Noname manuscript No.
(will be inserted by the editor)

Single Tag Scheme for Segment Routing in Software-Defined Network

Nattapong Kitsuwan · Eiji Oki · Takashi Kurimoto · Shigeo Urushidani

Received: date / Accepted: date

Abstract This paper proposes a scheme to reduce a size of a packet header for a segment routing (SR) scheme in a software-defined network (SDN). The SR scheme inserts a segment identification (SID) list into the packet header to indicate a path for the source-destination pair of the packet. The path can be split into different segments to suit the service requirement and the segments are carried by the SID-list whose length increases with the number of segments. This also increases the packet overhead, and an additional packet is needed if the packet length exceeds the maximum transmission unit (MTU). Moreover, it may not be possible to implement SR in SDN due to the limited number of stacked labels provided by the switch vendor. In the proposed scheme, the SID-list is replaced by a single tag to indicate a node edge, called a swapping node. The tag is replaced by a new tag at the swapping node. With this scheme, the size of SID-list is fixed and does not vary with the number of segments, and no additional packets are required. A mathematic model to balance the number of flow entries in each swapping node is introduced by minimizing the maximum number of flow entries in each swapping node over the network. We implement the proposed scheme on the transmission-Japan science information network (SINET5) and demonstrate confirms its functionality.

N. Kitsuwan
The Department of Computer and Network Engineering, The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu-shi, Tokyo 182-8585, Japan.
E. Oki
Graduate School of Informatics, Kyoto University, Yoshida-honmachi, Sakyo-ku, Kyoto 606-8501, Japan.
T. Kurimoto · S. Urushidani
National Institute of Informatics 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan.
E-mail: kitsuwan@uec.ac.jp

Keywords Software-defined network · Segment routing · Multi-protocol label switching.

1 Introduction

Traffic engineering is an approach to optimize network resources and to facilitate trustworthy network operations. It proceeds by predicting or analyzing of the traffic flowing in the network. Unfortunately, network protocols based on the shortest path algorithm, such as interior gateway protocols (IGPs), are not able to overcome the traffic congestion problem. End-to-end approaches were proposed to resolve this problem.

Multi-protocol label switching (MPLS) [1] adopts the end-to-end approach. MPLS allows packets to be transmitted across layer-2. All the subsequent routing switches perform packet forwarding based only on the MPLS label, which is a number to identify a group of IP packets which are to be forwarded in the same manner, over the same path, and with the same forwarding treatment. Each packet is given an MPLS label at ingress nodes. Nodes along the end-to-end path look for the MPLS label, instead of IP header, and process the packets based on the label. An egress node removes the label and forwards the original IP packet to the destination. Label Distribution Protocol (LDP) and Resource Reservation Protocol (RSVP) are signal protocols that implement the label exchanging needed between nodes in the network. Using these signal protocols raises the operating cost of MPLS.

Segment routing (SR) was introduced as an approach to provide flexible scalable and granular control to MPLS-traffic engineer (MPLS-TE) networks [2]- [4]. It removes the signal protocols in the MPLS control plane to simplify the network, as the MPLS data plane remains

static. SR implements MPLS without creating additional tunnels. Packet forwarding is achieved through the use of segments, which represent a list of instructions. In SR, only the ingress label edge routers need keep per-service state information. State management requirements from the midpoint (label switch routers) and tail end (egress label edge routers) are removed. This allows SR to scale significantly better than RSVP-TE while providing most of the same functions [5].

In the SR network, segments represent subpaths that a router can combine to form a complete route to a network destination. Each segment has an identifier, called a segment identifier (SID), that is shared by the entire network. SR leverages the existing MPLS data plane. SIDs are equivalent to labels in MPLS. A path is denoted by a stack of SIDs, called an SID-list. The SID-list equals the label stack in the MPLS architecture [6]. The given path is divided into several segments. An SID in the SID-list indicates the next hop. At each hop, the next hop is indicated by the top SID in the SID-list. A packet travels to the next hop using an IGP, i.e. an open shortest path first (OSPF) or intermediate system to intermediate system (IS-IS). The top SID is removed from the SID-list when the top SID matches the router ID.

