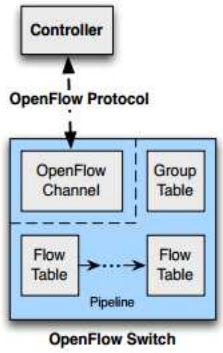


Figure 1. The illustration of the SDN architecture [11].



Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Figure 2. The OpenFlow controller and the switch and the flow table entry fields [12].

2.2 The Extended Dijkstra's Algorithm

Given a weighted, directed graph $G=(V, E)$ and a single source node s, the classical Dijkstra's algorithm [13] can return a shortest path from the source node s to every other node, where V is the set of nodes and E is the set of edges, each of which is associated with a non-negative weight (or cost). In the original Dijkstra's algorithm, nodes are associated with no weight. The paper [7] extends the original algorithm to consider both the edge weights and the node weights for end-to-end routing. The extended Dijkstra's algorithm (EDA) is also applied to achieve load balancing and multicasting in [14].

Figure 3 shows the extended Dijkstra's algorithm, whose input is a given graph $G=(V, E)$, the edge weight setting ew, the node weight setting nw, and the single source node s. The extended algorithm uses $dist[u]$ to store the distance of the current shortest path from the source node s to the destination (i.e., receiver) node u, and uses $pred[u]$ to store the predecessor node preceding u on the current shortest path. Initially, $dist[s]=0$, $dist[u]=\infty$ for each $u\in V$, $u\neq s$, and $pred[u]$=null for each $u\in V$. The algorithm finally returns a set SP of shortest paths from s to all other nodes according to $dist[u]$ and $pred[u]$ for each $u\in V$, $u\neq s$.

The extended Dijkstra's algorithm is very useful in deriving the best routing path to send a packet from a specific source node to a destination node for the SDN environment in which significant latency occurs when the packet goes through intermediate nodes and edges (or links). As shown below, the paper [7] defines the edge weights and node weights so that the extended Dijkstra's algorithm can be applied to derive routing path for SDN environments. Assume the SDN network topology is derived and modeled as a weighted, directed, and connected graph $G=(V, E)$. For a node $v\in V$ and an edge $e\in E$, let $Flow(v)$ and $Flow(e)$ denote the set of all the flows passing through v and e, respectively, let $Capacity(v)$ be the *capacity* of v (i.e., the number of bits that v can process per second), and let $Bandwidth(e)$ be the bandwidth of e (i.e., the number of bits that e can transmit per second). The node weight $nw[v]$ of v is defined according to Eq. (1), and the edge weight $ew[e]$ of e is defined according to Eq. (2).

$$nw[v] = \frac{\sum_{f\in Flow(v)} Bits(f)}{Capacity(v)}, \quad (1)$$

where $Bits(f)$ stands for the number of flow f's bits processed by node v per second.

$$ew[e] = \frac{\sum_{f\in Flow(e)} Bits(f)}{Bandwidth(e)}, \quad (2)$$

where $Bits(f)$ stands for the number of flow f's bits passing through edge e per second.

Note that we can easily obtain the number of a flow's bits processed by a node or passing through an edge with the help of the "counters" field of the OpenFlow switches' flow tables. Also note that the numerators in Eq. (1) and Eq. (2) are of the unit of "bits", and the denominators are of the unit of "bits per second". Therefore, the node weight $nw[v]$ and the edge weight $ew[e]$ are of the unit of "seconds". When we accumulate all the node weights and all the edge weights along a path, we can obtain the end-to-end latency from one end (source) to the other end (destination or receiver) of the path.

Algorithm: EDA (Extended Dijkstra's Algorithm)

Input: $G=(V, E)$, ew, nw, s //$G=(V, E)$ is a graph with edge and node weights stored in ew and nw, and s is the source

Output: SP //SP is the set of shortest paths from s to all other nodes

```
1:  dist[s]←0; dist[u]←∞, for each u≠s, u∈V
2:  insert u with key dist[u] into priority queue Q, for each u∈V
3:  while (Q≠∅)
4:    u←Extract-Min(Q)
5:    for each v adjacent to u
6:      if dist[v] > dist[u]+ew[u,v]+nw[u] then
7:        dist[v]←dist[u]+ew[u,v]+nw[u]
8:        pred[v]←u  //v's predecessor in the shortest path is u
9:  calculate the shortest path from s to u to add into set SP according
    to dist[u] and pred[u], for each u∈V, u≠s,
10: return SP
```

Figure 3. The extended Dijkstra's algorithm.

