International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 15, Number 2, pages 220–239, July 2025

A Flow Distribution Algorithm with Segment Routing for Software-Defined Networking

Momoka Mizuno
Graduate School of Science and Technology,
University of Tsukuba, Tsukuba, Ibaraki, 305-8573, Japan

Shigetomo Kimura
Institute of Systems and Information Engineering,
University of Tsukuba, Tsukuba, Ibaraki, 305-8573, Japan

Received: February 14, 2025
Revised: May 5, 2025
Accepted: June 6, 2025
Communicated by Hideharu Kojima

Abstract

Previously, routers have been responsible for both data forwarding and network management. Software-Defined Networking (SDN) separates these functions into a data plane and a control plane in order to simplify network operations and enable the construction of programmable networks. However, flow updates transmitted using the OpenFlow protocol introduce a communication overhead. Since Ternary Content Addressable Memory (TCAM) is used to quickly search flow entries, the associated costs rise significantly as the scale of the network increases. To address these problems, a method has been proposed that reduces flow entry update work and minimizes traffic overhead by aggregating some flows on a specific path in an SDN-managed network using SR (Segment Routing)-MPLS (Multi-Protocol Label Switching). However, while overall work is reduced, the load on the specific path used for the flow aggregation increases.

This paper proposes a flow distribution algorithm with SR that efficiently utilizes network resources, while also addressing the two aforementioned limitations of Software-Defined Networking (SDN) to enable a more reliable SDN infrastructure. To evaluate the proposed algorithm, simulation experiments compared the proposed algorithm with the shortest-path algorithm on four network topologies with 13, 24, 48 or 58 nodes, and our results showed that the average standard deviation of the number of packets forwarded by each node under the proposed method was 861.04–1496.17 packets lower by comparison with the shortest-path method when the total number of transmitted packets was 50,000, and 1743.37–2950.34 packets lower when the total number of transmitted packets was 100,000. We also noted that the average path length for each packet under the proposed method was just 0.15–1.08 hops longer than that of the shortest-path method.

Keywords: Flow Distribution Algorithm, Segment Routing, Traffic Engineering, Software-Defined Networking

1 Introduction

Software-Defined Networking (SDN) has been a focus of interest due to its ability to simplify network operations and enable the construction of programmable networks. Previously, routers have been

responsible for both data forwarding and network management. SDN separates these functions into a data plane and a control plane. In the data plane, SDN switches forward packets based on their flow entries, while a centralized controller manages the control plane. For example, SDN can implement traffic engineering (TE) [1], [2] to improve network utilization and enhance performance in terms of latency and packet loss [3].

The communication between the SDN switches and the controller commonly uses the OpenFlow protocol [4], [5]. When an SDN switch receives a packet whose forwarding switch cannot be determined, it sends a Packet-In message including the packet to the controller. After the controller receives the message, it installs, updates, or deletes some entries in the flow entries of the switch.

Each flow entry contains match fields, counters, and actions that determine how the switch should process the packet such as forwarding to a particular port, dropping the packet, or modifying the packet headers. Furthermore, OpenFlow supports a wide range of match fields between layer 2 and layer 4 headers, as well as packet metadata for precise and complex flow definitions.

However, SDN presents two significant problems. The first problem is that traffic overhead arises between the SDN controller and the data plane as each switch's flow entry is periodically updated, which may interfere with data forwarding. The second problem is that since Ternary Content Addressable Memory (TCAM) is used to quickly search flow entries, the associated costs rise significantly as the network scale increases [6], [7]. In the traditional SDN, forwarding information was stored in each switch's flow table and the SDN controller may update the flow table if needed. In order to resolve these two issues, Ziyong Li et al [8] have proposed Segment Routing (SR) that specifies flows through stacking segments. The first issue is addressed by eliminating the need to update the switch's flow table. For the second issue, even if the number of flows increases, the flows are specified through SR segments stacked on the packets, which prevents the TCAM from being overburdened.

This paper proposes a flow distribution algorithm with SR that efficiently utilizes network resources, while also addressing the two aforementioned limitations of Software-Defined Networking (SDN) to enable a more reliable SDN infrastructure [9].

The rest of this paper is organized as follows. Section 2 introduces MPLS (Multi-Protocol Label Switching) and SR (Segment Routing) and then MPLS is compared with SR. Section 3 covers previously published work that relates to our research. Section 4 explains the flow path determination method. Section 5 proposes the flow distribution algorithm. To show the effectiveness of the proposed algorithm, Section 6 explains the network simulation experiments that were conducted to compare the proposed algorithm with the shortest-path algorithm on three network topologies with 13, 24, or 58 nodes. The results of these experiments showed that the average standard deviation of the number of packets forwarded by each node under the proposed method was 917.65–1523.46 packets lower by comparison with the shortest-path method when the total number of transmitted packets was 50,000, and 1843.43–3025.4 packets lower when the total number of transmitted packets was 100,000. We noted that the average path length for each packet under the proposed method was just 0.15–0.81 hops longer than under the shortest-path method. Finally, Section 7 concludes the paper and discusses future work.

2 Segment Routing

This section introduces MPLS (Multi-Protocol Label Switching) and SR (Segment Routing) based on the proposed algorithm.

2.1 Multi-Protocol Label Switching

Larger networks operated by ISPs tend to employ Interior Gateway Protocols (IGPs) like OSPF for internal routing and typically select the shortest path for the internal router. However, problems can arise, depending on the network topology, that some links are overutilized, leading to congestion on these links.

To address this issue, traffic engineering techniques have been introduced, including Multi-Protocol Label Switching (MPLS) networks, among others. In MPLS, the Label Distribution Pro-

tocol (LDP) is generally used to exchange label information with neighbor routers and select the shortest path as the Label Switched Path (LSP).

When a communication begins and the first packet arrives at the entrance of the MPLS network, e.g., the upper left router in Fig. 1, its path within the MPLS network for the communication is determined, and the label identification for this path is attached before the packet header. Routers refer to this label for rapid packet forwarding. Subsequent packets with the same destination follow the same path. Since the label is unique only per link, but not across the whole MPSL network, each router swaps the label when forwarding a packet (the upper middle and upper right routers in Fig. 1) or pops one when the packet is exiting the MPLS network (the lower right router in Fig. 1).

Explicit routing protocols, such as Resource Reservation Protocol-Traffic Engineering (RSVP-TE) allow for the selection of non-shortest paths where this benefits traffic distribution. For example, if less frequently used links are given higher priority, then traffic distribution is more balanced across the entire network.

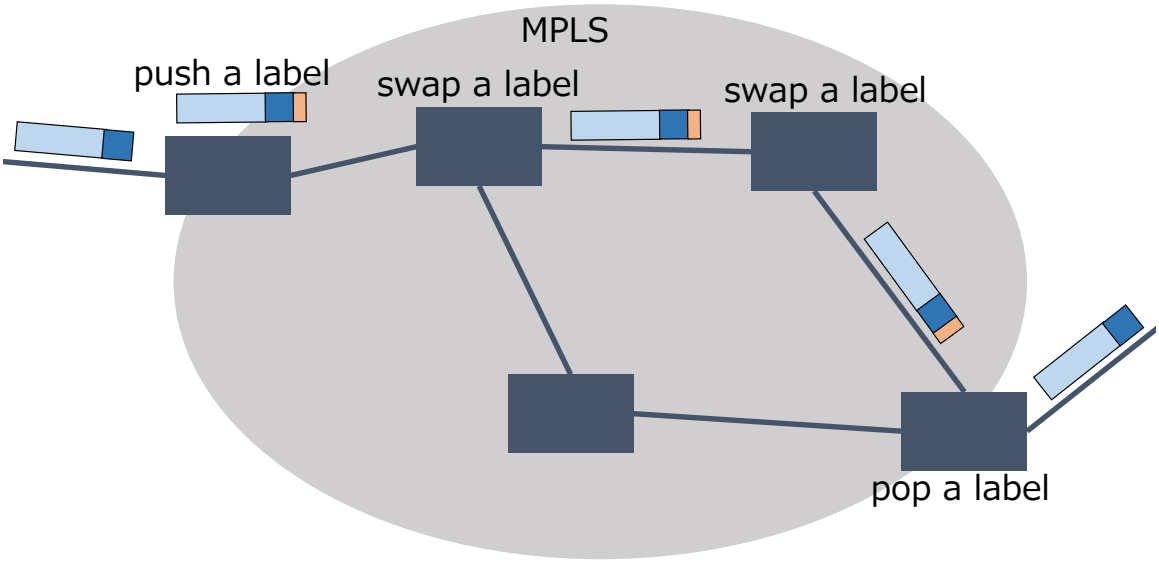

Figure 1: Label Switching of MPLS

2.2 Segment Routing

Segment Routing (SR) is another traffic engineering technique in which the network is represented as a series of segments and packet forwarding is realized on the level of these segments. The most typical segments are node segments, which indicate specific nodes, and adjacency segments, which represent the adjacency relationships between nodes. Each node is assigned a Segment Identifier (SID). The node ID is unique across the network. The adjacency ID is a local value specifying the relationship between nodes.

