# Efficient Unicast Routing Algorithms in Software-Defined Networking

Jang-Ping Sheu, Quan-Xiang Zeng, Jagadeesha RB, and Yeh-Cheng Chang
Department of Computer Science, National Tsing Hua University
Hsinchu, 30013, Taiwan
sheujp@cs.nthu.edu.tw, s101062564@m101.nthu.edu.tw, jagadeesha.rb@gmail.com, jas1123kimo@gmail.com

***Abstract*- The recently developed Software-Defined Networking (SDN) provides flexibility, scalability and reduce the cost of hardware for operators and vendors. In this paper, we propose four routing algorithms in SDN to minimize network latency and balance the traffic load of each switch link in a network. The first routing algorithm is the shortest path first (SPF) algorithm. The SPF algorithm can find the shortest path with maximum bottleneck bandwidth (MBB) among all shortest paths. The second algorithm is the bandwidth aware routing (BAR) algorithm. The BAR algorithm can find a path with MBB on a network. The third algorithm is the *k*-SPF, which can find a path with MBB among the first *k* shortest paths. Finally, our fourth algorithm *k*-BAR can find the shortest path among the first *k* MBB paths on a network. Simulation results show that the performance of our algorithms is better than previous works.**

***Keywords- Load balance; Routing algorithms; Software-defined networking***

## I. INTRODUCTION

Due to the closed nature of today's computer network infrastructure, it is extremely hard to make the change, which is a huge obstacle to the innovation. Thus, an open, standard and programmable platform, Software-Defined Networking (SDN) [5, 10] is developed enabling researchers to run their new ideas in production networks. SDN is a new network architecture that separates the network control plane from the network data plane. The SDN controller determines the forwarding path for each flow in the network. The data plane is located in the network equipment as before. OpenFlow [1, 6] is a standardized interface that can be used by the controller to communicate with the data plane. SDN provides researchers the flexibility to program the network and run their new ideas practically. Mainly, in SDN when compared to a traditional network we could have a global view of the network.

The authors in [7] show that minimizing network latency and maximizing throughput are two important issues in data center networks. As a result, we want to propose routing algorithms that achieve the goal of not only reducing the network latency but also balancing the traffic load of each switch link to increase the network throughput. A path is known as the shortest path if the length of this path is smallest among all paths between a source and destination pair. In [4], the authors propose an online learning shortest path routing algorithm to improve the network performance. The bottleneck bandwidth of a path is the smallest link bandwidth of the path. The authors in [2] use the integer linear programming approach to formulating the widest pair of disjoint paths problem and reduces the original integer linear programming problem, which is an NP-complete to a linear programming problem to lower the difficulty of solving the problem. In order to achieve low network latency, the authors in [3] propose a load balancer for OpenFlow based data center networks, and implement a dynamic routing algorithm in the load balancer to distribute traffic load across multiple paths. A few schemes based on multicast are defined in [11]-[13].

In this paper, we propose four unicast routing algorithms to achieve the goal of reducing the latency and balancing the traffic load of the network. The first algorithm, named shortest path first (SPF) algorithm, will focus on finding the shortest path from source to destination. In order to utilize the network bandwidth, if more than one shortest path exists, we will select the path, which has the maximum bottleneck bandwidth (MBB). SPF algorithm is used to route the latency sensitive applications. The second algorithm, named bandwidth-aware routing (BAR) algorithm focuses on finding a path with MBB from source to destination. If more than one path with the same MBB exists, we will select the one which has the shortest path. BAR algorithm is used to balance the workload of the switch links. The third algorithm is the *k*-SPF, which can find a path with MBB among the first *k* shortest paths. The fourth algorithm is *k*-BAR which finds the shortest path among the first *k* MBB paths on a network. The time complexity of our SPF and BAR algorithm is $O(|E| + |V|\log|V|)$ and the time complexity of our *k*-SPF and *k*-BAR algorithm is $O(|V|\log|V| + k|E|)$, where $|V|$ and $|E|$ are the numbers of switches and links of a network. The simulation results show that our four routing algorithms can lower the unsatisfied request rate and raise the bandwidth satisfaction rate, and the network link utilization.

The rest of the paper is organized as follows. We present preliminary in section II. In section III, we describe our algorithm in detail. Section IV presents the performance of our algorithms. Conclusion is given in section V.

## II. PRELIMINARIES

In this section, we describe the SDN architecture of our system and the problem in detail. An SDN architecture consists of four components: links, SDN-enabled switches, SDN controller, and hosts. We assume that an SDN consists of a number of switches interconnected by a set of links and have multiple links between the switches. An SDN network topology can be represented as a weighted, directed graph $G = (V, E)$ in which $V$ is a set of vertices and $E$ is a set of edges interconnected

vertices in $V$. Each vertex in $V$ represents a switch in SDN and each edge in $E$ represents a switch link in SDN. Since multiple links between a switch pair is allowed, we use the notation $e(u_i, v_i)$ to indicate the $i^{th}$ edge from vertex $u$ to vertex $v$ and use the notation $S_{u,v}$ to indicate the set of all edges from vertex $u$ to vertex $v$, where $u, v \in V$.