2.3 The Minimum Steiner Tree Algorithm

Given an undirected, weighted graph $G=(V, E)$ and a set R of nodes, called *terminals*, where $R \subseteq V$, the minimum Steiner tree problem is to find a minimum-weight tree, called the minimum Steiner tree, to span all terminals in R. When $R=S$, the minimum Steiner tree is actually the minimum spanning tree. Furthermore, when each edge cost is 1, the minimum Steiner tree is the tree with the minimum number of edges to span all terminals. The minimum Steiner tree problem has been proven to be NP-hard. Thus, there probably exists no deterministic algorithm running in polynomial time complexity to solve such a problem. However, many polynomial-time-complexity heuristic algorithms have been proposed to solve the problem.

The SCTF (Selective Closest Terminal First) algorithm [8], shown in Figure 4, is one of the heuristic algorithms to solve the minimum Steiner tree problem. This paper focuses on the SCTF algorithm for the following reasons. First, the algorithm regards the network as a directed (instead of undirected) graph to solve the problem. It thus can be applied to more practical network environments where the edges between two nodes can be asymmetric (e.g., the edges may have different bandwidth). Second, the algorithm is parameterized to select between fast algorithm execution time and the low tree weight.

The basic concept of the SCTF algorithm is shown below. The SCTF algorithm first finds the shortest path from the source to every terminal for obtaining the "shortest" shortest path P^*. All the nodes in P^* are added into the Steiner tree T. The algorithm then puts all nodes in P^* into the priority queue Q according to the priority order: source > terminal node > non-terminal node. It then constructs the shortest path from every of the first κ (kappa) nodes in Q to every terminal that is not yet included in the Steiner tree. The terminal z associated with the "shortest" shortest path (SSP) is then added into the Steiner tree, which accounts for the algorithm name. Afterwards, the algorithm derives from P^* a subpath, called *Branch*, from u to z such that u is the only nodes in T. Note that the SSP calculation is performed only for the first κ nodes in Q, so P^* may go through some nodes already in T and needs to be pruned as *Branch*. Finally, nodes and edges in *Branch* are added into T. It is worthwhile mentioning that κ is used to limit the algorithm computation overhead. The larger κ is, the heavier the computation is. However, larger κ values usually lead to better results of Steiner trees.

Algorithm: SCTF (Selective Closest Terminal First) algorithm

Input: $G=(V, E)$, ew, s, $R=\{r_1,\ldots,r_n\}$ and κ //$G=(V, E)$ is a graph with edge weights stored in ew, s is the source, R is the group (set) of receivers associated with s, and κ is a control knob for the priority queue

Output: $T=(V_T, E_T)$ //$T=(V_T, E_T)$, where $V_T \subseteq V$ and $E_T \subseteq E$, is a Steiner tree rooted at s and spanning all nodes in R

```
1:  Q←{s}, V_T←{s}, E_T←∅   //Q: priority queue
2:  while (R≠∅) do
3:    B ← the set of the first Min(κ, |Q|) nodes in Q
4:    P* ← ShortestPath(x, y), where x∈B and y∈R are arbitrary
5:    for each x in B do
6:      for each y in R do
7:        if w(P ← ShortestPath(x, y)) < w(P*)
8:          P* ← P  //P* is the "shortest" shortest path
9:    z ← the terminal at which P* terminates
10:   Branch ← subpath(u, z) such that only u is in V_T
11:   insert nodes in Branch into Q
12:   V_T ← V_T ∪ {nodes in Branch}
13:   E_T ← E_T ∪ {edges in Branch}
14:   R_T ← R_T − {terminals in Branch}
15: return T
```

Figure 4. The SCTF algorithm.

3. THE PROPOSED ALGORITHM

The proposed algorithm is called M-SCTF/EDA, which stands for the modified SCTF algorithm with EDA (extended Dijkstra's algorithm). It is intended for the scenario of multiple sources, each of which is associated with a group of receivers for multicasting. It constructs a set of multiple Steiner trees such that every tree is rooted at a source and spans all receivers of the group associated with the source. The purpose of the algorithm is to find multiple Steiner trees such that the average source-to-receiver delay and the bandwidth consumption are both kept as low as possible.