In SR, the entrance node stacks segments into the packet header that represent the route from the entrance node to the exit node. The internal routers then simply forward the packet according to the segments in the packet header. Since the entrance node can stack multiple segments, SR also allows implementations of Traffic Engineering (TE) [10].

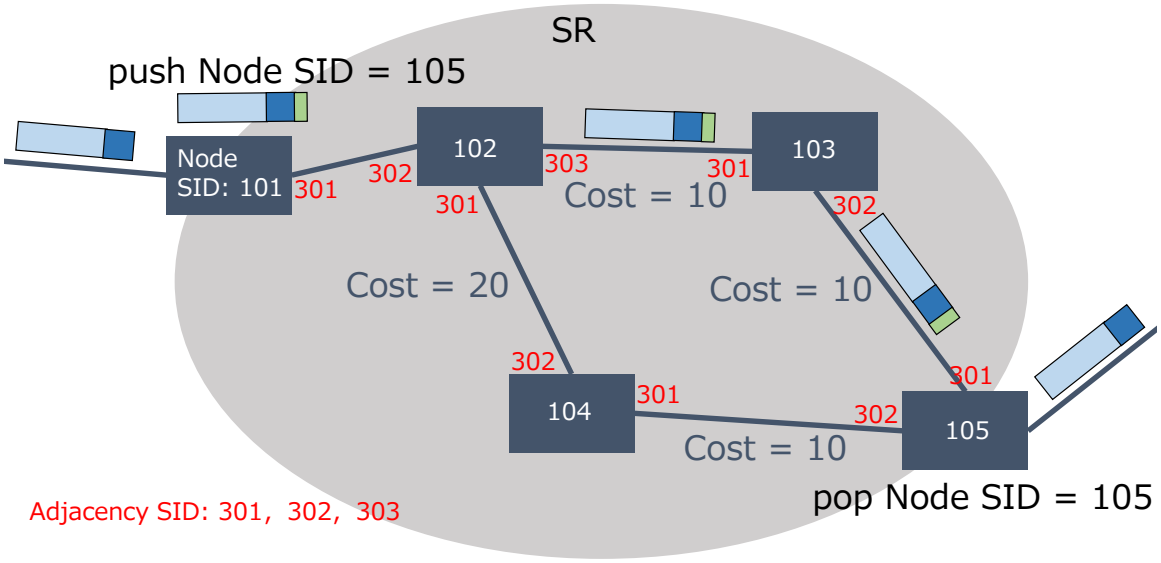

Figure 2: A Case Stacked SR Node SID = 105.

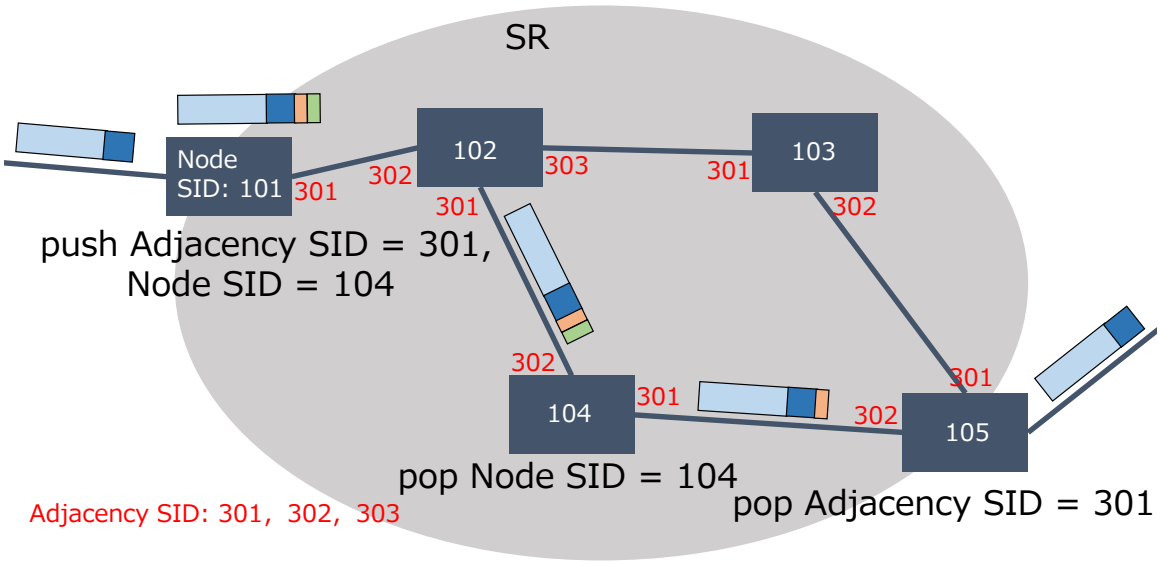

Figure 3: A Case Stacked SR Node SID = 104 and Adjacency SID = 301.

For instance, in the network illustrated in Fig. **??**sr_node, the entrance node at Node SID = 101 stacks a segment with Node SID = 105. The packet will traverse through the nodes with Node SID = 102, Node SID = 103 and Node SID = 105, which is the route with the minimum total cost. If the path needs to route through the node with Node SID = 104, the entrance node will stack the SIDs for Adjacency SID = 301 and Node SID = 104, as depicted in Fig. **??**sr_adj. From the

entrance node, the packet is forwarded via the node with Node SID = 102 and then to the node with Node SID = 104 with the minimum cost. When the packet reaches this node, the Node SID = 104 segment is popped from the stack, and the next SID in the stack, Adjacency SID = 301, is referred to forward the packet to the corresponding adjacency segment, and the packet then reaches the exit node with Node SID = 105.

Combining Node SIDs and Adjacency SIDs simplifies route policy management. Since SR natively supports Equal Cost Multi Path (ECMP), flow distribution across equal cost paths is also achieved.

2.3 MPLS vs SR

Although both MPLS and Segment Routing (SR) provide source routing technologies, SR is simpler to implement as a protocol and makes it relatively easy to write route policies. In fact, MPLS requires multiple protocols such as a Label Distribution Protocol (LDP) for label distribution and a Resource Reservation Protocol with Traffic Engineering extensions (RSVP-TE) for signaling. MPLS also relies on Interior Gateway Protocols (IGPs) such as Open Shortest Path First (OSPF) or Intermediate System to Intermediate System (IS-IS) for exchanging route information between nodes. In contrast, SR does not require LDP or RSVP-TE and can be implemented on a single IGP like OSPF or IS-IS. This leads to a simpler network with lower operational costs [11].

Furthermore, if traffic has to be distributed across multiple Equal Cost Multi-Paths (ECMPs) in MPLS, the entrance node needs to establish Label Switched Paths (LSPs) for each path. However, since SR has native support for ECMP, the traffic is automatically distributed across ECMPs. Moreover, SR allows straightforward path specification on a per-packet basis using Node IDs and Adjacency IDs.

However, in SR, the entrance node can only stack a limited number of SIDs, commonly referred to as the Maximum Stack Depth (MSD) [12]. Standard networks typically have such an upper limit. When a large number of segments are stacked, the packet header may exceed this stack depth.