For each edge $e \in E$, $w(e)$ denotes the link weight and $b(e)$ denotes the remaining bandwidth of the switch link. Let $s \in V$ be a vertex called source and $d \in V$ be a vertex called destination. Assume $p$ is a path from $s$ to $d$, the cost of the path $p$, denoted by $d(p)$ is equal to the sum of the weights on every link of the path. The communication capacity of the path $p$, denoted by $c(p)$ is equal to the bottleneck bandwidth of the path. We note that many of the flow tables of the switches are implemented using ternary content addressable memory (TCAM), which is extremely expensive, power consuming and therefore of limited size [14][15]. Thus, if there is more than one path with the same cost, we will select the path whose switches have the maximum bottleneck flow entries. For each $v \in V$, $f(v)$ denotes the available flow entries of the switch $v$. Let $\sigma(p)$ denote the bottleneck flow entries of a path, which is equal to the minimum flow entries of the switches that make up the path.

In the following, we define two kinds of routing strategy which are used in this paper. The first strategy is to find the shortest path with maximum bottleneck bandwidth (MBB) among all shortest paths from $s$ to $d$. The bottleneck bandwidth of a path is defined as the minimum bandwidth of all links of the path. The second problem is to find a shortest path with MBB in a network.

**1) Shortest path first:** Find a shortest path from $s$ to $d$. In order to utilize the network bandwidth, if more than one shortest path exists, we will select the path with MBB among all shortest paths. As we mentioned above, TCAM is a scarce resource. Therefore, if there exist multiple paths with the same bottleneck bandwidth, the path which has the maximum bottleneck flow entries will be selected.

For example, in Fig. 2.1, the alphabet in the circle is the switch's ID and the number is the available flow entries of the switch. The first value on a directed edge is the weight of the edge and the second value is the available bandwidth of the edge. In Fig. 2.1, we can find three shortest paths from $s$ to $d$ with the same cost $d(p) = 21$: *s - a - e - d*, *s - b - c - e - d*, and *s - b - f - d*. The path with MBB among these candidate paths will be selected. Since the two paths: *s - b - c - e - d* and *s - b - f - d* have the same bottleneck bandwidth $c(p) = 8$, the path with maximum $\sigma(p)$ will be selected. So, the path *s - b - f - d* with $d(p) = 21$, $c(p) = 8$, and $\sigma(p) = 6$ is the final result.

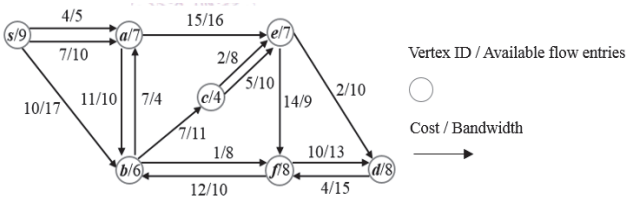



Figure 2.1: An example of SDN topology

**2) Bandwidth aware routing:** Find a path with MBB in network from $s$ to $d$. If more than one path with the same MBB exists, we will select the path which has a shortest path. If there exist more than one shortest path, the path, which has the maximum bottleneck flow entries, will be selected. In Fig. 2.1, there are three paths: *s - a - e - d, s - a - b - c - e - d*, and *s - b - c - e - d* have the same $c(p) = 10$ from $s$ to $d$. And the two paths: *s - a - e - d* and *s - b - c - e - d* have the same minimum cost $d(p) = 24$. Since the path *s - a - e - d* has larger $\sigma(p)$ than the path *s - b - c - e - d*, the path *s - a - e - d* with $c(p) = 10$, $d(p) = 24$ and $\sigma(p) = 7$ is selected as the final solution.

## III. ROUTING ALGORITHMS

In this section, we present our algorithms in detail.

### A. Shortest Path First (SPF) Algorithm

Our first algorithm is to find the shortest path from source to the destination whose bottleneck bandwidth is maximum among all shortest paths. If there exists more than one shortest path with the same bottleneck bandwidth, we will select the path, which has the maximum bottleneck flow entries.

In this paper, we modify the *relax* operation of Dijkstra's shortest path algorithm [8] to find the shortest path. First, we define some notations which are used in the SPF algorithms. For each vertex $v \in V$, $d(v)$ denotes the cost of the path from source $s$ to $v$, $c(v)$ denotes the capacity of the path from source $s$ to $v$, $\sigma(v)$ denotes the available flow entries of the path from source $s$ to $v$, and $p(v)$ denotes the parent vertex of $v$. When a vertex $v \in V$ can be reached from source with a finite cost, we can *relax* all of the outgoing edges $e(v_i, w_i) \in S_{u,v}$ as follows.

- If $d(w) > d(v) + w(e(v_i, w_i))$, the value of $d(w)$ is updated as $d(v) + w(e(v_i, w_i))$, $c(w)$ is changed to $\min\{c(v), b(e(v_i, w_i))\}$, $\sigma(w)$ is updated as $\min\{\sigma(v), f(w)\}$ and $p(v)$ is set as $v$.
- If $d(w) = d(v) + w(e(v_i, w_i))$ and $c(w) < \min\{c(v), b(e(v_i, w_i))\}$, $c(w)$ is changed to $\min\{c(v), b(e(v_i, w_i))\}$, $\sigma(w)$ is updated as $\min\{\sigma(v), f(w)\}$, and $p(v)$ is set as $v$.
- If $d(w) = d(v) + w(e(v_i, w_i))$, $c(w) = \min\{c(v), b(e(v_i, w_i))\}$ and $\sigma(w) < \min\{\sigma(v), f(w)\}$, $\sigma(w)$ is updated as $\min\{\sigma(v), f(w)\}$ and $p(v)$ is set as $v$.

