Computer Networks 103 (2016) 44–55

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

An efficient routing algorithm based on segment routing in software-defined networking

Ming-Chieh Lee, Jang-Ping Sheu*

Department of Computer Science, National Tsing Hua University, Taiwan

ARTICLE INFO

Article history:
Received 9 September 2015
Revised 18 December 2015
Accepted 28 March 2016
Available online 8 April 2016

Keywords:
Routing algorithm
Software-defined networking
Segment routing
Traffic engineering

ABSTRACT

Software-defined networking (SDN) is an emerging architecture that offers advantages over traditional network architecture. Segment routing (SR) defines the path of information through the network via an ordered list of multi-protocol label switching (MPLS) mechanisms on the packet headers at the ingress device, and this system makes SDN routing management more simple and efficient. SR can also solve some scalability issues in SDN. In this paper, we propose a routing algorithm for SDN with SR that can meet the bandwidth requirements of routing requests. Our algorithm considers the balance of traffic load and reduces the extra cost of packet header size in a network. Simulation results show that the performance of our algorithm is better than that of previously developed algorithms in terms of the average network throughput and the average rejection rate of routing requests.

1. Introduction

Software-defined networking (SDN) [1] is an emerging architecture that is flexible, manageable and responsive to rapid changes. This architecture enables the network controls to be directly programmed through an open standardized interface called OpenFlow [2]. The SDN paradigm provides clear advantages over traditional network architecture, and it has attracted growing attention from many businesses in recent years. SDN provides fine-grained traffic distribution control in a network, as it can decide forwarding behavior based on different combinations of packet header fields, unlike traditional coarse-grained systems of destination-based forwarding. However, the SDN attribute implies that an SDN switch requires a larger sized flow table than that of a traditional switch for storing the same number of flows [3]. Flow tables are implemented by ternary content addressable memory (TCAM) [4], which is an extremely expensive and power-hungry resource. In general, TCAM can support from a few hundred to several thousand entries which direct SDN in facing challenges such as network scalability [5].

SR is currently undergoing an Internet Engineering Task Force (IETF) draft [6], which is driven by Cisco and is supported by many leading telecom companies. In general, the SR approach shows promise as an alternative network-operating model. SR is a network technology that offers a new method of packet forwarding that minimizes the need for keeping large numbers of network information states and therefore helps to overcome the TCAM deficiency problem. This approach can definitely add capacity to IPs and multi-protocol label switching (MPLS) networks, as the SR data plane can be applied to IPv6 and MPLS. There are many advantages when we integrate SR in the data plane and in SDN-based control layer technologies. As for scalability and agility, SR avoids the requirement for millions of label codings to be stored in each network device along the path, and it reduces the number of forwarding rules in the TCAM. Furthermore, SR eliminates the complexity of maintaining large numbers of forwarding rules, so there are no communication delays between network devices and the SDN controllers [7].

In this paper, we study the traffic engineering issue in SDN [8] with SR. We show that through traffic engineering, we can achieve the goals of optimizing network performance and network resource utilization. With SR, the SDN controller only needs to encode end-to-end route information into the packet header as an ordered list of labels to the ingress network device. The controller does not need to add forwarding rules to each individual device along the path, and it can re-send a packet without the need for new forwarding state redistribution in the related switches. In addition, the intermediate nodes do not need to maintain any per-flow state. Intermediate nodes simply act on the routed behavior, which is recorded in the segment header. This approach provides a much more simple and scalable solution for traffic engineering. However, as OpenFlow [9] defines each MPLS label with 32 bits (which is a large label), SR has a drawback in that it requires extra per-packet header size overhead. This requirement arises because

* Corresponding author.
E-mail addresses: s102062516@m102.nthu.edu.tw (M.-C. Lee), sheujp@cs.nthu.edu.tw (J.-P. Sheu).

http://dx.doi.org/10.1016/j.comnet.2016.03.017
1389-1286/

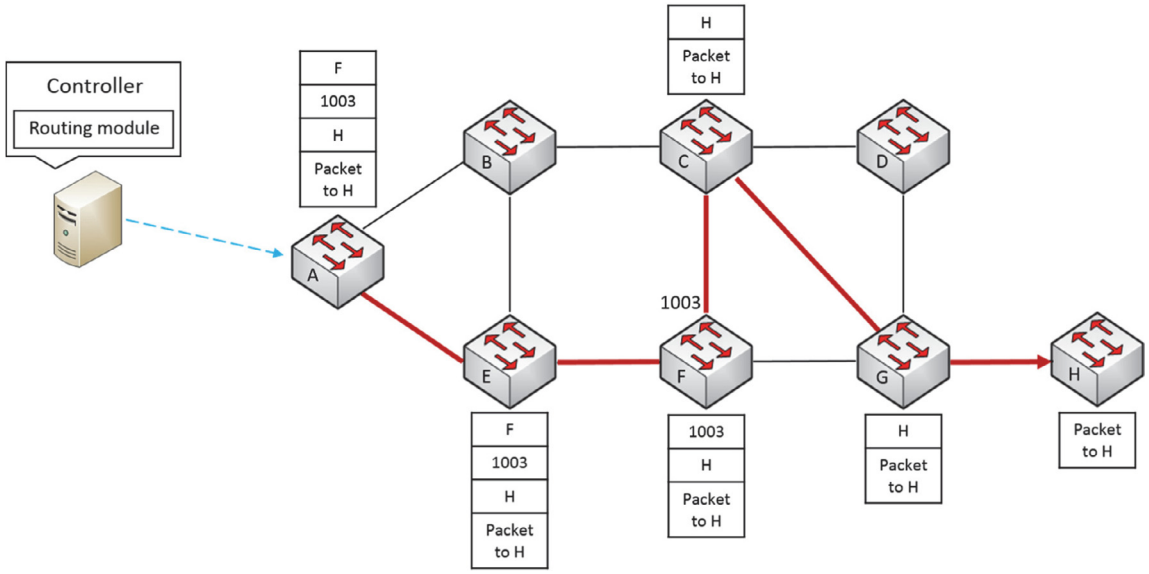

Fig. 2.1. SR traffic engineering with an SDN controller.

SR represents the route information as a set of labels, and it implements the labels by using the MPLS label field in the packet header.

In this paper, we propose a heuristic routing algorithm with a bandwidth guarantee. The proposed algorithm not only achieves the goal of using traffic engineering to balance the network traffic load, but it also promises to reduce the network overhead cost. In other words, our algorithm adopts an extra-hop limitation, and it considers a link's residual bandwidth and link criticality. Link criticality is based on the concept of the minimal interference routing method (MIRA) [10]. This concept is applied to minimize the possibility of rejecting requests when the network becomes overloaded, with an aim of upgrading the network's performance. We obtain a better network performance than that of other traditional routing algorithms such as the shortest path first (SPF) [11], widest shortest path (WSP) [12], shortest widest path (SWP) [13] or MIRA [10] algorithms. Extensive simulations show that our algorithm can lower the unsatisfied request rate and raise the bandwidth satisfaction rate compared to that of the previous approaches.

The rest of this paper is organized as follows. We introduce the related previous works in Section 2. In Section 3, we present our proposed algorithm. The performance evaluation is presented in Section 4, and Section 5 concludes the paper.

2. Related works

2.1. Preliminary of segment routing

For each pair of host communications within the same SDN/SR domain, a network uses an interior gateway protocol (IGP), such as the open shortest path first (OSPF) protocol, to enable routing to the destination by default. SR [6] uses a *node segment* to represent the action that a packet should follow to take the shortest route. In other words, every node in the same SR domain keeps a *node segment* information link to each of the other nodes in its forwarding table, as the default rules and the *node segment* represents global awareness. In addition, SR adopts an *adjacency segment* to control traffic, which represents the action that the packet should transfer to a specific egress data link with an adjacent node, and this *adjacency segment* represents local awareness. However, unlike in the *node segment*, each node only needs to install its local *adjacency segment* rules in its forwarding table. The combination of *node segments* and *adjacency segments* forms a sequence list of labels that are applied to the packet header by the SDN controller, and they are reflected instantly as the desired traffic path. Therefore, SR brings several orders of scaling gains, as it does not hold any state for the flows in the intermediate devices. However, SR contributes to another noticeable problem, because it uses an MPLS label field to place the segment labels. As a result, SR may require a larger packet header, which reduces the available bandwidth.