Software-defined networking (SDN) is a network technology aimed at making the network as flexible and as rapid as the virtualized server and storage infrastructure of the modern data center [7]. SDN strips the control plane from switches and moves it to the centralized controller. The controller makes decisions from its global view of the network. The administrators can control and manage the network by using software on the centralized controller.

SR is an SDN technology whose packet-forwarding mechanism serves as an alternative to OpenFlow. In OpenFlow, the controller distributes flow entries to nodes along the path of a source-destination pair. Instead of pushing the flow entries to all nodes on the path, SR encodes the forwarding path as a loose source route in an MPLS header and insert it into a packet only at the ingress router. It is possible that the route of a source-destination pair will have a large number of segments and more segments means MPLS stack size becomes larger. The maximum size of the MPLS stack that a router can handle depends on the vendor. Some SDN switches may not able to support large MPLS stacks. For example, a Juniper switch [9, 10] can handle up to five segments, while a Pica8 switch supports only two segments [11]. Therefore, a route that contains more than three segments cannot be realized by a Pica8 switch. This conflicts with the SDN concept that the architecture and control be vendor-independent. Moreover, adding more segments extends the packet length until it exceeds the maximum transmission unit (MTU). This would necessitate the use of a new packet to carry the data that exceeds the MTU. Therefore, network overhead increases to the detriment of user data.

The preferred path routing (PPR), which aims to mitigate the MTU and data plane processing issues from the SR scheme, dedicates one MPLS label for the strict explicit path or two MPLS labels for the loose explicit path for each routing [12]. The total number of MPLS labels required for PPR is $V+C$, where V is the number of nodes in the network and C is the number of requested pair of source and destination. In case of full mesh requests, the total number of MPLS labels is $V+V(V-1)$. The number of MPLS labels may not be enough to support all requests in a large network size since it is limited.

This paper proposes a scheme that employs the single stack approach of MPLS to reduce packet header size in the SR scheme. In this scheme, a unique ID, an MPLS label, is assigned to each node. The path computation element (PCE), which can be implemented in a dedicated server, computes a path for a source-destination pair if it is requested by a user or a network manager [13–16]. The path is divided into several segments as in the SR scheme. In each segment, the MPLS label is used to guide the packet to the edge of the segment, called a swapping node. At the swapping node, the MPLS label us replaced with the next label. The new MPLS label specifies the next swapping node on the requested path. The packet travels from swapping node to swapping node on the shortest path. With this scheme, packet header size is greatly minimized as each packet holds only one MPLS label. Moreover, as the maximum transmission unit (MTU) is never exceeded by routing information, no additional packets, are needed. PPR [12] achieves the same explicit routing as the proposed scheme. The proposed scheme is different from PPR as follows. The proposed scheme does not use a dedicated MPLS tag for requested routing. The required number of MPLS tags is V. The number of MPLS stack for both the strict explicit path and the loose explicit path are only one.

The processing load of the swapping node is problematic and must be considered. We introduce a mathematic model to balance the number of flow entries in each swapping node over the network by minimizing the maximum number of flow entries in each swapping node. The proposed scheme is implemented on the science information network of SINET5, and a demonstration confirms its functionality.

The paper is organized as follows. Section 2 describes in detail the operation mechanism of SR. In

Section 3, the proposed single tag scheme is elucidated. Section 4 introduces a mathematic model to balance the number of flow entries in each swapping node over the network. Section 5 illustrates the performance of the proposed scheme and discusses the results. Section 6 demonstrates of proposal as implemented on a transmission-Japan network. Finally, section 7 provides our conclusions.