Initially, all of the vertices are set as unvisited. For source vertex $s$, the initial values of $d(s)$, $c(s)$, $\sigma(s)$, and $p(s)$ is set as 0, $\infty$, $\infty$, and *, respectively. For each non-source vertex $v \in V$, the initial values of $d(v)$, $c(v)$, $\sigma(v)$, and $p(v)$ are set as $\infty$, 0, 0, and *, respectively. For example, Fig. 3.1(a) shows the initial state of a directed graph $G$. We will visit an unvisited vertex with the smallest cost and *relax* its neighbor vertices. Thus, the source vertex $s$ will *relax* its neighbor vertices and the result is shown in Fig. 3.1(b). After doing $|V|$ *relax* operations, the final result is shown in Fig. 3.1(c). We can trace back the path from vertex $d$ to $s$ and get the final path *s - b - f - d* with $d(p) = 21$, $c(p) = 8$, and $\sigma(p) = 6$. The time complexity of our SPF algorithm is the same as the Dijkstra's shortest path algorithm, which is equal to $O(|E| + |V|\log|V|)$.

### B. Bandwidth Aware Routing (BAR) Algorithm

In this subsection, we focus on finding a path with MBB of a network by modifying the SPF algorithm. We first modify the *relax* operation of SPF algorithm. For each vertex $v \in V$, the BAR algorithm selects a shortest path from source to $v$ which has the MBB in a network instead of selecting the shortest one in SPF

algorithm. When a vertex $v \in V$ can be reached from source $s$ with a non-zero residual bandwidth, we *relax* all of the outgoing edges $e(v_i, w_i) \in S_{u,v}$. If $c(w) < \min\{c(v), b(e(v_i, w_i))\}$, the value of $c(w)$ will be updated as $\min\{c(v), b(e(v_i, w_i))\}$. After doing $|V|$ times of *relax* operation, we can get the value of MBB from $s$ to $d$. In case of several paths with the same MBB, we select the one with the shortest path among them and delete all edges of smaller MBB in the original graph. Then execute the SPF algorithm on the reduced graph to find shortest path with the MBB from $s$ to $d$.

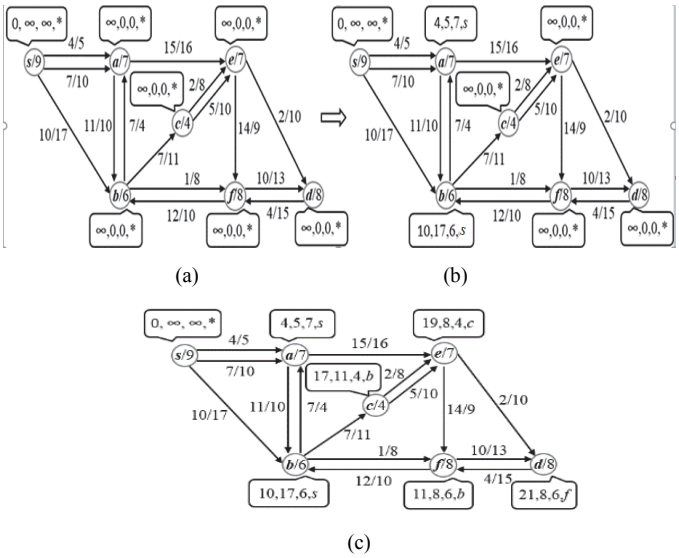



Figure 3.1: An example of the SPF algorithm

For example, Fig. 3.2(a) shows the initial state of a graph $G$. Initially, all of the vertices are set as unvisited and the values of $d(v)$, $c(v)$, $\sigma(v)$, and $p(v)$ are shown in Fig. 3.2(a). In each relax operation, we will visit the unvisited vertex with the largest capacity $c(v)$ and *relax* its neighbor vertices. In the first round, vertex $s$ will *relax* its neighbor vertices and the result is shown in Fig. 3.2(b). After doing $|V|$ *relax* operations, we have the value of MBB from $s$ to $d$ which is equal to 10. Then, we delete the edges of the original graph whose remaining bandwidth is smaller than 10 and get a new graph as shown in Fig. 3.2(c). In order to lower the communication latency and fully utilize the TCAM resource of the switch, we apply the SPF algorithm to Fig. 3.2(c). After the SPF algorithm is done, we get the path $s$ - $a$ - $e$ - $d$ with $c(p) = 10$, $d(p) = 24$ and $\sigma(p) = 7$ as shown in Fig. 3.2(d). The time complexity of running $|V|$ *relax* operations is $O(|E| + |V|\log|V|)$. The time cost to delete the edges whose bandwidth is smaller than MBB consumes $O(|E|)$. The time of applying SPF on the reduced graph is $O(|E| + |V|\log|V|)$. Thus, the performance of the BAR algorithm is $O(|E| + |V|\log|V| + |E| + |E| + |V|\log|V|)$ which is equal to $O(|E| + |V|\log|V|)$.