We illustrate an overview of intra-domain SDN-based SR in Fig. 2.1. The mark on each switch is the *node segment*, and the number next to the switch, *F*, is the *adjacency segment* of switch *F*. The SDN controller computes an explicit route by the routing module, and it configures the forwarding table of the ingress switch with an ordered list of segments. For example, assume that there is a traffic demand from switch *A* to *H*. The path information {*A-E-F-C-G-H*} is encoded in the packet header as a set of MPLS label stacks, which completely removes the need for installing rules in the switches along the path. First, the switch *A* processes the top label, which is a node segment, and then forwards the packet along the shortest path toward the network device with the *node segment F*. Then, switch *F* pops the top label, and the following top label is 1003. Therefore, switch *F* forwards the packet toward its output port 1003. After arriving at switch *C*, the switch forwards the packet to *H* along the shortest path, and so on. Thus, SR can reduce the number of forwarding rules in TCAM.

The following paragraphs summarize some of the previous studies related to the SR technology. First, the authors in [14] consider the problem of determining the optimal parameters for SR in offline and online cases. The authors propose a traffic matrix oblivious algorithm for the offline case, and another algorithm for the online case. In the online case, the network has a centralized controller that can use an online approach to solve the SR problem, as is done in our SDN-based environment. These authors give formulas and linear programs for defining the optimal parameters of SR. Their paper focuses on determining the optimal parameters as traffic split values. These values are applied to minimize the worst-case link utilization by taking equal-cost multipath routing (ECMP) into account in offline cases. The traffic split values also serve to minimize rejections of requests in online cases. However, our research is dedicated to designing an efficient routing algorithm for better network performance (such as improved

network throughput and rejection rate) and to focus on finding the single shortest path without ECMP. We consider the link criticality and the link residual bandwidth in evaluating the weight of a link. Moreover, our algorithm limits the maximal path length of a route to reduce the network's consumption of bandwidth.

The authors in [15] raise the issue of energy consumption when deploying large-scale distributed infrastructures. They propose the use of SR-based energy-efficient traffic engineering to reduce the energy consumption of backbone networks. Through the SDN approach, the network can selectively switch off a subset of links. The authors implement this technique in the OMNET++ simulator that dynamically decides the number of power-on links, and therefore saves energy. The goal of these authors is to determine the status of network devices, regardless of whether they are required to transfer data. Our paper, however, aims to build a bandwidth-satisfying path that increases the total network throughput and reduces the request rejection rate, rather than decreasing the energy consumption.

The authors in [16] introduce an architecture that integrates the SDN paradigm with SR-based traffic engineering. Their paper focuses on the problem of mapping computed paths onto SR paths. They propose an SR path assignment algorithm that aims to find the shortest list of segments corresponding to the desired path. Our work, however, focuses on the issue of a routing algorithm rather than an SR label assignment.

The authors in [17] propose a segment list-encoding algorithm to express a given path, which minimizes the segment list depth in SR-based networks. In addition, these authors provide a method for ECMP-aware shortest path computation that is subject to a considered set of multiple constraints. Our work also takes the label stack depth into consideration, and we propose a routing algorithm to reduce the extra cost of the packet header, which is caused by label stack depth. In addition to the label stack depth issue, we consider the balance of traffic load in our routing algorithm, with an aim to improve performance in terms of the network throughput and the rejection rate. As a result, our work focuses on the issue of improving the routing algorithm rather than solving the segment list computation problem.

In [18], an SR technology is implemented in a multi-layer network test bed. The authors design an SDN-based SR solution to control network edge nodes for configuring the label stacks. This paper also demonstrates scalability tests for different label stacking conditions. The main issue dealt with in their work is the implementation and demonstration of SDN-based SR. However, the authors do not discuss the routing algorithm for the SDN controller to configure the route in edge nodes.

2.2. Traffic engineering

Many existing works treat traffic engineering as an indispensable tool for upgrading network performance, which works by explicitly directing the traffic through a finite, competitive network resource. Traffic engineering configures the routing scheme to control how traffic is routed across the network, to optimize network performance and use network resources efficiently. It is critical that traffic-engineering methods utilize the measured traffic matrix for the diagnosis and management of network congestion. The traffic matrix represents the volume of traffic between sets of source-and-destination pairs over a given time interval, and this element is a key factor in network planning. The estimation of the traffic matrix involves simultaneously gathering traffic flow measurements and information on routing. SDN allows a more dynamic network measurement, which can more easily determine the precise and timely traffic matrix due to OpenFlow and the centralized controller [19,20]. As a result, when we study traffic engineering, we can acquire predictions of future traffic trends through the estimated traffic matrix, and we can find good routing configurations [21].

The major objectives of traffic engineering include distributing the network load to all links and diminishing network congestion. In such engineering, we want to prevent some links in the network from being overloaded, and to prevent other links from being underutilized due to a static path selection scheme. In addition, the chosen paths need to satisfy some constraints, such as a bounded end-to-end delay or a guaranteed bandwidth requirement to meet specified quality of service (QoS) delivery standards.

We introduce several traffic-engineering studies that are used to solve various aspects of our problem. One of the simplest and most frequently used algorithms for unicast routing is the SPF (shortest path first) algorithm, which is adopted by a common standard routing protocol called OSPF (open shortest path first) [11]. This algorithm selects a path that has the least number of links and uses the smallest amount of resources. However, this approach does not consider the available capacity of all feasible candidate links in its path selection, and thus it may cause some links to become congested earlier, which can lead to poor resource utilization.

In [12,13], the routing strategies used include the widest-shortest path (WSP) and the shortest-widest path (SWP) algorithms. The width of a path represents the available bandwidth, and the length commonly corresponds to the number of hops. Therefore, the objective of the WSP algorithm is to select the shortest path that has the largest amount of residual bandwidth. The main result of applying the WSP algorithm is to reduce network cost, because this algorithm focuses on resource preservation by choosing the minimum hop count path. The SWP algorithm, however, finds the maximum amount of available bandwidth from the source node to the destination node. If there are multiple paths with the same maximum available bandwidth, this algorithm selects the shortest path. The main result of using the SWP algorithm is load balancing, due to selection of the path with the maximum available bandwidth among all feasible paths for each request. However, using the SWP algorithm involves the risk of increased network cost, as the widest path generally means a longer hop count path, which ties up more resources and thus decreases network throughput.

In [10], the authors introduce the concept of "interference," which considers that routing a flow along a particular path can reduce the maximum flow between some other pairs. This concept suggests that a newly routed connection should follow a path that does not cause too much interference to other paths whose links may be critical to future traffic demands for other pairs of hosts. These authors prove that the interference problem is NP-hard, and they propose a heuristic minimum interference routing algorithm (MIRA) to maximize the minimum-maximum flow between all of the other source-destinations. The authors show that the MIRA's number of rejections is lower than that of other algorithms that do not consider the minimum interference criteria. However, the MIRA also has some shortcomings. For example, the complexity of repeated maximum flow computation requires $O(VE^2)$, and the MIRA concentrates on the effect of interference on critical links only, thus ignoring the non-critical links (where V is the number of switches and E is the number of links between switches). Therefore, the routing path lengths can become long enough to make a path practically unusable.