The two metrics used in the proposed algorithm are described in the following. The first one is the source-to-receiver delay defined in Eq. (3).

$$\frac{\sum_{r \in R} Delay(r)}{|R|}, \tag{3}$$

where $Delay(r)$ is the delay between receiver r and its associated source, and R is the set of all receivers.

The second one is the bandwidth consumption defined in Eq. (4).

$$\frac{\sum_{e \in E} \sum_{f \in Flow(e)} Bits(f)}{\sum_{e \in E} Bandwidth(e)}, \tag{4}$$

where E is the set of all edges, $Flow(e)$ is the set of all flows passing through edge e, and $Bits(f)$ is the number of f's bits passing through e per second.

Note that the bandwidth consumption is a ratio between 0 and 1, which stands for the ratio of total bits transmitted per second to the total bandwidth. Also note that the bandwidth consumption does not equal the average edge (link) utilization of all edges.

Algorithm: M-SCTF/EDA

Input: $G=(V, E)$, ew, nw, n, $S=\{s_1,\ldots,s_n\}$, $M=\{R_1,\ldots,R_n\}$, κ
//$G=(V, E)$ is a graph with edge and node weights stored in ew and nw, n is the number of sources, R_i is the i^{th} group (set) of receivers associated with s_i, $1\le i\le n$, and κ is a control knob for the priority queue
Output: $T=\{T_1,\ldots,T_n\}$ //T is a set of Steiner trees, where $T_i=(V_i, E_i)$, $V_i \subseteq V$ and $E_i \subseteq E$, is a tree rooted at s_i and spanning all nodes in R_i, $1\le i\le n$

1: $Q_i \leftarrow \{s_i\}$, $V_i \leftarrow \{s_i\}$, $E_i \leftarrow \emptyset$, for every i, $1\le i \le n$;
   //Q_i: i^{th} priority queue
2: **while** ($M\neq\emptyset$) do
3: $r_a \leftarrow Arg\ Min_{r \in \left(\cup_{R_j \in M} R_j\right)} w(\text{EDA}(r\text{'s source}, r))$, where $r_a \in R_a$
4: $M \leftarrow M - R_a$
5: **while** ($R_a\neq\emptyset$) do
6: $B \leftarrow$ the set of the first $Min(\kappa, |Q|)$ nodes in Q
7: $P^* \leftarrow \text{EDA}(x, y)$, where $x \in B$ and $y \in R_a$ are arbitrary
8: **for** each x in B do
9: **for** each y in R_a do
10: **if** $w(P \leftarrow \text{EDA}(x, y)) < w(P^*)$
11: $P^* \leftarrow P$ // P^* is the "shortest" shortest path
12: $z \leftarrow$ the terminal at which P^* terminates
13: $Branch \leftarrow \text{subpath}(u, z)$ such that only u is in V_a
14: insert nodes in $Branch$ into Q_a
15: $V_a \leftarrow V_a \cup \{\text{nodes in } Branch\}$
16: $E_a \leftarrow E_a \cup \{\text{edges in } Branch\}$
17: $R_a \leftarrow R_a - \{\text{terminals in } Branch\}$
18: **return** T

Figure 5. The proposed M-SCTF/EDA algorithm.

Figure 5 shows the M-SCTF/EDA algorithm, which takes advantage of both the SCTF algorithm and the extended Dijkstra's algorithm to keep the source-to-receiver delay and the bandwidth consumption as low as possible for multiple-source multicast scenarios. The algorithm constructs a Steiner tree for each group of receivers. A group R_a is chosen for the Steiner tree construction if it has the "shortest" shortest path from the source to a receiver (ref.: line 3 in Figure 5) among all receiver groups in M, which initially includes all receiver groups. The algorithm then removes R_a from M and follows the similar idea of the SCTF algorithm to construct a Steiner tree rooted at s_a and spanning all receivers in R_a. (ref: lines 5-17 in Figure 5). The algorithm terminates and returns a set T of Steiner trees if it detects that M is empty.