### C. *k*-SPF Algorithm

We propose a *k*-SPF algorithm to find a path with MBB among the first $k$ shortest paths, where $k \geq 2$ is a predefined number. Our *k*-SPF algorithm is refined from the algorithm proposed in [9]. We first reverse the direction of every edge in the original network graph. Then, we use our SPF algorithm to find one-to-all shortest paths starting at vertex $d$ and construct a shortest path tree rooted at $d$. After finding the shortest path tree from $d$ to all vertices in a network graph, we reverse the direction of every edge in the shortest path tree. In this way, we can find the shortest paths from all vertices to vertex $d$ and obtain a shortest path tree $T$ from all vertices to $d$. Let $T_i$ be the shortest path from vertex $i$ to vertex $d$ and $d(T_i)$ denote the cost of $T_i$. Let $A^i = v_1^i - v_2^i - v_3^i - \ldots - v_{\beta_i}^i - d$ be the $i^{\text{th}}$ shortest path from source $s$ to destination $d$, where $v_1^i = s$ and $v_2^i, v_3^i, \ldots, v_{\beta_i}^i$ are the $2^{\text{nd}}, 3^{\text{rd}}, \ldots, \beta_i^{\text{th}}$ vertex of the $i^{\text{th}}$ shortest path, where $1 \leq i \leq k$ and $(\beta_i + 1)$ represents the number of vertices of $A^i$. We can determine the $i^{\text{th}}$ shortest path $A^i$ from $s$ to $d$ after we have $A^{i-1}$.

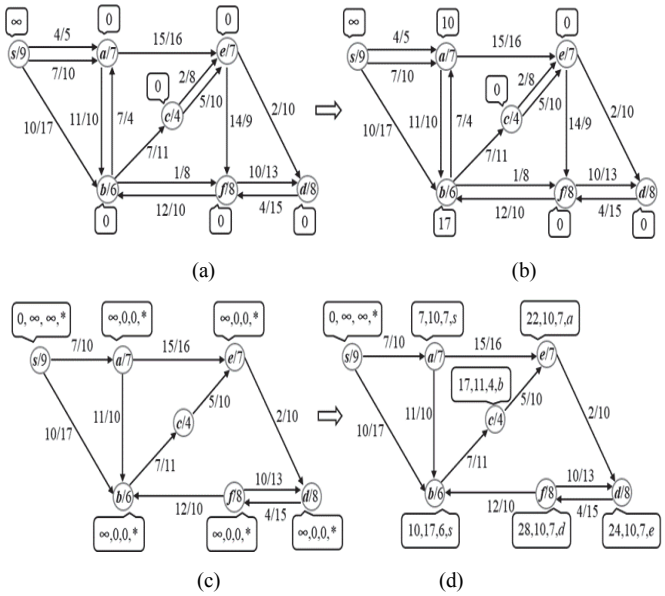
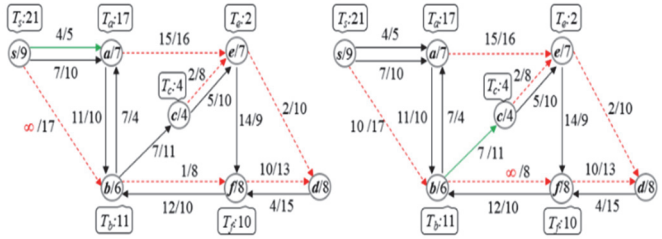



Figure 3.2: An illustration of the BAR algorithm, steps (a)-(d).

In [9], we know that $A^i$ is a deviation from $A^m$, for $1 \leq m \leq i$-1. Thus, we need to find the $A^1$ first and one can determine $A^1$ easily from $T$ as the first shortest path from $s$ to $d$ is $T_s$. In order to determine $A^i$, we need to find $\beta_{i-1}$ branch paths, which are, deviated from the vertices of $A^{i-1}$ except vertex $d$, for $2 \leq i \leq k$. The $\beta_{i-1}$ shortest branch paths can be obtained by iterating the following procedure $\beta_{i-1}$ times. In $j^{\text{th}}$ iteration, let $A_j^i$ be a shortest branch path from $s$ to $d$ which is deviated at $v_j^{i-1}$ of $A^{i-1}$ and let the sub-path of $A^{i-1}$ from $v_1^{i-1}$ to $v_j^{i-1}$ be the root path of $A_j^i$ called $R_j^i$, for $1 \leq j \leq \beta_{i-1}$. Let $n_1^j, n_2^j, \ldots, n_{\pi_j}^j$ be the $1^{\text{st}}, 2^{\text{nd}}, \ldots, \pi_j^{\text{th}}$ neighbor vertex of $v_j^{i-1}$, where $\pi_j$ denotes the number of neighbor vertices of $v_j^{i-1}$. We compute the shortest path cost from $v_j^{i-1}$ to $d$ through $n_p^j$ which is equal to $w(e(v_j^{i-1}, n_p^j)) + d(Tn_p^j)$, for $1 \leq p \leq \pi_j$. After obtaining all of the shortest path costs from $v_j^{i-1}$ to $d$ through the neighbor vertices of $v_j^{i-1}$, we can find a shortest path from $v_j^{i-1}$ to $d$ and this path is the spur path of $A_j^i$ called $S_j^i$. Note that, in order to avoid computing the same spur path, which has the same root path of the first $i$-1 shortest paths. We set $w(e(v_j^m, v_{j+1}^m))$ as infinity if $R_j^i$ equals to the sub-path of the first $j$ vertices of $A^m$ and the next edge of vertex $v_j^m$ $e(v_j^m, v_{j+1}^m)$ is used in $A^m$, for $1 \leq m \leq i$-1. Then $A_j^i$ can be obtained by combining $R_j^i$ and $S_j^i$ and $A_j^i$ is added to a set $P$. After the $\beta_{i-1}^{\text{th}}$ iteration is done, we can find all shortest branch paths which are deviated from the vertices of $A^{i-1}$. Then $A^i$ will be the path with minimum cost in the set $P$. Repeat the above procedure $k$-1 times to find the first $k$ shortest paths. Finally, we select a path with maximum $c(p)$ among the first $k$ shortest paths to be the route between $s$ and $d$.