3 Related Works

A multipath load balancing mechanism on SDN has been proposed to improve both high throughput and better Quality of Service (QoS) in the network [13]. This mechanism first employs the Depth First Search (DFS) algorithm within the SDN controller to generate a list of optimal paths based on routing distance. Communication between the controller and switches use the OpenFlow protocol. The SDN controller periodically collects the available bandwidth of each link as the link cost. The weights of all paths in the list obtained from the DFS algorithm are then calculated, and the optimal path is selected.

Two Traffic Engineering (TE) schemes i.e., End-to-End Segment Routing (e2e-SR) and Per-Domain Segment Routing (pd-SR) have been proposed for multi-domain networks [11]. When packets are transmitted to nodes in other domains, the source domain controller is typically unaware of the destination domain network, but for effective TE communication between SR controllers is needed. The e2e-SR scheme facilitates communication between controllers that enables the source domain's controller to acquire an end-to-end segment list. In the pd-SR scheme, the destination domain's controller sends a virtual label instead of the segment list to the source domain's controller. Since only the destination domain's controller can translate the virtual label into the correct segment list for forwarding, the confidential internal topology information is preserved.

An algorithm has been proposed that divides flows into multiple sub-flows and transfers a single flow across multiple paths [14]. When a large volume of flows arrives, the SDN controller first uses the MPTCP algorithm to divide each flow into sub-flows. Switches then forward the packets based on the SR segment list created by the controller, selecting as many sub-flows as possible that meet the transmission conditions.

While existing research demonstrates the usefulness of SDN-based MPTCP, the issue of TCAM memory limitations, as previously mentioned, remains unresolved.

4 Proposed Network Architecture

This section proposes a flow distribution algorithm with Segment Routing (SR) for a Software-Defined Networking (SDN) environment to improve the overall utilization of the network resources. The proposed algorithm generally specifies Node SIDs, but uses Adjacency SIDs to reduce the number of stacked labels in the header of each packet. However, each node is required to store all Node SIDs in the SR network. The native Equal Cost Multi-Path (ECMP) functions in SR are also utilized.

The SDN controller monitors the packet forwarding load across switches. If it detects that the load on a certain switch is too high, it uses Adjacency SIDs to redistribute the flows through the switch.

Fig. 4 explains the flow path determination method. SDN separates the network into a data plane and a control plane. The data plane consists of switches, while the control plane includes the SDN controller and a database that stores the network state. It is assumed that the SDN controller is aware of the topology of the data plane.

When a switch receives a new flow, ① the switch notifies the source and destination of the flow to the SDN controller. ② The controller refers to the network state stored its database. ③ Then, the controller determines the segments to be stacked on packets in the flow. ④ The controller sets the segments information to the flow entry of the switch. The switch appends the received list of segments to the packet header and forwards the packet based on the stacked segments. The computed path is stored for a predefined time in the switch to let subsequent packets in the same flow follow the same path.

Fig. 5 explains how the network state is maintained. The database in the control plane stores the number of packets forwarded by each switch. ① Once a flow has been forwarded, ② the switch reports the number of transmitted packets to the controller. ③ The controller stores this information in the database and uses it to determine the paths for new flows.

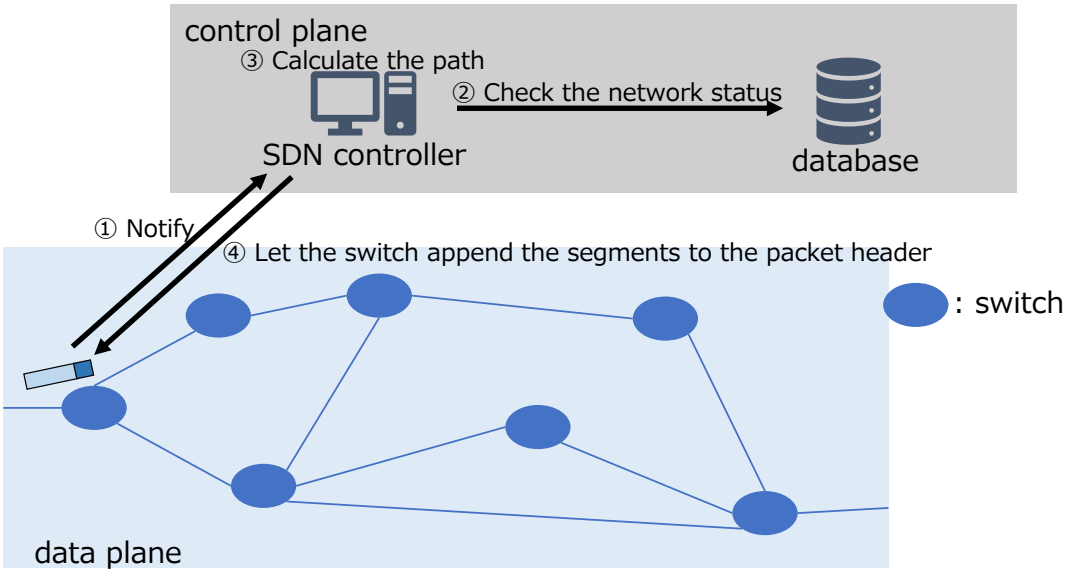

Figure 4: Path Decision Method.

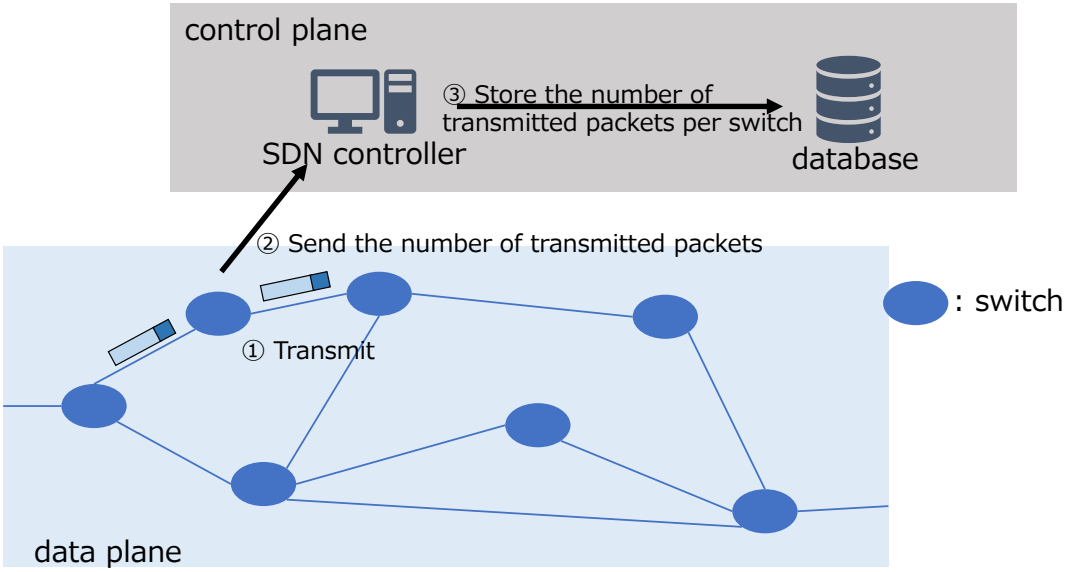

Figure 5: Save Method of Network Status.

In the proposed architecture, link loads are adjusted every 10,000 total transmitted packets to further distribute the packets. The next section provides a detailed explanation of this approach.

5 Proposed Flow Distribution Algorithm with SR for SDN

Algorithm 1 Change Node Alpha

Require: $\alpha, node[]$
  $sum \Leftarrow 0$
  for all $n \in node$ **do**
    $sum \Leftarrow sum + n$.packetCount
  end for
  $ave \Leftarrow sum/node$.number
  for all $n \in node$ **do**
    if n.packetCount $> ave * 1.1$ **then**
      $\alpha_n \Leftarrow \alpha_n - 1$
    else if n.packetCount $< ave * 0.9$ **then**
      n.link $= 1$
    else if n.isAlphaChanged $=$ **true then**
      $\alpha_n \Leftarrow \alpha$
    end if
  end for

α_n is a variable assigned to each node that governs the probability with which each generated path is selected. The initial value α is determined through experiments described in Section 6.2. α_n is dynamically adjusted by Algorithm 1.