Note that the shortest path from a source to a receiver is calculated by the extended Dijkstra's algorithm, which considers both the node weights and the edge weights for obtaining the shortest path that is more suitable for practical scenarios. The calling of EDA algorithm is abbreviated as EDA(x,y) in the M-SCTF/EDA algorithm with the meaning that the algorithm will directly return the shortest path from node x to node y.

Also note that the priority to order nodes in the priority queue is modified as: source > non-terminal node (less hop count from the source first) > terminal node. Due to the priority modification, the source-to-receiver delay may decrease, while the bandwidth consumption may not necessarily increase. This is helpful for the proposed M-SCTF/EDA algorithm to construct Steiner trees with both low source-to-receiver delay and low bandwidth consumption.

4. PERFORMANCE EVALUATION

The proposed M-SCTF/EDA algorithm is simulated by the EstiNet emulator along with a Ryu controller for two different multicast scenarios. The parameter setting of these two scenarios is shown in Table 1; it is derived according to the specifications of off-the-shelf products, namely NEC ProgrammableFlow PF5248 Switch, Xinguard Pica8 3290 Switch, and HP 3500 Series Switch. The two scenarios are depicted in Figure 6 and Figure 7, respectively, in which node 21 is the controller. The first scenario has only one source (node 40) and 18 receivers, while the second scenario has two sources (node 40 and node 23), which both have 8 receivers.

Other parameters of the simulation are described as follows. The multicast data are UDP packets sent at the constant bit rate of 2500 kbps, which is the rate of HD (high definition) video data of the 720P H.264 high profile format. The control knob κ for the priority queue is set as 4 for all simulation cases.

Table 1. Parameter setting of the scenario 1 and 2

Parameter	Setting
Bandwidth on edges	100Mbps ~ 1Gbps
Capacity of each node	10Gbps ~ 179Gbps
Number of sources	1 or 2
Number of receivers	18 or 16
Number of switches	20
Number of edges	63
Controller	Ryu ver 1.7.90
Simulation time per case	1000 sec

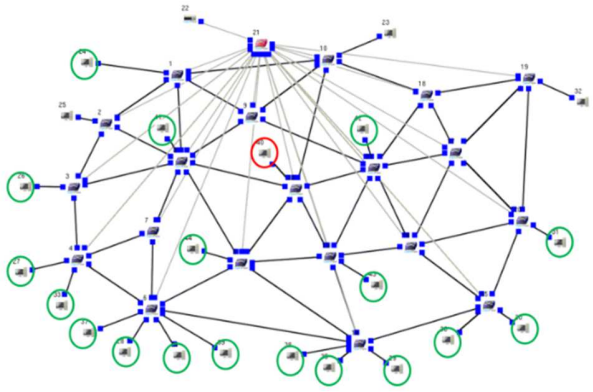


Figure 6. The simulation scenario 1.

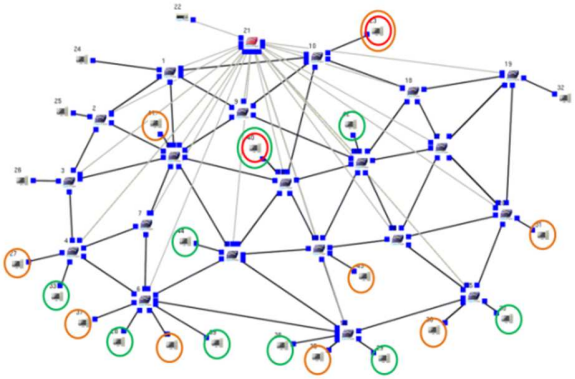


Figure 7. The simulation scenario 2.