2 Segment routing

The SR scheme is used to replace the MPLS-TE network, so we dispense with the RSVP protocol in the control plane and the need to maintain tunnel states at each node in the data plane of the network. There are two data plane applications for SR: MPLS and IPv6. A segment is represented as a label in MPLS, and by the IPv6 routing extension header in IPv6. SR eliminates the need for label distribution protocols, such as LDP and RSVP as well as the border gateway protocol (BGP). Their functionality is realized by segments within link state IGP protocols. A route from a source to a destination is divided into several segments. In each segment, a route from the ingress node to the egress node of that segment is determined by using the shortest path method. In the SR scheme, each node and link in the network is labeled by a unique identifier, a 32 bit integer called Segment IDentifier (SID). There are basically two types of SID [19]. The first, node-SID, identifies a specific node in the network. The node reads the node-SID from the packet header and forwards the packet to the corresponding node where the route is determined by equal-cost multi-path (ECMP) routing [8]. The second, adjacency-SID, specifies a link between two adjacent nodes. The node forwards the packet to a neighbor node through the corresponding link. The set of SIDs, called SID-list, is pushed at the ingress node of the first segment.

Figure 1 illustrates an example of packet routing in the SR scheme. Source $H1$, which connects to node A, sends a packet over a grid topology to destination $H2$ connected to node M. When an IP packet from $H1$ arrives at node A, node A sends a *packet_in* to the controller to ask for instructions. The controller requests a route from PCE. PCE determines the route for this traffic as $ABCGHLPONM$. The PCE determines an MPLS stack for this request as $\{C, G, H, P, M\}$. The MPLS stack is sent from the controller to node A to push it as an SID-list to the packet. The packet travels through nodes as in the SID-list to $H2$. At each node that is listed in the SID-list, the top MPLS stack is popped.

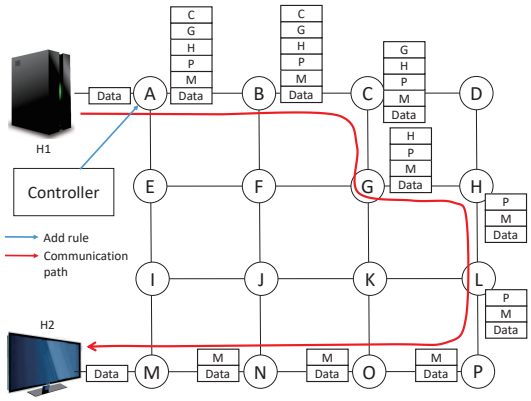


Fig. 1 Example of SR scheme

Node *A*

Match	Action
MPLS=A	Pop, Resubmit
MPLS=B	Out:3
MPLS=C	Out:3
MPLS=D	Out:3
MPLS=E	Out:2
MPLS=F	Out:2
MPLS=G	Out:2
MPLS=H	Out:2
MPLS=I	Out:2
MPLS=J	Out:2
MPLS=K	Out:2
MPLS=L	Out:2
MPLS=M	Out:2
MPLS=N	Out:2
MPLS=O	Out:2
MPLS=P	Out:2
IP=H1	Out:1

Node *B*

Match	Action
MPLS=A	Out:1
MPLS=B	Pop, Resubmit
MPLS=C	Out:3
MPLS=D	Out:3
MPLS=E	Out:2
MPLS=F	Out:2
MPLS=G	Out:2
MPLS=H	Out:2
MPLS=I	Out:2
MPLS=J	Out:2
MPLS=K	Out:2
MPLS=L	Out:2
MPLS=M	Out:2
MPLS=N	Out:2
MPLS=O	Out:2
MPLS=P	Out:2

Node *C*

Match	Action
MPLS=A	Out:1
MPLS=B	Out:1
MPLS=C	Pop, Resubmit
MPLS=D	Out:3
MPLS=E	Out:2
MPLS=F	Out:2
MPLS=G	Out:2
MPLS=H	Out:2
MPLS=I	Out:2
MPLS=J	Out:2
MPLS=K	Out:2
MPLS=L	Out:2
MPLS=M	Out:2
MPLS=N	Out:2
MPLS=O	Out:2
MPLS=P	Out:2

Node *M*

Match	Action
MPLS=A	Out:1
MPLS=B	Out:1
MPLS=C	Out:1
MPLS=D	Out:1
MPLS=E	Out:1
MPLS=F	Out:1
MPLS=G	Out:1
MPLS=H	Out:1
MPLS=I	Out:1
MPLS=J	Out:1
MPLS=K	Out:1
MPLS=L	Out:1
MPLS=M	Pop, Resubmit
MPLS=N	Out:3
MPLS=O	Out:3
MPLS=P	Out:3
IP=H2	Out:1