To prevent switches being burdened with a higher load, Algorithm 1 adjusts the link cost and the weight value α_n for node n. This algorithm operates as an application within the SDN controller and calculates the average ave of the packet forwarding count of all nodes. When the value α_n

is higher, the probability that the switch will be selected for a path becomes lower. The variable n.count represents the number of packets forwarded by the node n, while n.link (initially set to 1) represents the weight of all adjacent links to node n.

If the packet forwarding count of a switch exceeds $ave*1.1$, the algorithm decrements the switch's α value by 1, which makes it harder for that switch to be selected for packets forwarding. Conversely, if the count is less than $ave * 0.9$, the costs of all adjacent links are reset to a minimum value of 1, which makes it easier for these links to be selected. For switches that do not meet either of these conditions, the switch's α_n value is set to the initial value α. Note that the values 1.1 and 0.9 were chosen based on preliminary experiments that used the values 1.05 and 1.1 for the upper boundary and 0.9 and 0.95 for the lower boundary. These preliminary experiments also attempted combinations of these boundaries. Although the results are omitted for a space constraint, in the majority cases, the combination of 1.1 for the upper boundary and 0.9 for the lower boundary gave good results. Preliminary experiments to determine the α parameter show in Section 6.2.

Algorithm 2 Change Link Costs

Require: $\alpha_n, \alpha, ave, node[]$
  $sum \Leftarrow 0$
  for all $n \in node$ **do**
    $sum \Leftarrow sum + n$.packetCount
  end for
  $ave \Leftarrow sum/node$.number
  for all $n \in node$ **do**
    if n.packetCount $> ave + \alpha_n$ **then**
      n.link $= \alpha$
    else if n.packetCount $< ave$ **then**
      n.link $= 1$
    end if
  end for

Algorithm 2 is for changing the link costs. In this algorithm, if the packet forwarding count n.count is greater than the average number ave of packets forwarded by all switches plus α_n, the cost n.link of adjacent links of a switch n is set to α. If n.count is less than ave, n.link is overwritten with a cost of 1.

Algorithm 3 Generate Path

Require: $sour\text{-}SID, dest\text{-}SID, node[]$
Ensure: $LIST$
  $flag \Leftarrow false$
  for all $n \in node$ **do**
    $flag \Leftarrow (flag$ **or** n.isLinkChanged)
  end for
  if not $flag$ /* no link changed */ **then**
    $LIST \Leftarrow$ NodeSID ($dest\text{-}SID$)
  else
    $LIST \Leftarrow []$
    for all $n \in$ ShortestPath($sour\text{-}SID, dest\text{-}SID$) **do**
      $LIST \Leftarrow LIST+$ AdjacencySID(n)
    end for
  end if
  return $LIST$

Algorithm 3 is for determining the SID to be stacked in the segment list. The functions NodeSID and AdjacencySID are used to stack the argument SID onto the label header as either a NodeSID

or an AdjacencySID, depending on the functions. This algorithm operates as an application within the SDN controller and is executed by the algorithm described in Fig. 4 ③. If the link cost is not adjusted for any link, the shortest path of SR with ECMP support is generated. Otherwise, the shortest path represented by Adjacency SIDs is generated.

6 Experiments and Evaluation

This section describes simulation experiments to evaluate the algorithm proposed in the previous section and compares the results with those for the shortest-path algorithm, which always selects the shortest path from the following perspectives:

- Standard deviation of the number of packets forwarded by each node.

- The number of hops until packet transmission is completed.

6.1 Simulation Settings

The network topologies used in the simulation experiments were the NSFNET (National Science Foundation Network) [15], the USA backbone IP network (USNET, int short) [15], the DFN (German National Research and Education Network for science and research) [16] and the PalmettoNet (backbone network in North Carolina, South Carolina, USA) [16] as shown in Fig. 6.

Fig. 7 shows the histogram of betweenness centrality for each topology. The horizontal axis represents betweenness centrality (a metric that indicates how frequently a node appears on the shortest paths between other nodes), and the vertical axis indicates the number of nodes.The small, medium, and non-hub large topologies exhibit a gently decreasing distribution, suggesting that many nodes possess moderate levels of betweenness centrality rather than a few nodes being exceptionally dominant.A distinctive characteristic of the small topology is the absence of completely peripheral nodes (i.e., nodes with zero betweenness centrality).In contrast, the large topology has an overwhelming number of nodes with near-zero betweenness centrality,while only four nodes exhibit extremely high betweenness centrality in the range of 0.2 to 0.5.

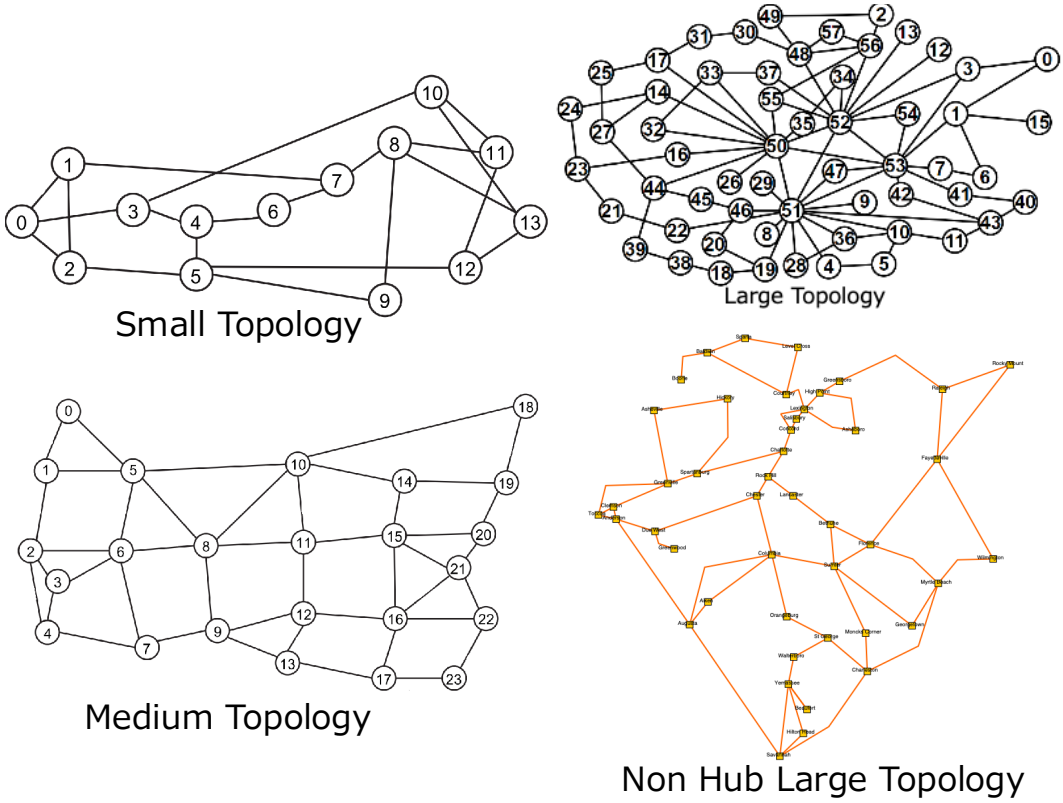

Figure 6: Network Topologies

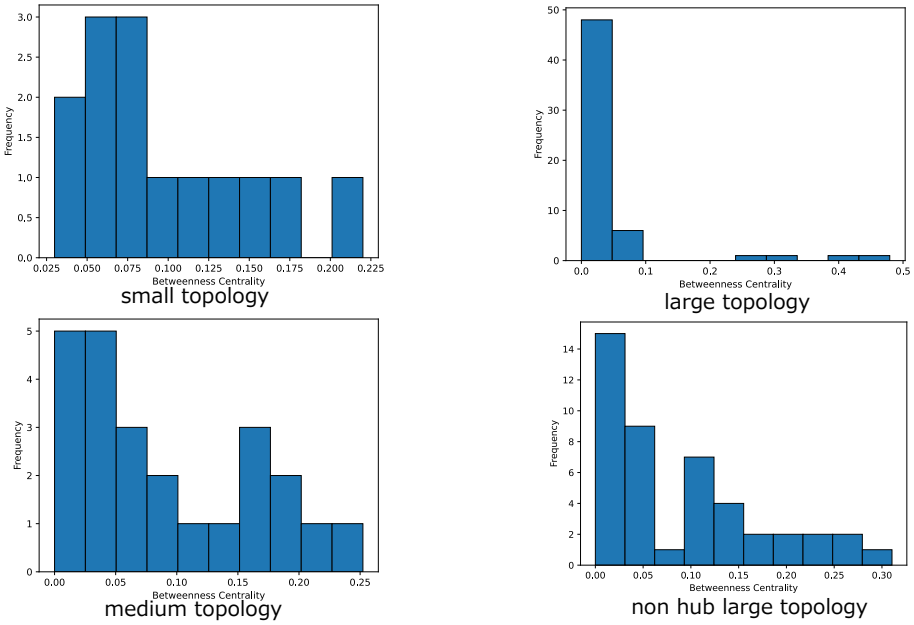

Figure 7: The histogram of the betweenness centrality

Although all distances of links between two adjacency nodes differ, only the number of hops is considered in the simulation experiments. Other experimental parameters are shown in TABLE 1, where the number of packets indicates the number of times during the task that a randomly selected node sends a packet to a different randomly selected node. The default values of α were determined as described in the next section.