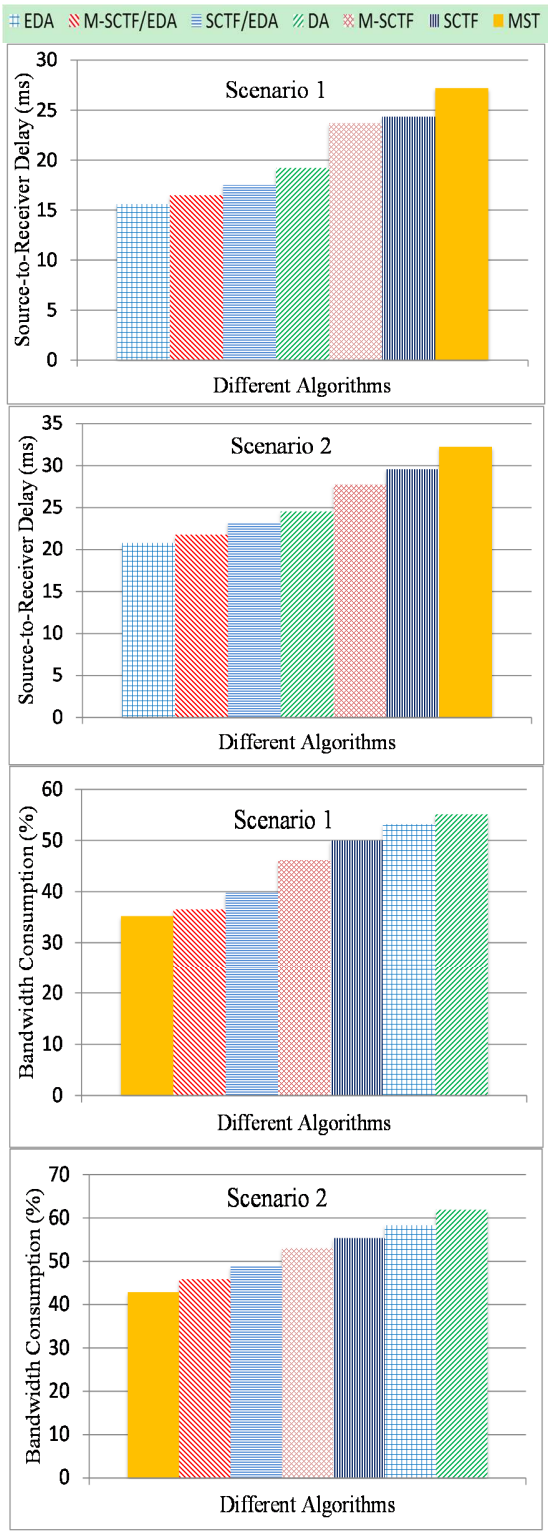


Figure 8. Performance comparisons for different algorithms.

The simulation experiments are performed for not only the proposed M-SCTF/EDA algorithm but also the SCTF/EDA, SCTF, M-SCTF, DA (i.e., the original Dijkstra's algorithm), EDA and the MST algorithm (i.e., the exhaustive algorithm to construct exactly the minimum Steiner tree using the minimum number of edges to span all receivers) for the sake of comparison. As shown in Figure 8, the proposed M-SCTF/EDA algorithm has very good performance. It has the second lowest source-to-receiver delay and bandwidth consumption. In terms of the source-to-receiver delay, it is inferior to only EDA, which focuses solely on the shortest delay. However, EDA has very high bandwidth consumption. In terms of the bandwidth consumption, it is inferior to only MST, which focuses solely on the lowest bandwidth consumption. However, MST is an exhaustive algorithm consuming a lot of computation and has very high source-to-receiver delay. We may well say that the proposed M-SCTF/EDA algorithm is the best algorithm suitable for the SDN architecture to achieve multicast with multiple sources when the source-to-receiver delay and the bandwidth consumption are both considered at the same time.

5. CONCLUDING REMARKS

The proposed M-SCTF/EDA algorithm takes advantage of both the extended Dijkstra's shortest path algorithm and the modified selective closest terminal first Steiner tree algorithm to construct a set of Steiner trees for multicast with multiple sources. The proposed algorithm reaches the goal of keeping as low as possible both the source-to-receiver delay and the bandwidth consumption. Simulation experiments through EstiNet emulator along with a Ryu controller are performed for the proposed algorithms and other related algorithms for the sake of comparison. The simulation results are compared in the aspects of the source-to-receiver delay and the bandwidth consumption. By the comparison results, we can observe the proposed M-SCTF/EDA algorithm is the only one among all compared algorithms to perform well in both aspects.