(a) Initial flow entries

Add flow

Match	Action
IP=H2	Push_MPLS:M,P,H,G,C, Out:3

(b) Flow entry after receiving packet

Fig. 2 Flow entries in SR scheme

Figure 2 shows the flow entries in each node in the SR scheme. The initial flow entries are listed in Fig. 2(a), and the flow entry received from the controller after node A receives the packet is shown in Fig. 2(b). For the initial flow table, an output port is specified to each destination node. If the packet that the MPLS label is the node ID itself, the top MPLS label is popped and the packet is reentered into the same table. As regards the flow from the controller, if the destination IP address of the packet is matched, the MPLS stack is added to the packet header and the packet is forwarded to the next node via the desired output port.

3 Proposed single tag scheme

The single tag scheme adopts one of the concepts of the SR scheme, which is removing the signal protocol from the MPLS architecture, while avoiding the problem of

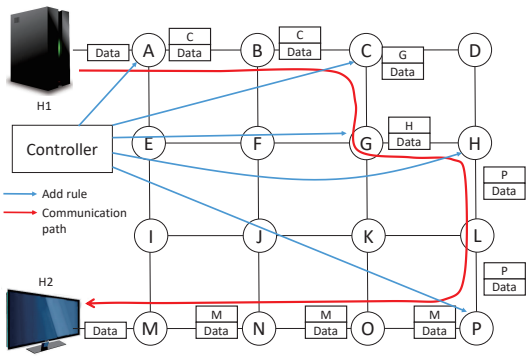


Fig. 3 Example of single tag scheme

Node	Match	Action
A	IP=H2	Push_MPLS:C, Out:3
C	IP=H2	Push_MPLS:G, Out:4
G	IP=H2	Push_MPLS:H, Out:3
H	IP=H2	Push_MPLS:P, Out:4
P	IP=H2	Push_MPLS:M, Out:1

Fig. 4 Flow entries of single tag scheme

large SID-lists. A route for a source-destination pair is divided into several segments, as in the SR scheme. The single tag scheme uses only one MPLS tag to guide a packet from the source to the destination along the determined route. The tag guides the packet from the ingress node to the egress node (swapping node) of the same segment using the shortest path. The tag, which indicates the next swapping node, is pushed at the ingress node of the first segment. The tag is replaced by a new tag at each swapping node. It should be noted that the set of swapping nodes is the same as the set of SID-list in the SR scheme. It is popped at an egress node of the last segment of the route. Nodes that are not swapping nodes pass the packet to a neighbor node without changing the tag. The number of nodes that need an additional flow entry is the number of swapping nodes - 1 (for the last swapping node) + 1 (for the ingress node of the first segment). It is superior to the SR scheme in that flow entries are needed only at the ingress node of the first segment.

There are two types of flow entries in the single tag scheme: initial and additional flow entries. The initial flow entry has a similar structure to the SR scheme, which consists of a match field that matches the MPLS tag or IP addresses of hosts connecting to the local node, and an action that specifies the output port or popping of the MPLS tag. The additional flow entries matches the destination IP address or the request. The action pushes an MPLS tag into the header and specifies an output port. In case of a swapping node, a swapping process is as follows. An MPLS tag is pop, and the destination IP address is matched. If the destination IP address does not belong to any host at the local node, the node pushes a new MPLS to specify an egress node of the next segment.