Table 1: Experimental Parameters

topology	small	medium	large	non hub large
number of nodes	13	24	58	45
number of links	21	43	87	64
number of packets	50 k, 100 k	50 k, 100 k	50 k, 100 k	50 k, 100 k
default α	3, 4	4, 4	6, 7	8, 10
simulation times	30 times	30 times	30 times	30 times

6.2 Preliminary Experiments

Preliminary experiments were conducted to determine the α parameter used in the proposed algorithm and the threshold of n.packetCount used in Algorithm 1. First, the results of the experiment to determine the parameter α are described, followed by the results of the experiment to determine the threshold.

The appropriate value of α varies depending on the topology and the number of packets, so preliminary experiments were conducted for all topologies and packet numbers, and the appropriate

α obtained from these experiments was used in the main experiment. Packets were transferred from randomly selected nodes to other randomly chosen destination nodes, and the variation in the number of packets transmitted by each node was observed. During the experiments, α was varied from 1 to 10 in the small, medium and large topology, from 1 to 12 in the non hub large topology, and the α value that resulted in the smallest variation in the number of packets transmitted—indicated by the lowest standard deviation—was selected for the main experiment.

The standard deviations of the number of packets transmitted by the nodes were compared, and the results are shown in Figs. 8, 9, 10 and 11.

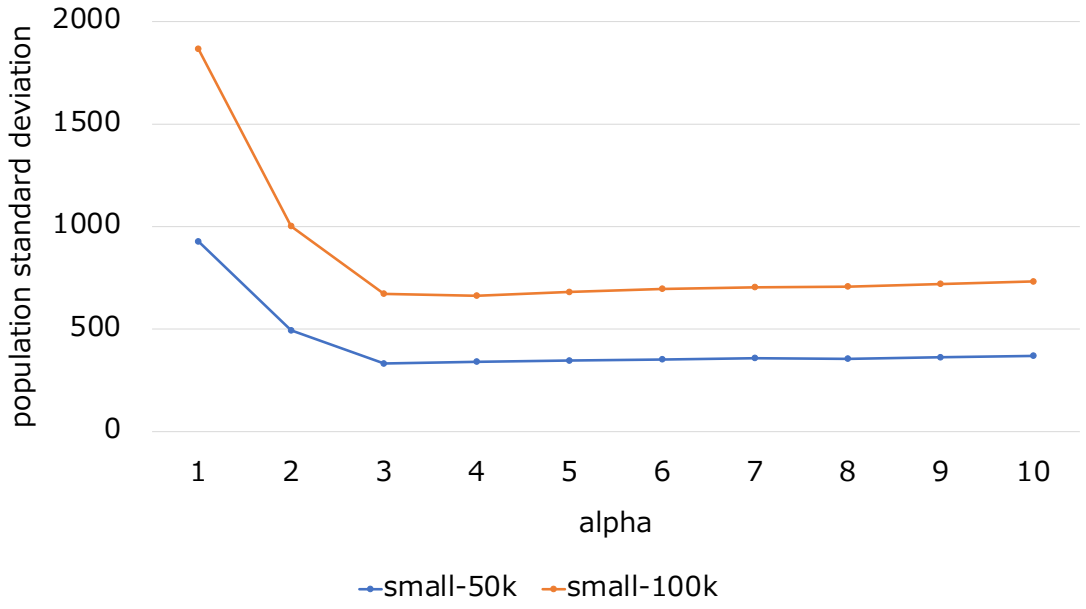

Figure 8: The standard deviation of the number of packets transmitted by the nodes in the small topology

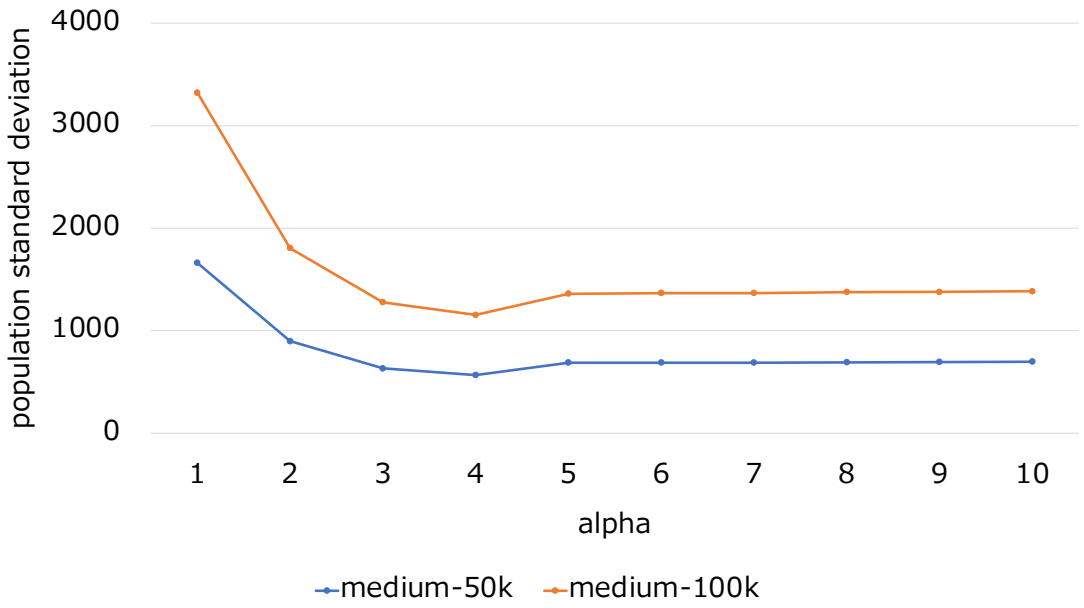

Figure 9: The standard deviation of the number of packets transmitted by the nodes in the medium topology

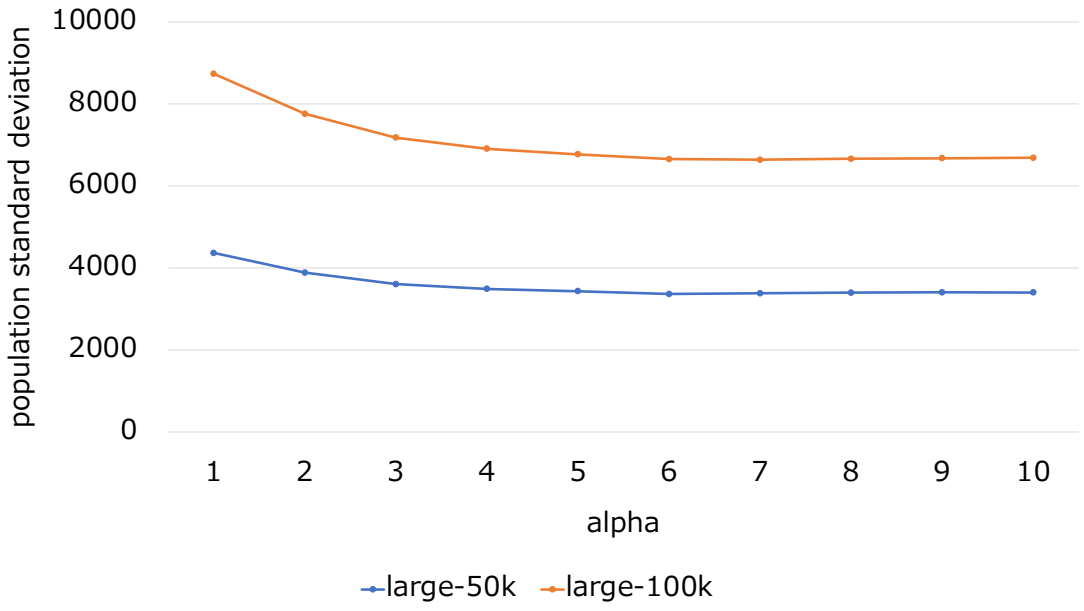

Figure 10: The standard deviation of the number of packets transmitted by the nodes in the large topology