3. Routing algorithm

3.1. Problem definition

Our research aims to enable Internet service providers, who are responsible for ensuring that traffic is routed over feasible paths

in their networks in ways that meet specific service guarantees. Therefore, the problem to be solved in this paper is to decide a routing plan for unicast communication in SDN that is based on SR strategy. The goal of our proposed routing algorithm is to compute an explicit bandwidth-satisfying path between a pair of communication nodes under a given QoS requirement. To increase the network throughput, the algorithm needs to reduce the potential for rejecting traffic demands and network congestion, and to attain a state of efficient network resource usage from a long-term perspective. In addition, we need to pay attention to the path hop count, because this factor directly causes extra wastage of network resources, due to the extra bandwidth occupied by the segment lists in the packet header.

Before presenting our proposed algorithm, we define the notations used in this paper. An SDN network topology can be represented as a weighted graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges. Each vertex in V and each edge in E represents a switch and a switch link in SDN, respectively. For each edge $e \in E$, let $b(e)$ denote the currently available bandwidth of a link that connects a pair of nodes. We define a weight function on each edge $e \in E$, which we call the link weight $w(e)$. We assume that $w(e) > 0$, and we use this assumption to calculate the optimal route, which is the minimum weight path for a pair of communication nodes. As the minimum weight path may result in a longer path length, a longer delay time, a larger packet header overhead and more consumption of link bandwidth resources than a smaller path length, our algorithm aims to limit the maximum path length between the source and the destination.

We also define a traffic matrix (TM), which records the traffic demands in a time interval P. That is, the TM is the set of communication nodes observed in a time interval P, and they are considered as the prediction of potential traffic demands in the near future. The traffic demand is represented as a request from a source $s \in V$ to a destination $d \in V$, with requested bandwidth B_{sd}. In addition, we define H_{sd} as the minimum hop count path between source node s and destination node d. Let MAX_{sd} be the tolerable maximum hop count between source node s and destination node d, *which is equal to* H_{sd}, plus an extra-hop count E_{sd}. The extra-hop count is used to limit the maximum path length of our solution. In this paper, we propose a routing algorithm to balance the link loads of switches under these path length restrictions.

3.2. Routing algorithm

In the proposed algorithm, we consider information regarding the link's residual bandwidth, path length and link criticality to select a path from source to destination. The link criticality is used to predict the future traffic load of the links. We consider each node's location in the network as the criterion for defining the criticality of a certain link. This concept is similar to that of betweenness centrality in graph theory [22], which is widely used to analyze social networks. The betweenness centrality of a node v is the average ratio by which a node s needs to pass through a specific node v to reach a node t via the shortest path in the network. In previous work, the betweenness centrality of a specific node v, $BC(v)$, is given by $\sum_{s \neq v \neq t} \sigma_{st}(v)/\sigma_{st}$. The 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. Clearly, a node with higher betweenness centrality has greater influence in its network, and thus it is prone to become the bottleneck.

We modify this idea from emphasizing node centrality to emphasizing link criticality, which means that the degree of criticality in a link is expected to carry the traffic between all of the possible pairs of nodes. We find the first k shortest paths as a means to measure every link's corresponding criticality, instead of finding all the possible routes between a pair of nodes. Finding all of the possible routes is not feasible, as the number of paths grows rapidly with the number of network nodes and links. We therefore assume that the first k shortest paths of a certain pair of nodes is qualified to represent the possibility of each link's criticality. This assumption is warranted because routes with shorter path lengths typically tend to utilize less network resources. As a result, we suppose that the optimal route with the minimum weight is one of the first k shortest candidate routes.

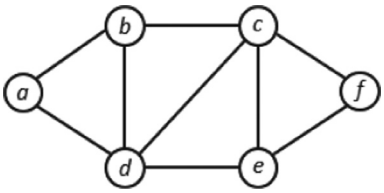

Fig. 3.1. Example of computing link criticality.

At the beginning of our research, we can only infer the link criticality by assuming that all pairs of nodes have an equal possibility for communication. That is, every host pair in the network can potentially establish a connection, and the measure of link criticality is mainly derived from the network topology. We also assume that we will re-compute and update every link criticality value in the time interval P. As the SDN controller has a global view of the network, it is easy for the controller to keep a network TM at any time. The statistical TM is used to predict the future traffic demands based on the observed communication pairs of nodes at the time interval P. Therefore, the SDN controller regards the estimated communicating pairs as the TM, rather than all of the pairs of nodes in the network.

The authors in [23–25] mention that they observe the related property of traffic predictability. That is, they assume that the traffic distribution observed in the recent past will hold in the future. In other words, packets exhibit a strong temporal and spatial locality phenomenon, rather than a purely random arrival pattern, or some other commonly used arrival pattern such as the Poisson distribution. Thus, we compute the first k shortest paths for every new pair in the TM at every time interval P to form the new link criticality value for every link. This link criticality parameter can help to defer loading on highly critical links, and it can prevent critical network resources from being quickly exhausted. In other words, the algorithm tends to choose the links that can balance the loads across the network.

Let K_{sd} denote a set of the first k shortest routes between a pair of source and destination nodes (s, d), and let $K_{sd}(e)$ be the number of times that the first k shortest routes include the link $e \in E$. In that case, $K_{sd}(e)/k$ is considered as the occurrence rate of link e in the first k shortest paths of the pair (s, d). Thus, we compute the first k shortest paths between every possible pair of nodes in the TM to determine the criticality of each network link. We express the total expected load on link e as the sum of the expected number of demands on the link e from all possible (s, d) pairs of nodes in the TM. We define the equation of link criticality $c(e)$ as Eq. (1). We can use the algorithm in [26] to compute the first k shortest paths. The intent behind this link criticality parameter is to characterize how likely it is that a particular link e can become the bottleneck link that has a high potential to be the path for any pair of communication nodes in the network.

$$c(e) = \sum_{\forall (s,\ d) \in \mathrm{TM}} K_{sd}(e)/k \tag{1}$$

Fig. 3.1 gives an example of the measurement of each link criticality in a small network. We illustrate $c(e)$ with $k = 2$ in this example. Assume that the TM includes three pairs of nodes (a, f), (d, f) and (a, e). Then, K_{af}, K_{df} and K_{ae} are {*a-d-c-f, a-d-e-f*}, {*d-c-f, d-e-*

Table 1
Measurement of link criticality.

Link	$c(e)$	Link	$c(e)$
a-b	0	d-c	1.5
b-c	0	d-e	1.5
c-f	1	e-c	0.5
a-d	2	e-f	1
b-d	0		

f} and {a-d-e, a-d-c-e}, respectively. Note that if there is more than one set of the first k shortest paths between two nodes, we randomly select one of them as the first k shortest paths. For example, in Fig. 3.1, there are three sets of the first two shortest paths between nodes a and e, which are {a-d-e, a-b-c-e}, {a-d-e, a-b-d-e} and {a-d-e, a-d-c-e}. We randomly select {a-d-e, a-d-c-e} as the K_{ae}. Thus, the link criticality value of a-d is 4/2, because K_{af} and K_{ae} pass through the link a-d twice. The link criticality value of d-c is 3/2, because K_{af}, K_{df} and K_{ae} each pass through the link d-c once. Table 1 shows the link criticality $c(e)$ of each edge e that is derived from the TM.

The link criticality value $c(e)$ is used to predict the traffic load of a link e in the next time interval P. As the current loading of each link e is also an important factor for the routing performance, we also consider the current loading status of each link $e \in E$. Let $f(e)$ denote the total amount of traffic flows carried by the link e. Let $b(e)$ denote the remaining bandwidth of the link e. Thus, $s(e) = f(e)/b(e)$ is denoted as the congestion index of the link e. The definition of the link congestion index $s(e)$ is an increasing and convex function. A convex function is a continuous function whose value at the midpoint of every interval in its domain never exceeds the arithmetic mean of its values at the end of the interval. Some related works [27,28] discuss the benefit of adopting increasing and convex functions to set the link weight. The function $s(e)$ increases quickly when the amount of traffic flow passing through the link e approaches its capacity. That is, heavily loaded links incur a heavy penalty, as computed by $s(e)$. Therefore, the principal intuition behind $s(e)$ is to use the link with the lightest utilization. As $s(e)$ increases rapidly when link utilization grows beyond a certain threshold, we can capture the severity of congestion based on the immediate link utilization information.