Figure 3 illustrates an operation example of the proposed single tag scheme. The initial flow entries are configured at every node as in Fig. 2. The packet is sent from host $H1$ to host $H2$. When the packet reaches node A, A sends *packet_in* to the controller. The controller determines the red path to host $H2$. There are five swapping nodes on the red path, which are nodes C, G, H, P, and M. However, the controller sends *flow_mod* messages to add flow entries at node A and the swapping nodes except for the last one. It should be noted that a flow entry to pop the MPLS tag already exists at every swapping node due to the initial flow. Therefore, a flow entry for popping the MPLS tag is not required at each swapping node. The additional flow entries are shown in Fig. 4. Node A pushes MPLS tag C to the packet header and forwards the packet to node B. The packet passes through node B to node C. At node C, tag C is pop and tag G is pushed before forwarding to node G. Node G pops tag G and pushes tag H before sending the packet to node H. At node H, tag H is pop and tag P is pushed before sending the packet to node P via node L. Node P pops tag P and pushes tag M and forwards the packet to node M via nodes O and N. When the packet arrives at node M, the MPLS tag is popped, and node M sends the packet to host $H2$.

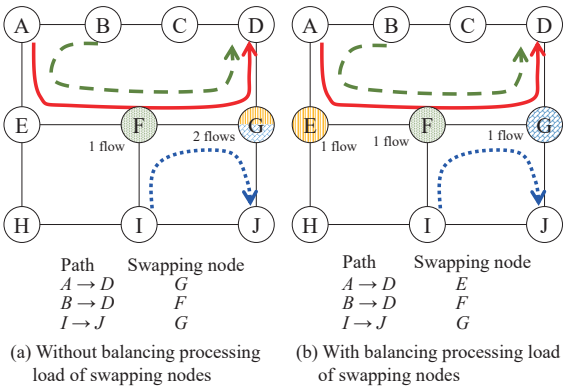
Several notes on the proposed scheme are described below. The proposed scheme does not downgrade the topology complex. The requested explicit routing for each pair of source and destination nodes is achieved by the proposed scheme. In a network that considers per-node and per-link as important factors, these factors are incorporated in the requested explicit routing. There is a case that packets for end-to-end are transferred more than seven hops. For example, the Japanese photonic network (JPN48) topology [17, 18], which is a network across Japan, the southernmost node needs more than 10 hops to reach the northernmost node. The proposed scheme achieves a lower size of a packet header for every number of hops, compared to the SR scheme.

4 Problem of allocating flow entries in swapping node

4.1 Overview

We consider the problem of allocating flow entries in the swapping nodes. A set of paths, each of which has a route including source and destination pair, is given. A set of shortest paths from any node to all other nodes in the network is given. Our objective is to allocate flow entries to the swapping nodes so as to minimize the maximum number of flow entries in each swapping node, which will balance the processing loads of the swapping nodes.

Figure 5 provides two examples of allocating flow entries in swapping nodes. There are three requested paths, $A \to D$ on path $AEFGD$, $B \to D$ on path $BAEFGD$, and $I \to J$ on path $IFGJ$. We consider two approaches. The first minimizes the number of swapping nodes for each requested path, but ignores the processing loads of swapping nodes. We introduce a minimum flow per request (MFPR) algorithm for this approach. Beginning with the end node of the previous segment, or the source node, the MFPR algorithm follows the requested path until it deviates from the shortest path to the destination and selects the next node on the requested path as a swapping node; this node defines the end of the current segment and the beginning of the next segment. Based on this approach, node F is set as a swapping node of path $B \to D$ and node G is set as a swapping node of paths $A \to D$ and $I \to J$, as shown in Fig. 5(a). The second approach minimizes the maximum number of flow entries in each swapping node to balancing the processing loads of swapping nodes. In Fig. 5(b), the swapping nodes for those three paths are balanced by setting nodes F, E, and G as swapping nodes for paths $B \to D$, $A \to D$, and $I \to J$, respectively.



(a) Without balancing processing load of swapping nodes

(b) With balancing processing load of swapping nodes

Fig. 5 Examples of allocating flow entries to swapping nodes.