For example, the red dotted lines as shown in Fig. 3.3(a) make up the shortest path tree $T$ from all vertices to $d$. We can determine $A^1$ from $T$ which is $T_s$: $s$ - $b$ - $f$ - $d$. In order to determine $A^2$, we need to find $A_1^2$, $A_2^2$ and $A_3^2$. Since the sub-path of $A^{2\text{-}1}$ from $v_1^{2\text{-}1}$ to $v_1^{2\text{-}1}$ is $s$, $R_1^2 = s$. Because $R_1^2$ equals to the sub-path of the first vertex of $A^1$ $s$ and the 2nd vertex of $A^1$ is $v_2^1 = b$, $w(e(s, b))$ will be set infinity as shown in Fig. 3.3(a).



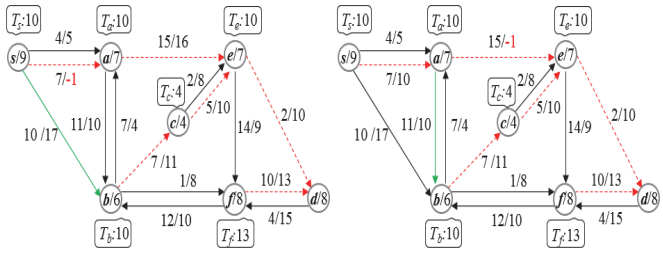
(a) (b)

Figure 3.3: An example of finding $A_1^2$ and $A_2^2$

In Fig. 3.3(a), the shortest path from $s$ to $d$ is $s$ - $a$ - $e$ - $d$ because this path has the minimum cost = $w(e(s, a)) + d(T_a) = 4 + 17 = 21$. Therefore, $S_1^2 = s$ - $T_a$ which is the path: $s$ - $a$ - $e$ - $d$. After $R_1^2$ and $S_1^2$ are determined, we can get $A_1^2$ by joining $R_1^2$ and $S_1^2$ which is the path: $s$ - $a$ - $e$ - $d$ with cost = 21 and $A_1^2$ will be added to the candidate path set $P$. Here, we find $A_2^2$ as follows. Since the sub-path of $A^{2\text{-}1}$ from $v_1^{2\text{-}1}$ to $v_2^{2\text{-}1}$ is $s$ - $b$, $R_1^2 = s$ - $b$. Because $R_2^2$ equals to the sub-path of the first two vertices of $A^1$ and the 3rd vertex of $A^1$ is $v_3^1 = f$, $w(e(b, f))$ will be set infinity as shown in Fig. 3.3(b). Therefore, the shortest path from $b$ to $d$ is $b$ - $c$ - $e$ - $d$ which has the minimum cost = $w(e(b, c)) + d(T_c) = 7 + 4 = 11$. As a result, $S_2^2 = b$ - $T_c = b$ - $c$ - $e$ - $d$. After joining $R_2^2$ and $S_2^2$, $A_2^2 = s$ - $b$ - $c$ - $e$ - $d$ with cost = 21 which will be added to $P$. Since there is no outgoing edge from vertex $f$ after $w(e(f, d))$ is set as infinity, $A_3^2 = null$. Now, there are two paths $A_1^2$ and $A_2^2$ in $P$ and $A^2$ will be the path with minimum cost in $P$. However, the paths $s$ - $a$ - $e$ - $d$ and $s$ - $b$ - $c$ - $e$ - $d$ have the same cost = 21, so the path: $s$ - $b$ - $c$ - $e$ - $d$ with larger available bandwidth than the path $s$ - $a$ - $e$ - $d$ will be selected as $A^2$.

The time to construct the shortest path tree from all vertices to vertex $d$ is $O(|E| + |V|\log|V|)$. To obtain the $i^{\text{th}}$ shortest path, the $k$-SPF algorithm makes $\beta_{i-1}$ iterations to compute all of the shortest branch paths deviating from $A^{i-1}$, where $\beta_{i-1} + 1$ is the length of $A^{i-1}$. In worst case, the value of $\beta_{i-1}$ is $(|V| - 1)$ and the edges that we need to scan in computing the spur paths: $S_1^i$, $S_2^i$,…, and $S_{\beta i-1}{}^i$ will not exceed $|E|$. As a result, the time to obtain the $i^{\text{th}}$ shortest path is $(|V| + |E|)$ and the time to obtain the first $k$ shortest paths is $k(|V| + |E|)$. The total running time of our $k$-SPF algorithm is $O(|E| + |V|\log|V| + k(|V| + |E|)) = O(|V|\log|V| + k|E|)$ which is better than the Yen's algorithm [9] whose time complexity is $O(k|V|(|E| + |V|\log|V|))$.