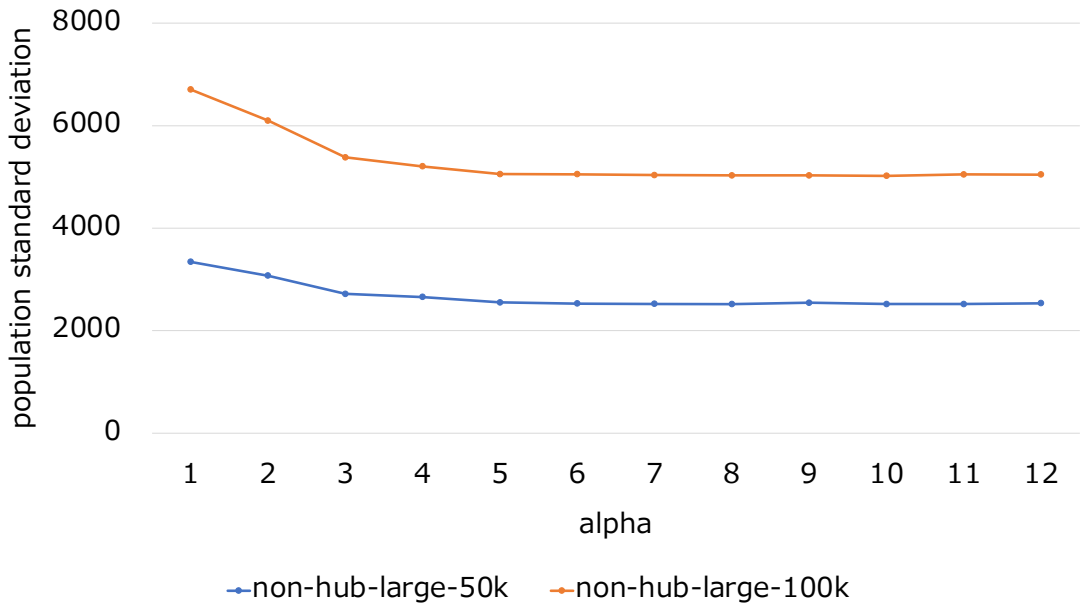

Figure 11: The standard deviation of the number of packets transmitted by the nodes in the non hub large topology

As a result of the experiments, in the small topology, $\alpha = 3$ minimized the population standard deviation for 50k packets, while $\alpha = 4$ was the minimum for 100k packets. In the medium topology, $\alpha = 4$ was the minimum for both 50k and 100k packets. In the large topology, $\alpha = 6$ was the minimum for 50k packets, and $\alpha = 7$ was the minimum for 100k packets. In the non-hub-large topology, $\alpha = 8$ was the minimum for 50k packets, and $\alpha = 10$ was the minimum for 100k packets. Therefore, these values were used in the experiments with the proposed algorithm.

The upper bound value of 1.1 and the lower bound value of 0.9 used in Algorithm 1 were also determined through preliminary experiments. Similar to the determination of parameter α, packets were transmitted from randomly selected nodes to other randomly chosen destination nodes, and the number of packets forwarded by each node was compared. However, this experiment was conducted only for the small and medium topologies. The combination of upper and lower bound values that resulted in the smallest standard deviation of the number of packets forwarded by each node was adopted for the algorithm across all topologies. The experimental results are presented in a table, where the standard deviation values of the number of packets forwarded by each node are rounded to the second decimal place.

Summarizing the experimental results, for 50k packets in the small topology, the best results were obtained with an upper bound of 1.10 and a lower bound of 0.90. For 100k packets in the small topology, the best results were also obtained with an upper bound of 1.10 and a lower bound of 0.90. For 50k packets in the medium topology, the best results were obtained with an upper bound of 1.05 and a lower bound of 0.90, while the second-best results were obtained with an upper bound of 1.10 and a lower bound of 0.90. For 100k packets in the medium topology, the best results were again obtained with an upper bound of 1.10 and a lower bound of 0.90. The most frequently optimal pair was an upper bound of 1.10 and a lower bound of 0.90. Therefore, Algorithm 1 adopts an upper bound value of 1.10 and a lower bound value of 0.90.

Table 2: The combinations of upper and lower bound values for 50k packets in the small topology are as follows.

upper and lower bound values	0.90	0.95
1.05	342.82	559.14
1.10	340.67	562.12

Table 3: The combinations of upper and lower bound values for 100k packets in the small topology are as follows.

upper and lower bound values	0.90	0.95
1.05	679.45	1173.65
1.10	678.31	1179.69

Table 4: The combinations of upper and lower bound values for 50k packets in the medium topology are as follows.

upper and lower bound values	0.90	0.95
1.05	531.58	568.63
1.10	537.98	558.63

Table 5: The combinations of upper and lower bound values for 100k packets in the medium topology are as follows.

upper and lower bound values	0.90	0.95
1.05	1077.24	1080.84
1.10	1063.35	1072.71

6.3 Analysis of Results

This subsection describes experiments where the parameter α, determined as described in the previous subsection, was used to compare the proposed algorithm with the shortest-path algorithm.

Box plots depicting the distribution of the number of packets transmitted by each node are presented: Fig. 12 shows the results for 50,000 and 100,000 packets in the small topology; Fig. 13 represents the same simulations for 50,000 and 100,000 packets in the medium topology, respectively; Fig. 14 represents for 50,000 and 100,000 packets in the large topology, respectively.

In both the small and medium topologies, the proposed algorithm shows significantly less variability compared to the shortest-path algorithm, regardless of the packet count.

On the other hand, in the large topology, the variation in the number of packets forwarded by each node is larger compared to the small and medium topologies. However, while the median of the shortest-path algorithm is positioned near the first quartile with many values concentrated below the median, the proposed algorithm exhibits a more even distribution, with a wider gap between the median and the first and third quartiles. This indicates that even in the large topology, the proposed algorithm results in less variation in the number of packets forwarded per node compared to the shortest-path algorithm. The reason for the larger variation in the number of packets forwarded by each node in the large topology, compared to the small and medium topologies, is characterized by a

small number of nodes having high betweenness centrality (ref. Fig. 7) Nodes with high betweenness centrality are also referred to as hub nodes. Hub nodes tend to relay packets more frequently than other nodes, and in a topology with many hub nodes, there exist flows that lack alternative routes. As a result, hub nodes handle a higher number of packet transmissions, increasing the overall variation.

To further investigate this, we conducted experiments using a non-hub topology, which contains fewer hub nodes than the medium topology but has a larger number of nodes. The results are presented in Fig. 15. The proposed algorithm clearly reduces the variation among nodes compared to the shortest-path algorithm. For 50k packets, the standard deviation for the proposed algorithm is 2508.26, whereas for the shortest-path algorithm, it is 3369.30, demonstrating a significant difference.

Based on these results, it is important to use a topology with fewer hub nodes to effectively utilize the proposed algorithm. By employing a topology with fewer hub nodes, network resources can be utilized more equitably under any conditions.