4.2 Integer linear programming problem

We consider directed graph $G(V, E)$, where V is a set of nodes and E is a set of links in the network. Let $(i, j) \in E$ be a directed link from node $i \in V$ to node $j \in V$. Let $p \in P$ be a path, where P is the set of paths. $p \in P$ is specified by a set of nodes $r_p = \{r(p,h) | r(p,h) \in V, p \in P, h \in [0, H_p]\}$, where $r(p,h) \in V$ is the $h(>0)$th transit node on $p \in P$, $r(p,0)$ is the source node of $p \in P$, $r(p,H_p)$ is the destination node of $p \in P$, and H_p is the number of hops for $p \in P$. Since the shortest path from one node to another node in the network can be computed in advance, each node knows the next hop node to the destination node. Let f_{vik} be a flag to express the information of the next hop node. f_{vik} is set to one if $k \in V : (v,k) \in E$ is the next hop node at $v \in V$ to destination $i \in V \backslash \{v\}$, and zero otherwise. Let x_{pv} be a binary decision variable that is set to one if $p \in P$ uses $v \in V$ as a swapping node, and zero otherwise. Let $y_{phh'}$ is a binary decision variable that is set to one if there is no swapping node from $(h+1)$th hop to $(h'-1)$th hop of $p \in P$, where $h \in [0, H_p - 1]$ and $h' \in [h+2, H_p]$, and zero otherwise. Let $z_{phh'}$ be a binary decision variable that indicates the product of binary decision variables $x_{pr(p,h')}$ and $y_{phh'}$, where $h \in [0, H_p - 1]$ and $h' \in [h+2, H_p]$. $z_{phh'}$ is set to one if both $x_{pr(p,h')} = 1$ and $y_{phh'} = 1$, and zero otherwise. Let S_v be the number of flow entries at $v \in V$. Let S be the maximum number of flow entries in each swapping node over the network, or $S = \max_{v \in V} S_v$. To balance the processing loads of swapping nodes, we formulate below the optimization problem of minimizing the maximum number of flow entries in the swapping nodes as an integer linear programming (ILP) problem.

$$\min\ S + \epsilon \sum_{v \in V} S_v \tag{1a}$$

$$\text{s.t. } x_{pr(p,h+1)} + \sum_{h' \geq h+2}^{H_p - 1} f_{r(p,h)r(p,h')r(p,h+1)} z_{phh'} + f_{r(p,h)r(p,H_p)r(p,h+1)} y_{phH_p} \geq 1, \quad \forall p \in P, h \in [0, H_p - 1] \tag{1b}$$

$$y_{phh'} \geq 1 - \sum_{i \geq h+1}^{h'-1} x_{pr(p,i)}, \forall p \in P, \quad h \in [0, H_p - 1], h' \in [h+2, H_p] \tag{1c}$$

$$y_{phh'} \leq 1 - \frac{1}{|V|} \sum_{i \geq h+1}^{h'-1} x_{pr(p,i)}, \forall p \in P, \quad h \in [0, H_p - 1], h' \in [h+2, H_p] \tag{1d}$$

$$z_{phh'} \leq x_{pr(p,h')}, \forall p \in P,$$

$$h \in [0, H_p - 1], h' \in [h + 2, H_p - 1] \quad (1e)$$

$$z_{phh'} \leq y_{phh'}, \forall p \in P, \\ h \in [0, H_p - 1], h' \in [h + 2, H_p - 1] \quad (1f)$$

$$z_{phh'} \geq x_{pr(p,h')} + y_{phh'} - 1, \forall p \in P, \\ h \in [0, H_p - 1], h' \in [h + 2, H_p - 1] \quad (1g)$$

$$\sum_{p \in P} x_{pv} = S_v, \forall v \in V \quad (1h)$$

$$S_v \leq S, \forall v \in V \quad (1i)$$

$$x_{pv} \in \{0, 1\}, \forall p \in P, v \in V \quad (1j)$$

$$y_{phh'} \in \{0, 1\}, \forall p \in P, \\ h \in [0, H_p - 1], h' \in [h + 2, H_p] \quad (1k)$$

$$z_{phh'} \in \{0, 1\}, \forall p \in P, \\ h \in [0, H_p - 1], h' \in [h + 2, H_p - 1] \quad (1l)$$