The BAR algorithm can find an optimal solution based on the criteria of MBB. The $k$-BAR algorithm is used to find a shortest path among the first $k$ largest bottleneck bandwidth paths in a network, where $k \geq 2$ is a predefined number. The $k$-BAR algorithm can be obtained by modifying the $k$-SPF one. For example, Fig. 3.2(a) shows the initial state of a graph $G$. Then the red dotted lines as shown in Fig. 3.4 make up the shortest path tree $T$ from all vertices to $d$ under the MBB constraint. Assume $B^i$ is a path with $i^{\text{th}}$ largest bottleneck bandwidth. We can determine $B^1$ from $T$ which is $T_s$: $s$ - $a$ - $e$ - $d$ with $d(p) = 24$, $c(p) = 10$ as shown in Fig. 3.4(a). To determine $B^2$, we need to find $B_1^2$, $B_2^2$ and $B_3^2$. The procedure of finding these paths is similar with the $k$-SPF algorithm. Then, we can find $B_1^2 = s$ - $b$ - $c$ - $e$ - $d$, $B_2^2 = s$ - $a$ - $b$ - $c$ - $e$ - $d$ as shown in Fig. 3.4(a), Fig. 3.4(b) respectively and similarly can find $B_3^2 = s$ - $a$ - $e$ - $f$ - $d$ . Note that, similar to the $k$-SPF algorithm, in order to avoid computing the same spur path which has the same root path of the first $k$-1 largest bottleneck bandwidth paths, we set $b(e(v_j^m, v_{j+1}^m)) = -1$ if $R_j^i$ equals to the sub-path of the first $j$ vertices of $B^m$ and the next edge of vertex $v_j^m$ $e(v_j^m, v_{j+1}^m)$ is used in $B^m$, for $1 \leq m \leq i$-1. Finally, we have the second largest bottleneck bandwidth path $B^2 = s$ - $b$ - $c$ - $e$ - $d$ with $d(p) = 24$, $c(p) = 10$ from $B_1^2$, $B_2^2$ and $B_3^2$.



(a) (b)

Figure 3.4: An example of finding $B_1^2$ and $B_2^2$

## IV. Performance Evaluation

In this section, we evaluate the performance of our proposed algorithm through simulations. We simulate an SDN environment by JAVA programming language. In our simulations, we create a network with 100 switches and 400 hosts. A controller controls all the switches. The number of links between switches is 850 that are assigned to each switch randomly. All links have the same initial bandwidth: 1GB/s. For the fair of comparison, we do not consider the TCAM size in the following simulations. When a host generates a communication request, the first packet of the request is sent to the controller to determine the communication path.

### *4.1 Traffic Pattern & Measurement*

(a) Measurement: Each host sends packets with each flow assigned according to the Gaussian probability and mean flow size ranges from 100MB/s to 900MB/s [4]. We then measure the performance according to the following criteria. (1) Average hop count or communication cost of a request: The lower average communication cost implies the lower packet transfer delay. (2) Average unsatisfied request rate: The average unsatisfied request rate is the ratio of unsatisfied requests over total requests. The unsatisfied request is a request that can only get partial bandwidth from a routing path which has no enough bandwidth to support this request. The higher unsatisfied request rate will result in lower link utilization. (3) Average bandwidth satisfaction rate: The average bandwidth satisfaction rate is the real assigned bandwidth over the desired (request) bandwidth. As a result, higher the bandwidth satisfaction rate is, the higher throughput of the network is. (4) Average link utilization: The total consuming bandwidth of each link over the total available

bandwidth of the switch links. The low link utilization means that the network resource (bandwidth) is wasted.

(b) Benchmark algorithms: We compare the experimental results of our proposed algorithms - SPF, *k*-SPF, BAR, *k*-BAR with the Open Shortest Path First (OSPF) routing protocol. The *k*-SPF algorithm and the *k*-BAR algorithm are simulated with *k* = 5, and 6 as we experimented them and found suitable at those values.

*4.2 Simulation Results*

Fig. 4.1 shows the average hop count of OSPF, SPF, 5-SPF, 6-SPF, BAR, 5-BAR, and 6-BAR in a network with 850 switch links. The OSPF and our SPF algorithm have the lowest hop count compared to all schemes because of shortest path strategy. The *k*-SPF will find a path with MBB among the first *k* shortest paths. Therefore, the average hop count of the 6-SPF algorithm is lightly higher than the 5-SPF algorithm. Since the BAR algorithm finds the path with the MBB in the network, it has the highest hop count among all schemes. The hop count of 6-BAR is better than 5-BAR but higher than the hop count of 5-SPF and 6-SPF.

Figure 4.2 shows the average unsatisfied request rate of a network with 850 switch links. In general, the average unsatisfied request rate increases when the average flow size of the requests increases. Since the OSPF algorithm finds a shortest path without considering the factor of link utilization, it may select a path with small residual bandwidth and lead to high unsatisfied request rate. In case of multiple shortest paths, our SPF algorithm will select the one with maximum bottleneck bandwidth. As a result, the unsatisfied request rate of our SPF algorithm is lower than that of the OSPF algorithm. The 5-SPF

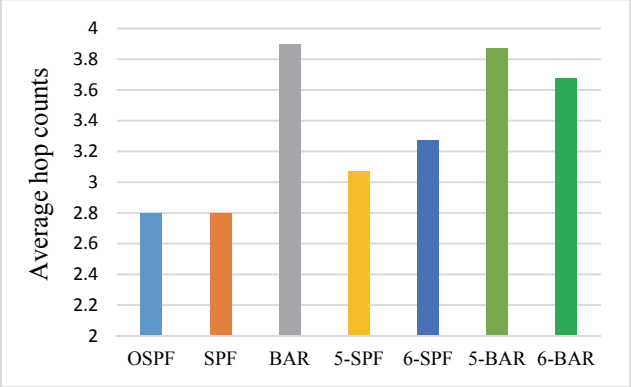



Figure 4.1: Average hop counts of a network with 850 switch links

and 6-SPF algorithms will pick a path with MBB among the first 5 and 6 shortest paths, respectively. Therefore, the unsatisfied request rate of 6-SPF algorithm is lower than that of 5-SPF one. Since the BAR algorithm always finds a path with MBB on a network, it has the lowest unsatisfied request rate among all schemes. Similarly, the unsatisfied request rate of 5-BAR is better than 6-BAR. When the average flow size is larger than 600 MB, each link can satisfy only one request in most of the time. Therefore, the average unsatisfied request rate will not change after the average flow size is over 600 MB. From Fig. 4.1 and Fig. 4.2, we can observe that the trend of the *k*-SPF and *k*-BAR algorithms is similar. If we increase the value of the variable *k* for both the algorithms, we can find that the simulation results of these two algorithms will be much closer to each other gradually. Therefore, we only evaluate the performance of *k*-SPF algorithm in the subsequent simulations.