In this situation, we use the link criticality value $c(e)$ and the link congestion index $s(e)$ to designate the weight of link e, which is denoted as $w(e)$. The link weight $w(e)$ is presented in formula (2) which encourages the use of links with less criticality and of links with more available bandwidth. The link weight function tends to refer to the link criticality $c(e)$ when the network load is light, and it tends to refer to the link congestion index $s(e)$ when the network load is heavy, because $s(e)$ tends to rise sharply along with the level of link utilization. In addition, when the network resource utility becomes saturated, it is reasonable to determine the link weight by the link congestion index rather than by the expected traffic load. After evaluating the link weight $w(e)$, our algorithm deletes the links that are not capable of providing the requested bandwidth resource B_{sd}. Finally, the weight of a path is the sum of the weight on every link of the path.

$$w(e) = c(e)\alpha + s(e)\,(1-\alpha), 0 < \alpha < 1 \tag{2}$$

As we mention above, a path with a higher hop count has more segment lists in the packet header. In addition, the longer the path is, the more expensive it is in terms of the total amount of network resources that are consumed. Thus, our problem lies in selecting the minimum weight path from a given source s to a destination d, for which the path length is no longer than a predetermined threshold MAX_{sd}. The optimal path selection under such constraints, and especially under additive constraints, is an NP-complete problem. However, the problem can be solved in polynomial time if the constraint parameter is restricted and fixed [29]. In our problem, for example, the constraint is the hop count. That is, we can solve the problem in polynomial time. We rely on Bellman–Ford algorithm [30] to solve the MAX_{sd}, which is hop-constrained for the minimum weight path problem.

Bellman–Ford algorithm is based on the principle of relaxation, in which the correct distance is gradually replaced by new smaller values until it eventually reaches the optimum solution. The longest possible path that is reachable from the source without a cycle has $|V|-1$ edges in the graph. The Bellman–Ford algorithm proceeds by relaxing all of the edges, and the procedure needs to be repeated $|V|-1$ times. We can determine that the Bellman–Ford algorithm proceeds by increasing the number of hop counts. That is, at each iteration, the algorithm compares the path from the source to a vertex of path length $x+1$ and the path of length x from the previous iteration. This process gradually generates the shortest paths from a single source node to all of the other nodes for each hop count at each relaxation round. As a result, Bellman–Ford algorithm only needs to iterate MAX_{sd} times at most to do the relaxation procedure and to identify the minimum weight path between a given source and a destination under the MAX_{sd} hop count constraint.

We illustrate the procedure of Bellman–Ford algorithm with a hop count constraint in Fig. 3.2. To illustrate the algorithm procedure, we represent the example with a directed graph. Note that an undirected graph also satisfies the concept of the directed graph, because each undirected edge of a graph can be regarded as two oppositely directed edges. The value on each edge is the link weight $w(e)$. For each vertex $v \in V$, the $p(v)$ is used to indicate the process of tracing back the desired path. The $d(v)$ on each vertex denotes the minimum weight of the path from the source vertex to v. In this example, we assume that the source vertex is a, and the destination vertex is e. The minimum hop count from a to e is 2, which is expressed by $H_{ae} = 2$, and we assume that the extra-hop count $E_{ae} = 2$. That is, $MAX_{ae} = 4$ in this example. As a result, we need to do the relaxation procedure four times to get the eligible path. The relaxation technique serves to check if we can improve the value of the minimum weight path. After a relaxation step, we may update the estimated weight of the minimum weight path.

Fig. 3.2(a) shows the initial state of a graph G. Initially, all of the $d(v)$ are set as ∞, except for the source vertex a, which is set as zero. Fig. 3.2(b) shows the result after the first relaxation step. $d(b)$ is updated to 5, because we can find another less weighted path to vertex b via vertex a with $d(a) + w(e(a, b)) < d(b)$. The equation $p(b, 1) = a$ means that the vertex b is relaxed from the vertex a with the first relaxation step. The result of the second relaxation is shown in Fig. 3.2(c). We can determine that the minimum weight path from a to e is {a-d-e}, with weight $d(e) = 9$. Similarly, the result of the third relaxation is shown in Fig. 3.2(d). After conducting the relaxation step four times, we can get the minimum weight path from a to e, as shown in Fig. 3.2(e). In other words, $d(e)$ is the minimum weight from source a to destination e after using at most four hops. We can trace back the final path by using the $p(e)$ from the destination vertex e. $p(e, 4) = c$ means that the parent of the vertex e is c in the fourth relaxation step. Thus, we can continue the tracing work from vertex c with a decreasing hop count by using $p(c, 3) = b$ in Fig. 3.2(d). Similarly, we check $p(b, 2) = d$ in Fig. 3.2(c). This process is repeated until we trace back to the source vertex a by applying $p(d, 1) = a$, as seen in Fig. 3.2(b). Then we can obtain the final path of {a-d-b-c-e} with weight 7 under the MAX_{ae} hop constraint. Without the hop count constraint, the minimum weight path between source a and destination e is {a-d-b-c-f-e} with weight 6.5, as is shown in Fig. 3.2(f).

In conclusion, our method computes the $c(e)$ and $s(e)$ to dynamically compose the link weight $w(e)$, and the links that have