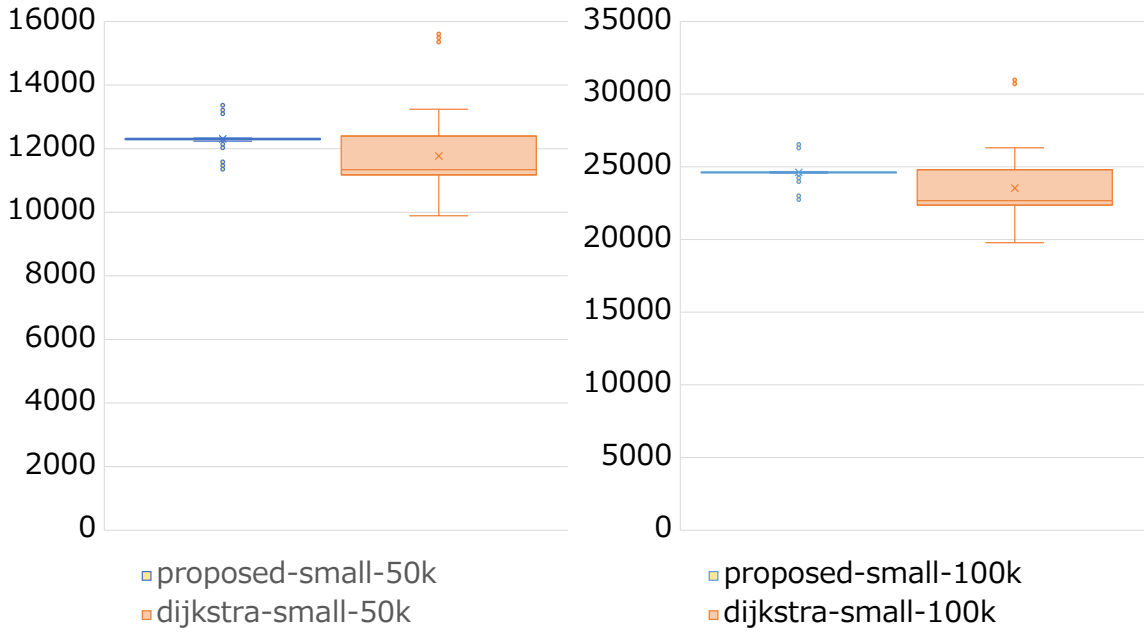

Figure 12: The box plot for the average number of packets forwarded by each node in the small topology (with 50 k, 100 k packets)

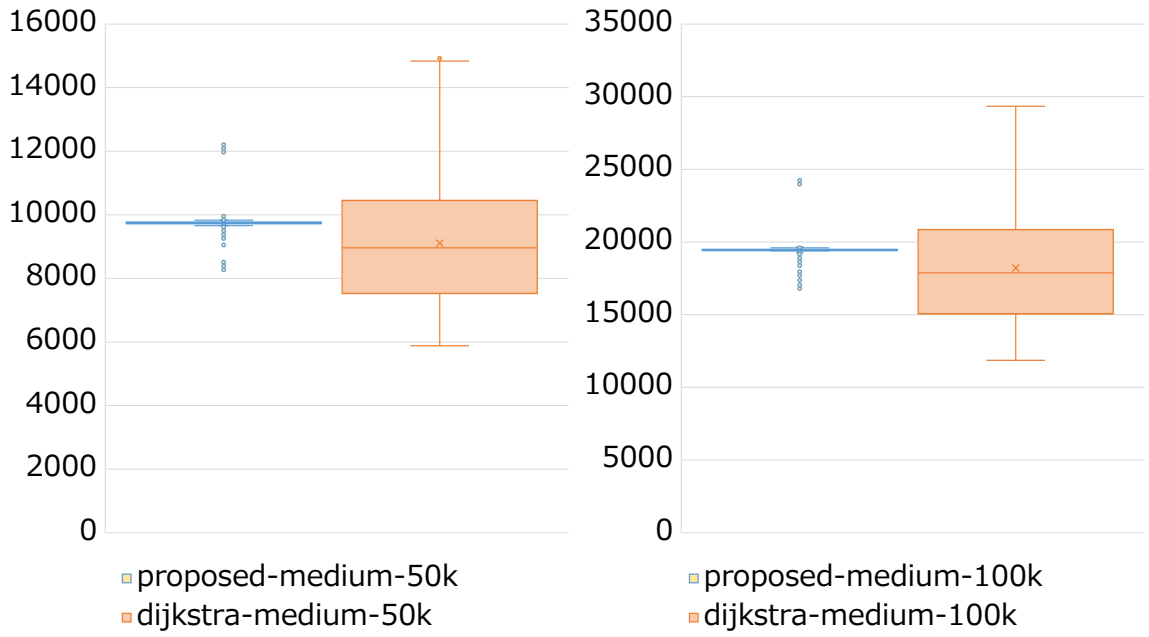

Figure 13: The box plot for the average number of packets forwarded by each node in the medium topology (with 50 k, 100 k packets)

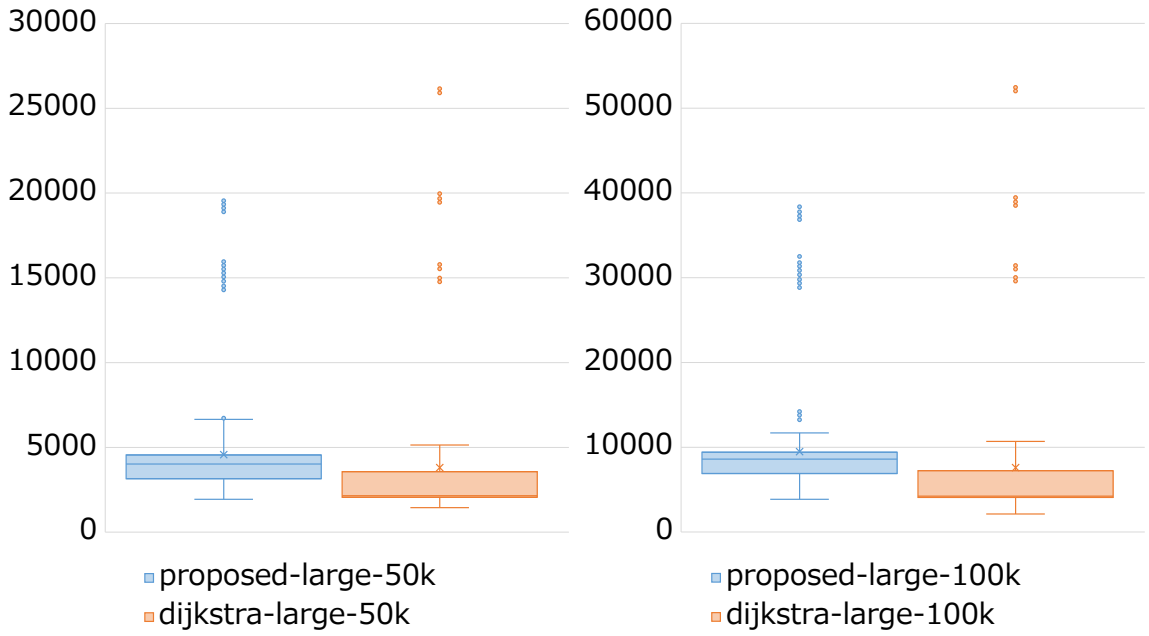

Figure 14: The box plot for the average number of packets forwarded by each node in the large topology (with 50 k, 100 k packets)