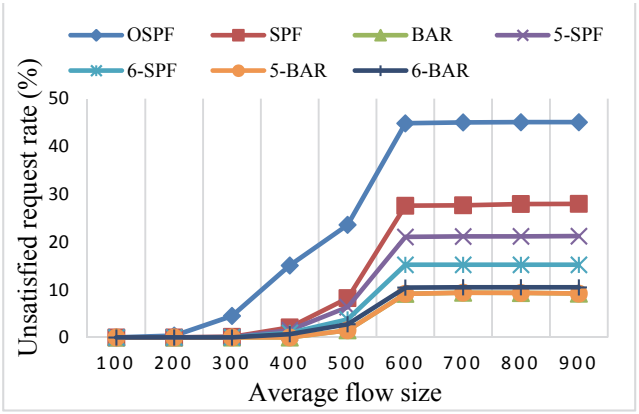



Figure 4.2: Average unsatisfied request rate of a network with 850 switch links

As we mentioned above, the bandwidth assigned to a request may be less than the desired bandwidth request. Generally, the service providers want to satisfy the user's requests as much as possible. In Fig. 4.3, we show the average bandwidth satisfaction rates of various routing schemes. Because the BAR algorithm always finds a path with MBB of a network, it has the highest bandwidth satisfaction rate among all schemes. The bandwidth satisfaction rate of the 6-SPF and 5-SPF algorithms are lower than that of the BAR because the *k*-SPF algorithm only considers the first *k* shortest path with MBB instead of choosing a path with MBB from a network. The SPF has better bandwidth satisfaction rate than the OSPF because the SPF chooses the path with MBB among all the shortest paths.

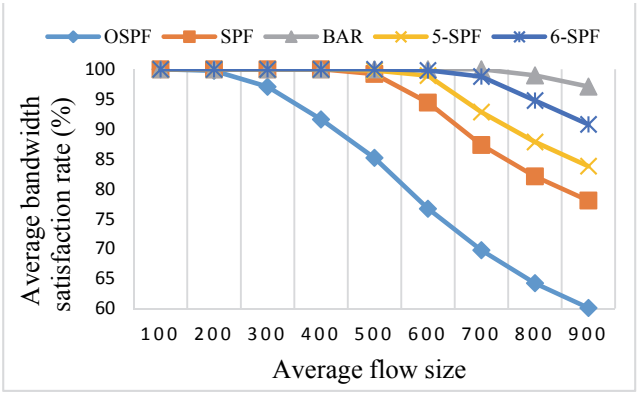



Figure 4.3: Average bandwidth satisfaction rate of a network with 850 switch links

Figure 4.4 shows the average link utilization of a network with 850 links for various routing protocols. In general, the average link utilization increases when the flow size of the requests increases. If a request cannot be satisfied, the consuming bandwidth of a request will be less than the requested bandwidth. As a result, the lower bandwidth satisfaction rate implies the lower link utilization. Therefore, the average link utilization OSPF is the lowest and BAR algorithm is the highest.

In the previous simulations, we assume that the cost of each link is equal to 1. Here, we assign the cost of each link vary from

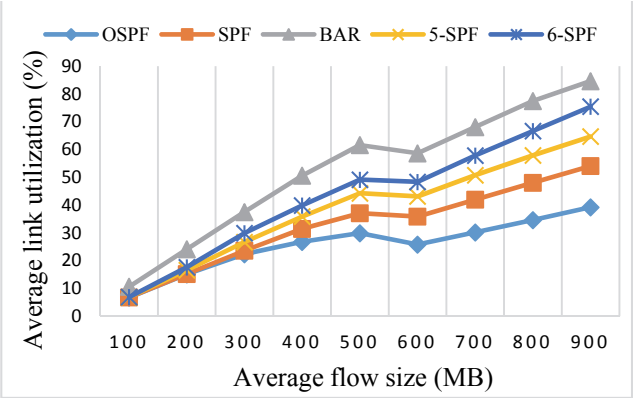



Figure 4.4: Average link utilization of a network with 850 switch links

1 to 10 randomly and evaluate the average unsatisfied rate of OSPF, SPF, 5-SPF, 6-SPF, and BAR. Figure 4.5 shows the unsatisfied request rate of the OSPF, SPF, 5-SPF, 6-SPF, and BAR schemes. Due to the variance of each link cost in this experiment is larger than the previous simulation, the number of shortest paths between a source and a destination is smaller than that of the previous experiments. Thus, the unsatisfied request rate of our SPF algorithm will be larger than the previous experiment. However, the unsatisfied request rates of 5-SPF and 6-SPF algorithms are increasing a little compared to Fig. 4.2. This shows the benefit of the *k*-SPF algorithm. The unsatisfied request rates of OSPF algorithm increases a little compared to the previous simulation with link cost = 1. This is because the OSPF algorithm can only find a shortest path to route from source to destination in both scenarios. Since the BAR algorithm always finds a path with MBB on a network without considering the communication cost, the unsatisfied request rate of the BAR algorithm is the same as the previous experiment.