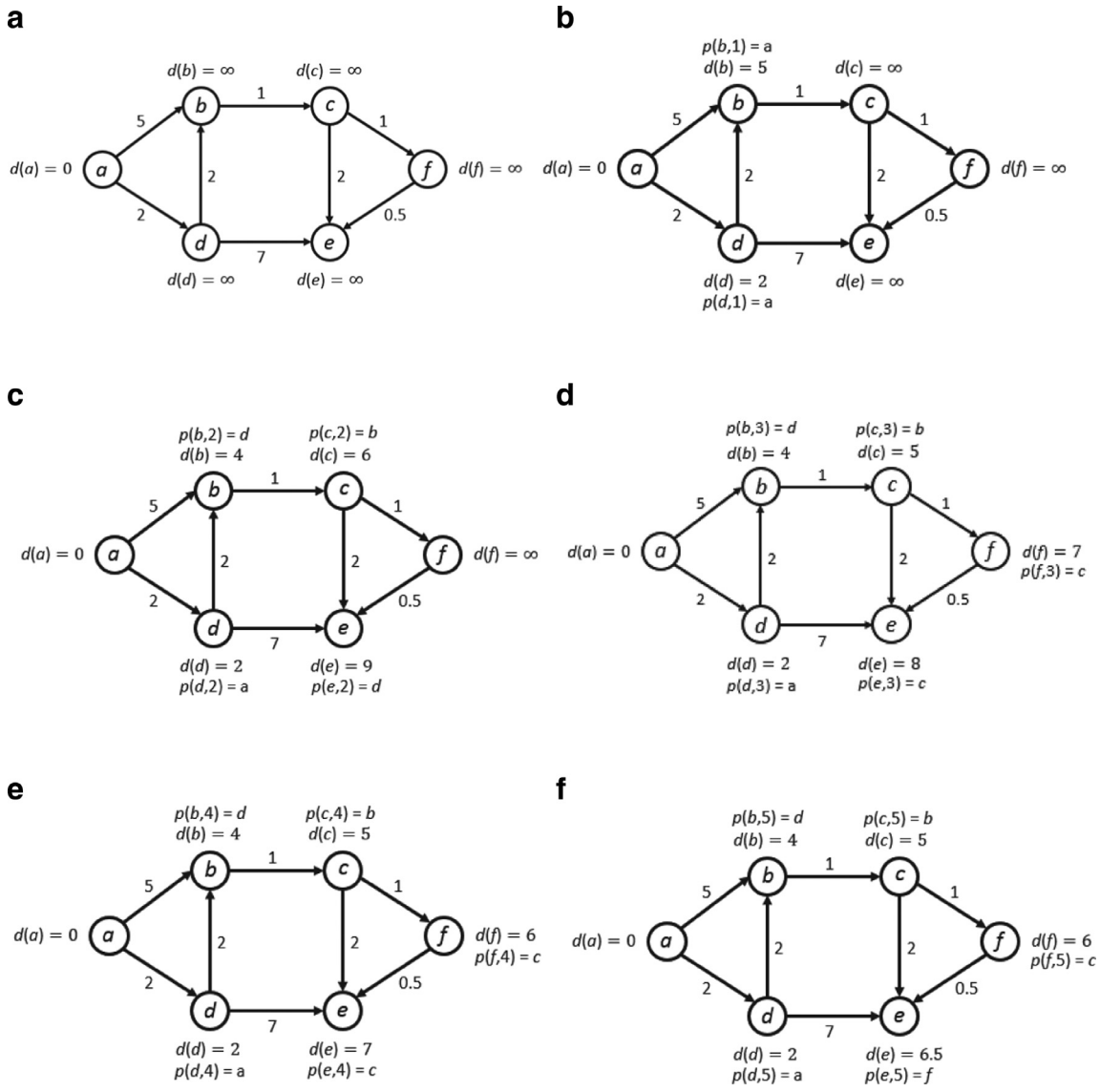

Fig. 3.2. Example of the hop count constraint Bellman–Ford algorithm.

a residual bandwidth smaller than the requested bandwidth are eliminated beforehand. Last, the Bellman–Ford algorithm with a hop count constraint is executed using the link weight $w(e)$ as computed from (2), whenever a flow setup request arrives at the network.

Algorithm 1: *Generate the weighted graph G*

1. Traffic matrices TM is updated after a time interval
2. Find the first k shortest paths for $\forall(s, d) \in$ TM
3. Compute the link criticality $c(e)$ according to Eq. (1)
4. For each edge $e \in E$
5. Compute the link congestion index $s(e)=f(e)/b(e)$
6. Compute the link weight $w(e)$ according to Eq. (2)
7. Delete the edges that are less than B_{sd} bandwidth requirement
8. Return G

3.3. Time complexity

The time complexity of finding the first k shortest paths algorithm is $O(k(|V|\log|V|+|E|))$, where $|V|$ is the vertices number, and $|E|$ is the edges number [26]. The first k shortest paths are computed every time interval P. As P is a small time interval, the number of communication pairs in the time interval is constant. So, the total time complexity in computing the first k shortest paths of nodes in TM is $O(k(|V|\log|V|+|E|))$. The time complexity of counting the link weight value is $O(E)$. The time complexity of finding an optimal path under the hop count constraint is $O(MAX_{sd}\ E)$. As $MAX_{sd} \ll |V|$, the total running time of our algorithm is $O(|V|\log|V|+|E|)$. The proposed algorithm is summarized as follows.

4. Performance evaluation

In this section, we compare the performance of our proposed routing algorithm with the SPF [11], WSP [12], SWP [13] and MIRA [10] algorithms in terms of rejection rate, network throughput, average link utilization and average length of selected paths. In addition, we compare the performance variation with different parameters applied in our algorithm. We run our simulation using JAVA programming language to model different network environments and different traffic flow distributions. The experiments with different network size variations are also investigated in this paper.

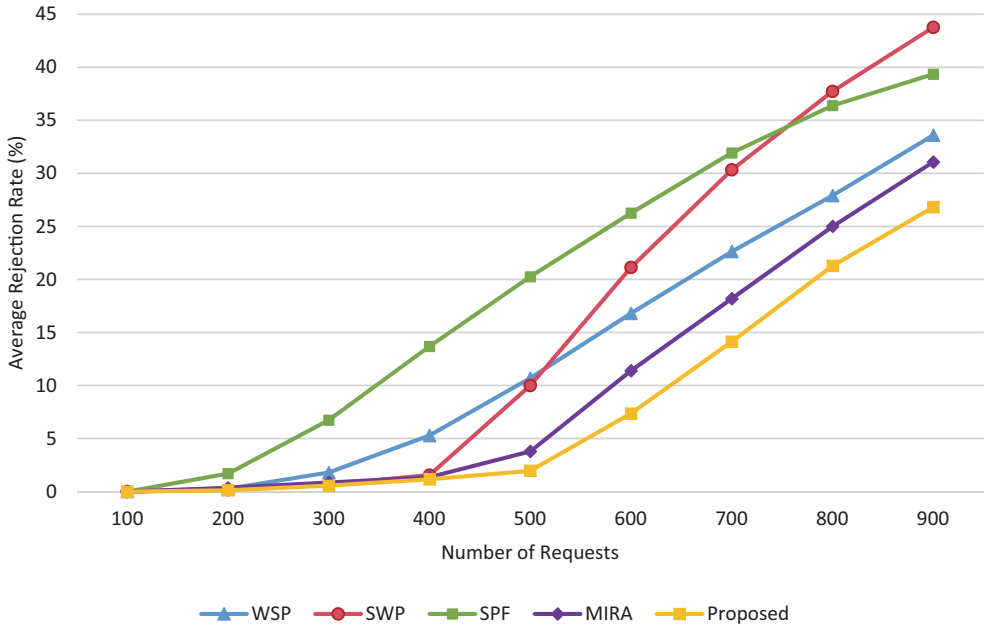

Fig. 4.1. Average rejection rate vs. number of requests.

Algorithm 2: *Find a minimum weight path for a weighted graph $G = (V, E)$ with MAX_{sd} hops*

```
1.  for each vertex v ∈ V do
2.    d(v) ← ∞
3.  end for
4.  d(s) ← 0 for source vertex
5.  for i ← 1 to MAX_sd do
6.    for each edge e(u, v) ∈ E with weight w(e(u, v)) do
7.      if d(u)+w(e(u, v)) < d(v) then
8.          d(v) ← d(u)+w(e(u, v))
9.          p(v, i) ← u /* the parent of vertex v is vertex u at i
iteration relaxation*/
10.       end if
11.     end for
12.   end for
13.   /* Trace back the route R from destination vertex with p(d, j) */
14.   cur_vertex ← destination vertex d
15.   add cur_vertex to R
16.   for j ← MAX_sd to 1 do
17.     if p(cur_vertex, j) exists then
18.       cur_vertex ← p(cur_vertex, j)
19.       add cur_vertex to R
20.     end if
21.   end for
22.   Return R and update the residual link bandwidth of the path R
```

The topologies of these synthetic networks are randomly created by using the Waxman method [31]. The probability of having an edge between node u and node v is given by the formula $xe^{-d/(yL)}$, where $0 < x, y \leq 1$ are the model parameters, d is the Euclidean distance between u and v, and L is the maximum Euclidean distance between any two nodes of the graph. An increase of the parameter x results in a graph with higher link density, and an increase in y yields a larger ratio of long edges to short edges. The generated Waxman topologies are 60 nodes with 201 links, 80 nodes with 314 links, 100 nodes with 413 links and 150 nodes with 668 links. In the simulations, each source host sends flow packets to the destination host, and the flow sizes range from 10 MB/s to 100 MB/s. The default link capacity is 1 GB/s. The parameter α is 0.5 and k is 5, where k stands for the first k shortest paths to compute the link criticality. In addition, the time interval P is set as the arriving of 100 requests, and the extra-hop E_{sd} is set to 3 in our following simulations.