We have observed that the Floyd-Warshall algorithm [15] can also be easily extended to consider both edge weights and node weights. We have also observed that if we rewrite the two inner "for" loops in the proposed M-SCTF/EDA algorithm and replace the extended Dijkstra's algorithm (EDA) with the "extended Floyd-Warshall algorithm", then the proposed algorithm can be improved in terms of computation overheads. This is because the Dijkstra's algorithm is a one-to-all shortest path algorithm, the Floyd-Warshall algorithm is an all-pair shortest path algorithm, and the two inner "for" loops are actually used to find the "shortest" shortest path among all pairs of shortest paths. We plan to realize the improvement in the future.

Many studies related SDN multicast have been proposed recently. They focus on different aspects of multicast, such as fault-tolerance [16], scalability [17][18], and load balancing [19][20]. The proposed algorithm is not compared with those algorithms, as they emphasize on aspects different from source-to-receiver delay and bandwidth consumption. We plan to improve the proposed algorithm by considering more aspects and to apply the proposed algorithm to more practical multicast scenarios, like live streaming and the video conference.

6. ACKNOWLEDGMENTS

This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan, under grant number 104-2221-E-008-017- and grant number 105-2218-E-008-008-.

7. REFERENCES

[1] Open Network Foundation (ONF). https://www.opennetworking.org/sdn-resources/sdn-definition (last accessed on March 2016).

[2] Jain, S., et al. 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review*. 43, 4, 3-14.

[3] Nunes, B. A., Mendonca, M., Nguyen, X. N., Obraczka, K., and Turletti, T. 2014. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*. 16. 3. 1617-1634.

[4] Farhady, H., Lee, H., and Nakao, A. 2015. Software-defined networking: A survey. *Computer Networks*. 81. 79-95.

[5] Fenner, B., et al. 2006. Protocol independent multicast - sparse mode (pim-sm): protocol specification (revised). *IETF RFC 4601*.

[6] Hwang, F. K., Richards, D. S., and Winter, P. 1992. *The Steiner tree problem*. Elsevier.

[7] Jiang, J. R., Huang, H. W., Liao, J. H. and Chen, S. Y. 2014. Extending Dijkstra's shortest path algorithm for software defined networking. In *Proc. of 16th IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 1-4.

[8] Ramanathan, S. 1996. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Transactions on Networking (TON)*. 1996

[9] Wang, S. Y., Chou, C. L., and Yang, C. M. 2013. EstiNet OpenFlow network simulator and emulator. *IEEE Communications Magazine*. 51. 9. 110-117.

[10] Ryu OpenFlow Controller. URL: http://osrg.github.io/ryu/

[11] Banse, C, and Rangarajan, S. 2015. A secure northbound interface for SDN applications. In *Proceedings of the* 2015 *IEEE Conference on* Trustcom/BigDataSE/ISPA.

[12] Open Networking Foundation. 2015. *OpenFlow Switch Specification version 1.5.1*.

[13] Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, 1. 1. 269-271.

[14] Jiang, J.-R., et al. 2014. Load balancing and multicasting using the extended Dijkstra's algorithm in software defined networking. In *Proc. of the International Computer Symposium 2014 (ICS 2014)*.

[15] Floyd, R. W. 1962. Algorithm 97: Shortest path. Communications of the ACM. 5. 6. 345.

[16] Pfeiffenberger, T., et al. 2015. Reliable and flexible communications for power systems: Fault-tolerant multicast with SDN/OpenFlow. In *Proc. of the 7th IEEE International Conference on New Technologies, Mobility and Security (NTMS)*. 1-6.

[17] Cui, W., and Qian, C. 2015. Scalable and load-balanced data center multicast. In *Proc. of 2015 IEEE Global Communications Conference (GLOBECOM)*. 1-6.

[18] Zhou, S., Wang, H., Yi, S., and Zhu, F. 2015. Cost-efficient and scalable multicast tree in software defined networking. *Algorithms and Architectures for Parallel Processing*. 592-605.

[19] Iyer, A., Kumar, P., and Mann, V. 2014. Avalanche: Data center multicast using software defined networking. In *Proc. of the 6th IEEE International Conference on Communication Systems and Networks (COMSNETS)*. 1-8.

[20] Craig, A., Nandy, B., Lambadaris, I., and Ashwood-Smith, P. 2015. Load balancing for multicast traffic in SDN using real-time link cost modification. In *Proc. of IEEE International Conference on Communications (ICC)*. 5789-5795.