Eq. (1a) indicates the objective function, where the first term, S, which is the maximum number of flow entries in the swapping nodes, is minimized first, and is followed by minimization of the second term, $\sum_{v \in V} S_v$, which is the total number of flow entries in the network. ϵ is set to a sufficiently small value that is less than $\frac{1}{|P||V|}$ so that the value of second term in Eq. (1a) cannot affect the minimization of the first term. Eq. (1b) indicates that a packet outgoing from $r(p, h), h \in [0, H_p - 1]$, must be transmitted to the next hop on $p \in P$ by using a flow entry or by shortest path routing. If the first term of the left hand side of Eq. (1b) is one, $r(p, h + 1), h \in [0, H_p - 1]$, has a flow entry for $p \in P$ as a swapping node. If the second term of left hand side of Eq. (1b) is one or more, the next hop node from $r(p, h), h \in [0, H_p - 1]$, for $p \in P$ is equal to the next hop node on the shortest path from $r(p, h), h \in [0, H_p - 1]$ to at least one of $r(p, h'), h' \in [h + 2, H_p - 1]$ that has a flow entry as a swapping node under the condition that there is no swapping node from $(h + 1)$th hop to $(h' - 1)$th hop of $p \in P$[1]. If the third term of left hand side of Eq. (1b) is set to one, the next hop node from $r(p, h), h \in [0, H_p - 1]$, for $p \in P$ is equal to the next hop node on the shortest path from $r(p, h), h \in [0, H_p - 1]$, to $r(p, H_p)$ under the condition of $y_{phH_p} = 1$. Eqs. (1c) and (1d) express that $y_{phh'}$ is set to one if there is no swapping node from $(h + 1)$th hop to $(h' - 1)$th hop of $p \in P$, and zero otherwise. Eqs. (1e)-(1g) express that $z_{phh'}$ is the product of binary variables $x_{pr(p,h')}$ and $y_{phh'}$, or $z_{phh'} = x_{pr(p,h')} \cdot y_{phh'}$. Eq. (1h) indicates that S_v is the sum of x_{pv} over $v \in V$. Eq. (1i) expresses $S = \max_{v \in V} S_v$ in cooperation with Eq. (1a) in a linear form. Eqs. (1j)-(1l) indicate that x_{pv}, $y_{phh'}$, and $z_{phh'}$ are binary decision variables.

[1] In the second term of left hand side of Eq. (1b), $h = h' + 1$ is not included in the summation, This is because the first and third terms consider the case of $h = h' + 1$.

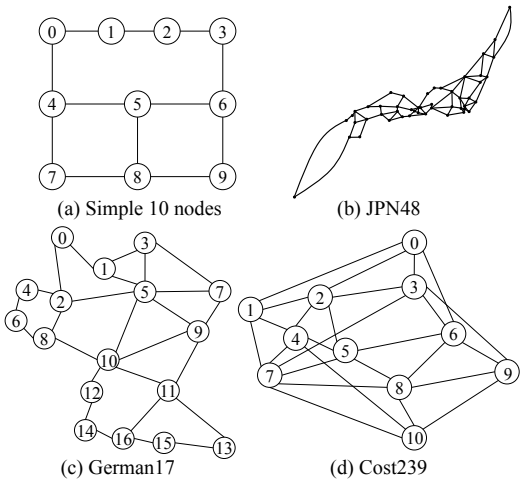


Fig. 6 Topologies for simulation.

Table 1 Result for simple 10 node topology

Avg. number of hops per request: 3.951

	Avg. max. number of flows in each node	Avg. total number of flows	Avg. computation time [sec]
ILP	10.707	40.993	0.056
MFPR	10.725	40.938	7.463×10^{-6}

Table 2 Result for JPN48 topology

Avg. number of hops per request: 8.405

	Avg. max. number of flows in each node	Avg. total number of flows	Avg. computation time [sec]
ILP	113.147	2549.035	10.485
MFPR	131.425	2547.297	10.440×10^{-6}

Table 3 Result for German17 topology

Average number of hops per request: 4.538

	Avg. max. number of flows in each node	Avg. total number of flows	Avg. computation time [sec]
ILP	21.050	181.991	0.121
MFPR	21.226	181.935	8.435×10^{-6}

5 Performance evaluation

We evaluated the number of flow entries in a swapping node when ILP was used to balance the processing load of swapping nodes over the network. We then analyzed the ratio of packet header and the number of extra packets for the proposed scheme and the SR scheme.