4.1. Simulation results

Fig. 4.1 shows the average rejection rates from using the SPF, WSP, SWP, MIRA and our proposed algorithm. The average rejection rate means the number of rejected requests as a function of the total number of requests that have arrived at the network. When the network cannot provide a path with sufficient bandwidth, it rejects the request due to the QoS strategy. In this test, the SPF algorithm performs the worst, as it rejects the highest number of requests, and this algorithm is the first to start rejecting requests. That is, the SPF always chooses the minimum hop count path, without considering the factor of network bandwidth. The SPF definitely uses up resources of the same path rapidly, and it contributes to unbalanced network resource use. The WSP performs better than the SPF, because it is an improvement of the SPF. The WSP avoids congestion by taking the widest path when more than one shortest path exists. However, the capability for load balancing by the WSP is limited, because this algorithm is still restricted to routes on the shortest paths. In addition, the WSP performs better than the SWP in an overall perspective, although the SWP also avoids network congestion by always selecting the widest path. The SWP performs better than the WSP only when the network is under a light load. However, the SWP may choose a longer path length, which contributes to an increase in extra network resource consumption. It is obvious that the rejection rate of SWP rises rapidly and sharply after it starts rejecting requests, especially in a more heavily loaded network. Neither the SWP nor the WSP considers the link criticality concept, and these algorithms might decide to route on paths that are critical to various pairs of nodes, thus causing congestion in the future. The MIRA, however, determines its route with a goal of causing minimum interference in the network, and it performs well. Undoubtedly, an algorithm that considers critical links has a better performance than the SPF, SWP or WSP. However, the MIRA only considers the critical links, so it may result in choosing many non-critical links, which can increase the path length and consume the network links resource. Our proposed algorithm considers link criticality and link usable bandwidth at the same time. In addition, our algorithm avoids taking a lengthy path, which is beneficial for saving network resources.

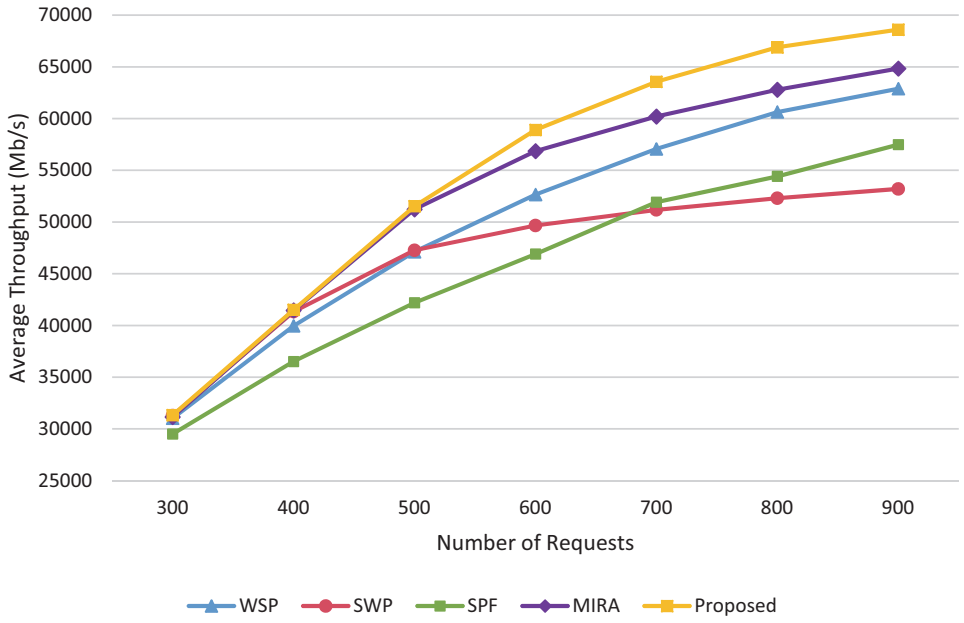

Fig. 4.2. Average network throughput vs. number of requests.

Fig. 4.2 shows the average network throughput of the five algorithms. The network throughput is the amount of satisfied bandwidth that is routed successfully as a function of the total amount of requested bandwidth that arrives at the network. In general, the average network throughput increases when the requested bandwidth increases. We can observe that the trends of the average rejection rate and the average network throughput are similar. Our method finds the highest network throughput, and this indicates that our algorithm performs better than all of the other benchmark algorithms. The SPF gets the lowest throughput overall, but the SWP performs even worse than the SPF when the network load becomes heavy, because the SWP tends to exhaust the network's resources earlier. The WSP shows a higher throughput than either the SPF or the SWP. The MIRA performs better than the WSP, but the difference between their throughputs becomes similar when the network load becomes heavy.

Fig. 4.3 shows the average link utilization, which is the total consumption of bandwidth by each link over the total default capacity of the network. In general, a higher rejection rate results in lower link utilization, because most of the requests are not routed. Accordingly, the SPF has the lowest average link utilization. The WSP's link utilization is higher than that of the SPF, because the throughput of the WSP is higher than that of the SPF. The link utilization from our method lies between that of the WSP and the MIRA. We get a higher link utilization than that of the WSP, because the network throughput of our method is better than WSP. However, the rates of SWP and MIRA link utilization are higher than that of our proposed algorithm, even though our network throughput is the highest. The SWP demonstrates the highest average link utilization through its attribute of routing on longer paths. Hence, the SWP utilizes much more network resources, despite its high rejection rate in a heavily loaded network. The MIRA also shows a tendency to route on longer paths, due to its lack of consideration for non-critical links. As a result, we can infer that the average load of a network strongly relates to the length of its traffic routes.

Fig. 4.4 shows the average hop count of various algorithms. The SPF and WSP algorithms have the lowest hop counts, because they use the shortest path strategy. It is understandable that the SWP shows the highest hop count among all of the schemes, because it finds a path with the maximum bandwidth bottleneck in the network. The SWP may need to make a detour to get to the widest path. In addition, the MIRA suffers from the similar problem of consuming more network resources by taking longer paths. We can verify this tendency by observing the MIRA hop count performance. The average hop count of our proposed method with extra-hop $E_{sd} = 3$ lies in the middle of all the compared algorithms.

In the previously presented simulations, we obtain the performance of the average rejection rate, the average network throughput, the average link utilization and the average hop count of a network with 60 nodes. In the following simulations, we evaluate the average rejection rate and average network throughput for networks of different sizes. Experiments on networks of differing sizes demonstrate that our algorithm performs well without network environment restriction. Simulations for various network sizes are conducted by using network topologies, which are 80 nodes, 100 nodes and 150 nodes.

Figs. 4.5 and 4.6 show the comparison between various schemes with incremental network sizes. For each network size, we fix the factor of network loading as a standard. That is, we capture the data under the same accumulated requested bandwidth as a function of the total network default capacity. We can infer from the experimental results that the SPF performs the worst among all of the schemes, in that it has the lowest throughput and the highest rejection rate under many tested topologies. The WSP and SWP show similar behavior to that seen in Figs. 4.1 and 4.2 on different networks. In addition, the MIRA performs well on every topology, but the time complexity of the MIRA cannot be tolerated in online routing, especially in a large-sized network. In conclusion, our proposed algorithm performs the best no matter what the size of the network, and it has reasonable time complexity for dynamic routing.