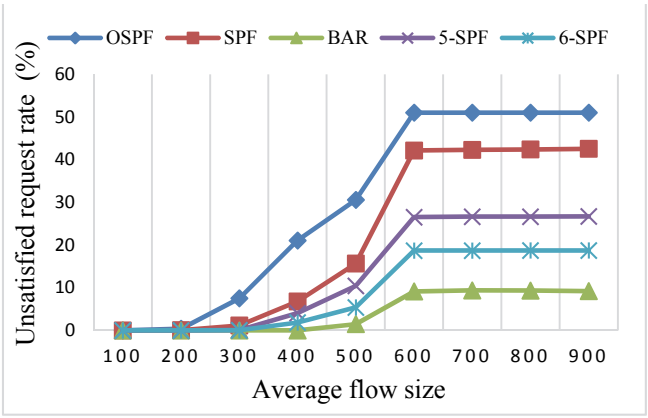



Figure 4.5: Average unsatisfied request rate of a network with each link cost from 1 to 10

## V. CONCLUSION

In this paper, we have presented four efficient routing algorithms in software-defined networks. The SPF algorithm selects the path with MBB if more than one shortest path exists. The BAR algorithm first finds the MBB of a network from *source* to *destination*. If more than one path with the same MBB exists, we select the path with the shortest cost. The *k*-SPF finds the path with MBB from the first *k* shortest paths. The *k*-BAR selects the shortest path from the first *k* MBB paths of a network. In our simulations, SPF algorithm has the lower bandwidth satisfaction rate, lower network link utilization but it has the smallest cost among all schemes. The BAR algorithm has the largest communication cost however, it has the highest bandwidth satisfaction rate and the highest network link utilization. This implies that our BAR algorithm can fully utilize the bandwidth of the network to accept requests as much as possible. Our *k*-SPF and *k*-BAR algorithm has the middle bandwidth satisfaction rate, network link utilization, and cost. Because *k* is tunable, one can adjust our *k*-SPF (*k*-BAR) algorithm according to the requirement of the network provider.

## REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," in *Proceedings of ACM SIGCOMM Computer Communication Review*, pp. 69–74, New York, USA, April 2008.

[2] B. Shen, B. Hao, and A. Sen, "On Multipath Routing Using Widest Pair of Disjoint Paths," in *Proceedings of Workshop on High Performance Switching and Routing*, pp. 134–140, Phoenix, USA, April 2004.

[3] Y. Li and D. Pan, "OpenFlow Based Load Balancing for Fat-Tree Networks with Multipath Support," in *Proceedings of IEEE ICC*, pp. 1–5, Budapest, Hungary, June 2013.

[4] T. He, D. Goeckel, R. Raghavendra, and D. Towsley, "Endhost-Based Shortest Path Routing in Dynamic Networks: An Online Learning Approach," in *Proceedings of IEEE INFOCOM*, pp. 2202–2210, Turin, Italy, April 2013.

[5] Open Networking Foundation, "Software-Defined Networking: the new norm for networks," 2012. https://www.opennetworking.org/images /stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[6] "Openflow switch specification v1.3.4," 2014. https://www.opennetworking.org/sdn-resources/onf-specifications

[7] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings of ACM SIGCOMM*, pp. 63–74, Seattle, USA, August 2008.

[8] T. H. Cormen, C. E. Leiserson, B. L. Rivest, and C. Stein, "Introduction to Algorithms," 3rd ed., Cambridge: MIT Press, 2009, pp. 658–664.

[9] J. Y. Yen, "Finding the K Shortest Loopless Paths in a Network," *Management Science*, Vol. 7, No. 11, pp. 712–716, July 1971.

[10] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proceedings of ACM SIGCOMM*, pp. 1–12, Kyoto, Japan, August 2007.

[11] Sai Qian Zhang , Qi Zhang ; Bannazadeh, H. Leon-Garcia, A., "Routing Algorithms for Network Function Virtualization Enabled Multicast Topology on SDN", *IEEE Transactions on Network and Service Management,* Vol 12, Issue 4, 2015.

[12] Liang-Hao Huang , Hui-Ju Hung ; Chih-Chung Lin ; De-Nian Yang, "Scalable and bandwidth-efficient multicast for software-defined networks" , IEEE Global Communications Conference (GLOBECOM), p 1890 – 1896, Austin, TX, December 2014

[13] Junjie Zhang, Kang Xi ; Min Luo ; Chao, H.J. "Load balancing for multiple traffic matrices using SDN hybrid routing "15th International Conference on High Performance Switching and Routing (HPSR), p 44 - 49 , Vancouver, BC , 2014

[14] Thomas Kohler, Frank Dürr and Kurt Rothermel "Update Consistency in Software-defined Networking based Multicast Networks" IEEE conference on Network function virtualization and Software defined networks (NFV-SDN), SanFrancisco, USA,. 2015

[15] Martin J. Reed, Mays Al-Naday, Nikolaos Thomos, Dirk Trossen, George Petropoulos, Spiros Spirou, "*Stateless multicast switching in software defined networks* ", Nov 2015. http://arxiv.org/abs/1511.06069