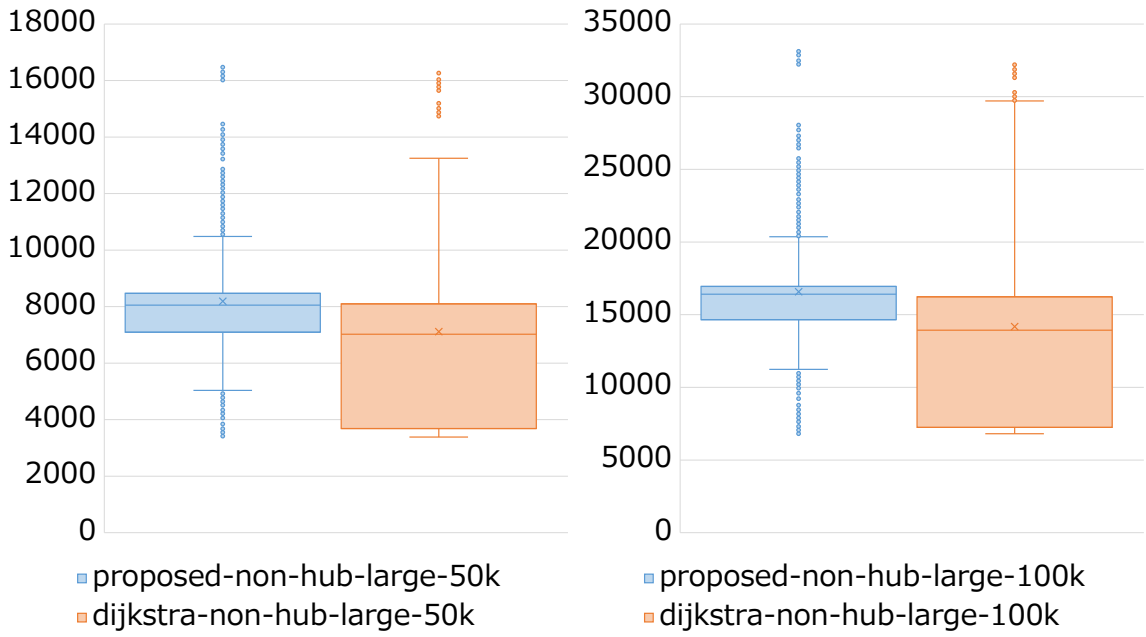

Figure 15: The box plot for the average number of packets forwarded by each node in the non hub large topology (with 50 k, 100 k packets)

On the other hand, the number of hops to transmit a packet may increase due to flow distribution. Figs. 16 depict the average number of hops and the 95% confidence interval with 50k and 100k packets for both the proposed algorithm and the shortest-path algorithm in the small, medium, large and non-hub-large topologies.

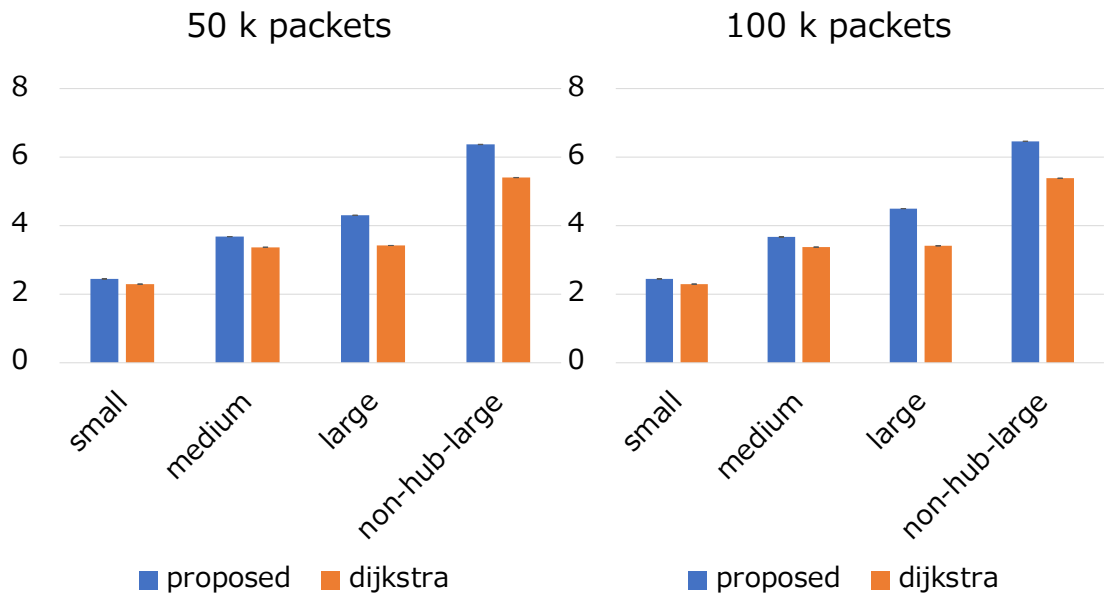

Figure 16: The average number of hops of packets in the topologies (with 50, 100 k packets)

The experimental results show no significant difference in the average number of hops required for packet delivery. However, in the case of the large topology and non-hub-large topology, the proposed algorithm shows approximately 1.1 more hops on average, i.e., packets need to travel approximately one additional hop. This is because, compared to other two topologies, the large topology and non-hub-large topology has fewer mesh-like sections and many nodes have only one adjacent node, which may result in an extra hop when plotting a route.

7 Conclusion

This paper proposed a flow distribution algorithm utilizing Segment Routing (SR) for Software-Defined Networking (SDN). Simulation experiments showed that the proposed algorithm effectively distributes flows across the network at the cost of less than one extra hop. Additionally, a topology with fewer hub nodes may effectively utilize the proposed algorithm.

However, the algorithm does not limit the number of labels in the SR label list, which can lead issues such as communication overhead due to an excessive number of labels [14] and takes no account of the limitation on the number of labels that MPLS devices can process, known as the Maximum Stack Depth (MSD) [12]. Future research will focus on a more practical algorithm with constraints on the number of labels.

References

[1] Priyanka Kamboj, Sujata Pal, Samaresh Bera, and Sudip Misra. QoS-Aware Multipath Routing in Software-Defined Networks. *IEEE Transactions on Network Science and Engineering*,

10(2):723–732, 2023.

[2] Yufeng Xin and Yifei Wang. Partitioning Traffic Engineering in Software Defined Wide Area Networks. In *2023 14th International Conference on Information and Communication Technology Convergence (ICTC)*, pages 596–601, 2023.

[3] Sugam Agarwal, Murali Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *2013 Proceedings IEEE INFOCOM*, pages 2211–2219, 2013.

[4] The Open Networking Foundation. *OpenFlow Switch Specification Version 1.5.1.* The Open Networking Foundation, 2015.

[5] Marcelo Pizzutti and Alberto E. Schaeffer-Filho. Adaptive Multipath Routing based on Hybrid Data and Control Plane Operation. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 730–738, 2019.

[6] Rajesh Narayanan, Saikrishna Kotha, Geng Lin, Aimal Khan, Sajjad Rizvi, Wajeeha Javed, Hassan Khan, and Syed Ali Khayam. Macroflows and Microflows: Enabling Rapid Network Innovation through a Split SDN Data Plane. In *2012 European Workshop on Software Defined Networking*, pages 79–84, 2012.

[7] Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. On the effect of forwarding table size on SDN network utilization. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1734–1742, 2014.

[8] Ziyong Li and Yuxiang Hu. PASR: An Efficient Flow Forwarding Scheme Based on Segment Routing in Software-Defined Networking. *IEEE Access*, 8:10907–10914, 2020.

[9] Momoka Mizuno and Shigetomo Kimura. A Flow Distribution Algorithm with Segment Routing for Software-Defined Networking. In *2024 Twelfth International Symposium on Computing and Networking Workshops (CANDARW)*, pages 15–21, 2024.

[10] Zahraa N. Abdullah, Imtiaz Ahmad, and Iftekhar Hussain. Segment Routing in Software Defined Networks: A Survey. *IEEE Communications Surveys & Tutorials*, 21(1):464–486, 2019.

[11] A. Giorgetti, A. Sgambelluri, F. Paolucci, F. Cugini, and P. Castoldi. Segment routing for effective recovery and multi-domain traffic engineering. *Journal of Optical Communications and Networking*, 9(2):A223–A232, 2017.

[12] Olivier Dugeon, Rabah Guedrez, Samer Lahoud, and Géraldine Texier. Demonstration of Segment Routing with SDN based label stack optimization. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 143–145, 2017.

[13] Farah Chahlaoui, Hamza Dahmouni, and Hassan El Alami. Multipath-routing based load-balancing in SDN networks. In *2022 5th Conference on Cloud and Internet of Things (CIoT)*, pages 180–185, 2022.

[14] Junjie Pang, Gaochao Xu, and Xiaodong Fu. SDN-Based Data Center Networking With Collaboration of Multipath TCP and Segment Routing. *IEEE Access*, 5:9764–9773, 2017.

[15] Gangxiang Shen and Rodney S. Tucker. Energy-Minimized Design for IP Over WDM Networks. *Journal of Optical Communications and Networking*, 1(1):176–186, 2009.

[16] Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.