Fig. 4.7 shows the performance of the average rejection rate under different extra-hop constraints. The hop count constraint is used to limit the path length and to save the link bandwidth resource. The path with a minimum weight summation may take a longer path length than the minimum hop count to reach the destination. However, SR needs extra labels to define each hop, and the larger packet header sizes consume larger amounts of bandwidth resources. As a result, we use MAX_{sd} (which is $E_{sd}+H_{sd}$) as

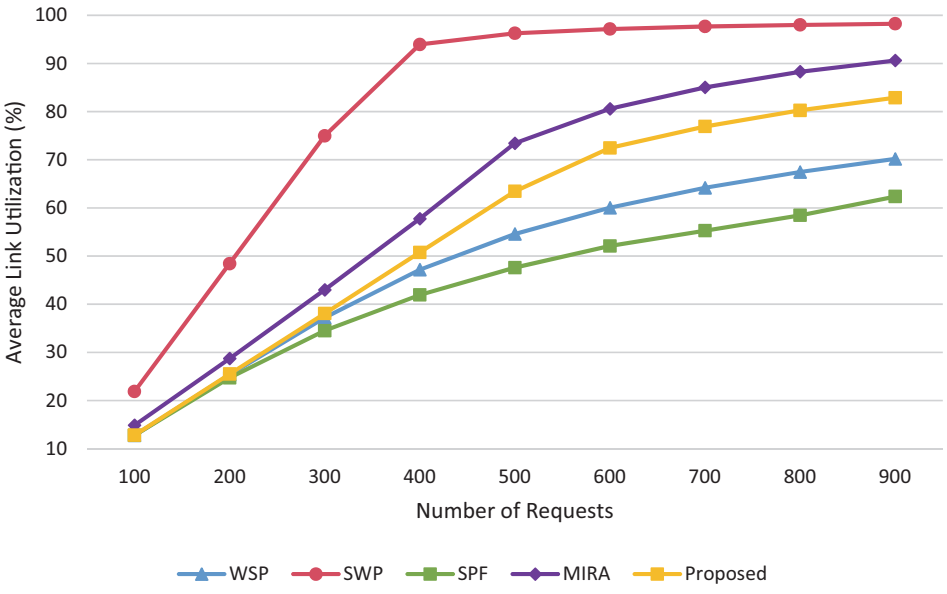

Fig. 4.3. Average link utilization vs. number of requests.

Fig. 4.4. Average hop counts of various schemes.

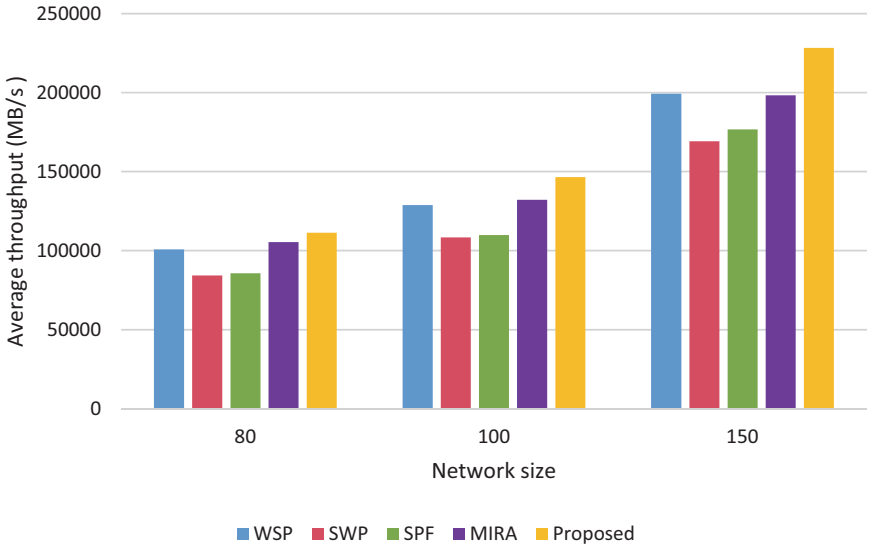

Fig. 4.5. Average throughput under different network sizes.

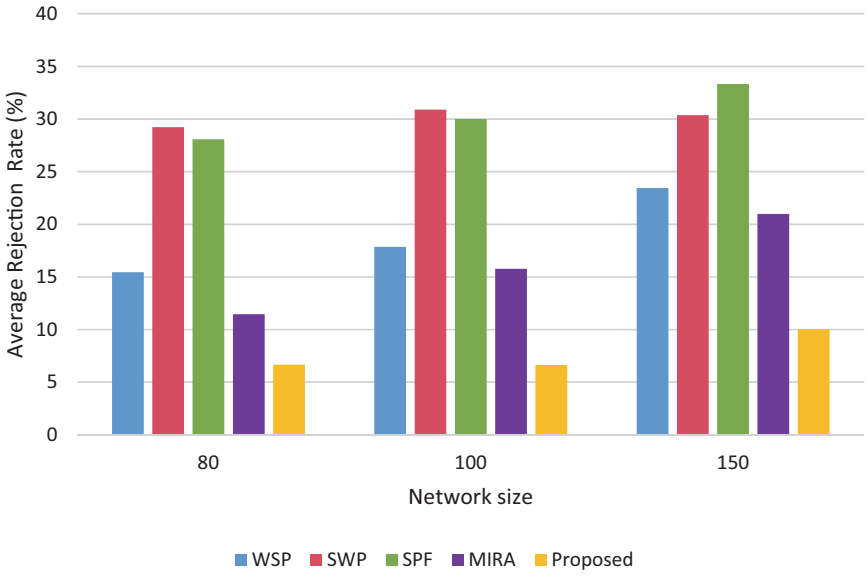

Fig. 4.6. Average rejection rate under different network sizes.

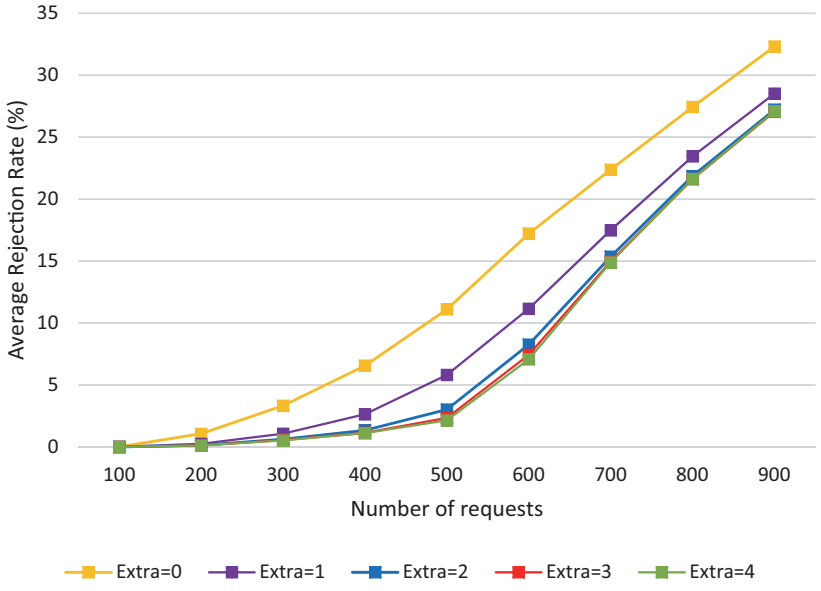

Fig. 4.7. Average rejection rate under different extra-hop constraints.

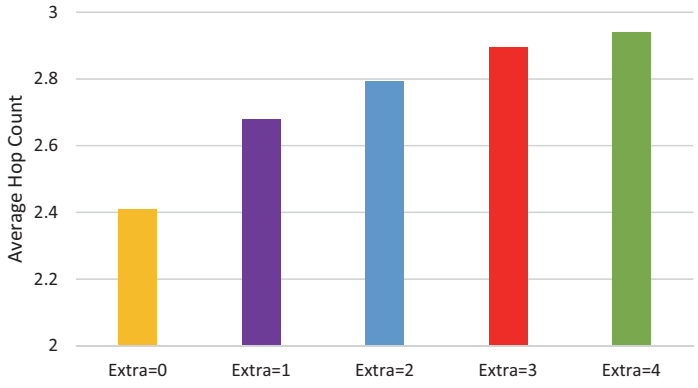

Fig. 4.8. Average hop count under different extra-hop constraints.

a restriction, to avoid the requirement for numerous labels. We can deduce that the larger the E_{sd} is, the better performance we can get, because we increase the chances of getting the smaller-weighted routing path. However, the improved performance ratio declines as the extra-hop E_{sd} grows larger than two hops. Fig. 4.8 shows the average hop count of the proposed algorithm under different extra-hop constraints. The larger the E_{sd} is, the more the cost also tends to increase, as the path length may be longer. That is, there is a tradeoff between the network performance and the network overhead. The more hops we take, the better performance we get, but we consume more network resources.

5. Conclusion

In this paper, we introduce an efficient heuristic algorithm for SR in software-defined networking. The goal is to build a bandwidth-satisfying path between a pair of communication nodes. In addition, the algorithm should minimize the possibility of rejecting traffic demands to increase the total network throughput. Our proposed algorithm takes the link criticality and link residual bandwidth into consideration. Thus, we define a new link weight-setting scheme to reduce network congestion. Moreover, the proposed algorithm also considers the extra network overhead caused by the segment labels in the packet headers. We limit the path length as a constraint to save network resources. The simulation results show that our method outperforms other routing algorithms in terms of average rejection rate and average network throughput under different network sizes. In addition, the time complexity of our algorithm is reasonable for dynamic online routing.

References

[1] ONF Market Education Committee. “Software-Defined Networking: The New Norm for Networks,” ONF White Paper, Palo Alto, US: Open Networking Foundation, 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.

[3] K. Kannan, S. Banerjee, Compact TCAM: flow entry compaction in TCAM for power aware SDN, Distrib. Comput. Netw. 7730 (2013) 439–444.

[4] Y. Kanizo, D. Hay, I. Keslassy, Palette: distributing tables in software-defined networks, in: Proceedings of IEEE INFOCOM, 2013.

[5] S.H. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, IEEE Commun. Mag. 51 (2) (2010) 136–141.

[6] “Segment Routing Architecture.” IETF Draft, https://www.ietf.org/id/draft-ietf-spring-segment-routing-06, 2015.

[7] “SDN State Reduction.” IETF Draft, https://tools.ietf.org/html/draft-ashwood-sdnrg-state-reduction-00, 2013.

[8] S. Agarwal, M. Kodialam, T.V. Lakshman, Traffic engineering in software defined networks, in: Proceedings of IEEE INFOCOM, 2013, pp. 2211–2219.

[9] “OpenFlow Switch Specification.” 2011, http://www.openflow.org.

[10] M. Kodialam, T.V. Lakshman, Minimum interference routing with applications to MPLS traffic engineering, in: Proceedings of IEEE INFOCOM, 2000, pp. 884–893.

[11] B. Fortz, J. Rexford, M. Thorup, Traffic engineering with traditional IP routing protocols, IEEE Commun. Mag. 40 (10) (2002) 118–124.

[12] R. Guerin, D. Williams, A. Orda, “QoS Routing Mechanisms and OSPF Extensions,” IETF RFC 2676, 1999.

[13] Z. Wang, J. Crowcroft, Quality of service routing for supporting multimedia applications, IEEE J. Sel. Areas Commun. 14 (1996) 1228–1234.

[14] R. Bhatia, Fang Hao, M. Kodialam, T.V. Lakshman, Optimized network traffic engineering using segment routing, in: Proceedings of IEEE INFOCOM, 2015, pp. 657–665.

[15] R. Carpa, O. Gluck, L. Lefevre, Segment routing based traffic engineering for energy efficient backbone networks, in: *Proceedings of IEEE Advanced Networks and Telecommuncations Systems* (*ANTS*), 2014, pp. 1–6.

[16] L. Davoli, L. Veltri, P.L. Ventre, G. Siracusano, S. Salsano, Traffic engineering with segment routing: SDN-based architectural design and open source implementation, in: *Proceedings of European Workshop Software Defined Networks* (*EWSDN*), 2015, pp. 111–112.

[17] F. Lazzeri, G. Bruno, J. Nijhof, A. Giorgetti, P. Castoldi, Efficient label encoding in segment-routing enabled optical networks, in: *Proceedings of International Conference of Optical Network Design and Modeling* (*ONDM*), 2015, pp. 34–38.

[18] A. Sgambelluri, A. Giorgetti, F. Cugini, G. Bruno, F. Lazzeri, P. Castoldi, First demonstration of SDN-based segment routing in multi-layer networks, in: *Proceedings of Optical Fiber Communication Conference, OSA Technical Digest* (*Optical Society of America*), 2015.

[19] M. Malboubi, L. Wang, C.N. Chuah, P. Sharma, Intelligent SDN based traffic aggregation and measurement paradigm (iSTAMP), in: Proceedings of IEEE INFOCOM, 2014, pp. 934–942.

[20] A. Tootoonchian, M. Ghobadi, Y. Ganjali, OpenTM: traffic matrix estimator for openflow networks, in: Proceedings of PAM, 2010, pp. 201–210.

[21] Y. Han, F. Moutarde, Statistical traffic state analysis in large-scale transportation networks using locality-preserving non-negative matrix factorization, Intell. Transp. Syst. IET (2013) 283–295.

[22] L.C. Freeman, Centrality in social networks: conceptual clarification, Soc. Netw. (1979) 215–239.

[23] S. Deng, H. Balakrishnan, Traffic-aware techniques to reduce 3G/LTE wireless energy consumption, in: Proceedings of CoNEXT, 2012, pp. 181–192.

[24] T. Benson, A. Anand, A. Akella, M. Zhang, MicroTE: fine grained traffic engineering for data centers, in: Proceedings of CoNEXT, 2011.

[25] R. Jain, S. Routhier, Packet trains-measurements and a new model for computer network traffic, IEEE J. Sel. Areas Commun. (2006) 986–995.

[26] N. Katoh, T. Ibaraki, H. Mine, An efficient algorithm for k shortest simple paths, Networks 12 (1982) 411–427.

[27] B. Fortz, M. Thorup, Internet traffic engineering by optimizing OSP weights, in: Proceedings of INFOCOM, 2000.

[28] B. Fortz, M. Thorup, OSPF/IS-IS weights in a changing world, IEEE J.n Sel. Areas Commun. 4 (2) (2002) 756–767.

[29] S. Upadhaya, G. Devi, Characterization of QoS based routing algorithms, Int. J. Comput. Sci. Emerg. Technol. 1 (3) (2010) 133–141.

[30] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.

[31] B.M. Waxman, Routing of multipoint connections, IEEE J. Sel. Areas Commun. 6 (9) (1988) 1617–1622.

Jang-Ping Sheu received the B.S. degree in computer science from Tamkang University, Taiwan, Republic of China, in 1981, and the M.S. and Ph.D. degrees in computer science from National Tsing Hua University, Taiwan, Republic of China, in 1983 and 1987, respectively.
He is currently a Chair Professor of the Department of Computer Science and Associate Dean of the College of Electrical and Computer Science, National Tsing Hua University. He was a Chair of Department of Computer Science and Information Engineering, National Central University from 1997 to 1999. He was a Director of Computer Center, National Central University from 2003 to 2006. He was a Director of Computer and Communication Research Center from 2009 to 2015, National Tsing Hua University. His current research interests include wireless communications, mobile computing, and software-defined networks. He was an associate editor of the IEEE Transactions on Parallel and Distributed Systems and International Journal of Sensor Networks. He is an associate editor of the International Journal of Ad Hoc and Ubiquitous Computing.
He received the Distinguished Research Awards of the National Science Council of the Republic of China in 1993–1994, 1995–1996, and 1997–1998. He received the Distinguished Engineering Professor Award of the Chinese Institute of Engineers in 2003. He received the K. -T. Li Research Breakthrough Award of the Institute of Information and Computing Machinery in 2007. He received the Y. Z. Hsu Scientific Chair Professor Award and Pan Wen Yuan Outstanding Research Award in 2009 and 2014, respectively. Dr. Sheu is an IEEE Fellow and a member of Phi Tau Phi Society.

Ming-Chieh Lee received the Master degree in Computer Science from National Tsing-Hua University, Taiwan, R.O.C in 2015. Her research interests include software-defined networks and wireless networks.