5.1 Balancing number of flow entries in swapping node using ILP

The topologies in Fig. 6 were used to evaluate the performance of two approaches: with/without balancing

Table 4 Result for Cost239 topology

Average number of hops per request: 3.219			
	Avg. max. number of flows in each node	Avg. total number of flows	Avg. computation time [sec]
ILP	12.943	88.246	0.010
MFPR	13.424	88.022	6.995×10^{-6}

the processing load of swapping nodes. The shortest path is given. We randomly generate a number of requests and their paths. The simulation was run 1,000 times. We evaluated the maximum number of flow entries in each swapping node, the total number of flow entries in the network, and the computation time.

Tables 1-4 show that the average maximum number of flows in each node is lower with ILP than with MFPR. The average total number of flows is higher with ILP than with MFPR. This is because more swapping nodes are needed for path control while reducing the maximum number of flow entries in each node. The average computation time of MFPR depends on the number of requested paths, $|P|$, and the number of hops in those requests; the maximum number of hops is $|V|-1$. The complexity of MFPR is $\mathcal{O}(|P||V|)$. The computation time of ILP, as shown in Tables 1-4, can be accepted in our examined topologies. As the network size increases, the computation time of ILP increases. In case that a shorter computation time is required than that of ILP, the introduced MFPR algorithm can be an option to be implemented for the proposed scheme. The average maximum number of flows in each node of MFPR is at most 16%, for JPN48, higher than that of ILP, in our examined topologies.

We note where the ILP problem is computed when the ILP approach is adopted. The ILP computation element can be implemented in a computationally powerful server different from a controller; the controller sends the request of ILP computation to the server and the server returns the solution to the controller.

5.2 Analysis of number of packets

The maximum transmission unit (MTU) is set to 1,500 bytes as default. The format of the Ethernet packet considered here is shown in Fig. 7. Ethernet frame size is 24 bytes, IPv4 header without any options is 20 bytes, TCP header without any options is 20 bytes. The format of an MPLS label consists of 20 bits for the label number field, 3 bits for EXP field, 1 bit for S field, and 8 bits for TTL field. In total, a single MPLS label occupies 32 bits (4 bytes). The length of a packet header carrying an SID-list with n labels is $24+4n+20+20 = 64+4n$ bytes.

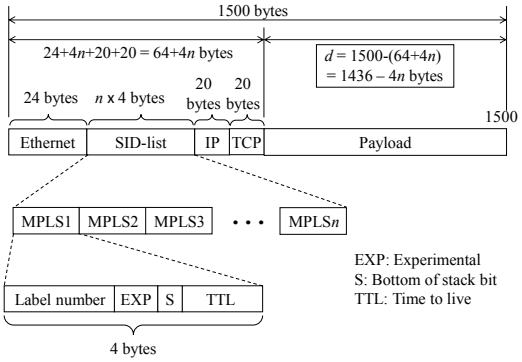


Fig. 7 Ethernet packet format with SID-list.

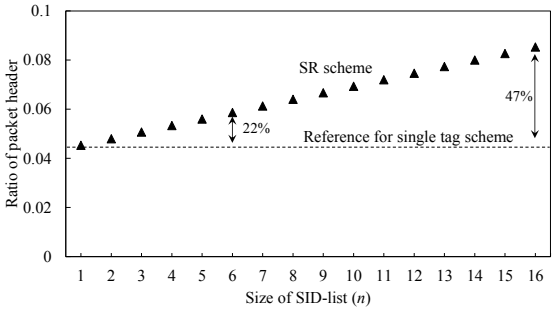


Fig. 8 Ratio of packet header with different SID-list lengths.

Figure 8 plots the packet header ratio, which is the length of the packet header divided by the packet length. The plots in the SR scheme are obtained by $\frac{64+4n}{1500}$. The dash line indicates the constant value of 0.04533, which is obtained by $\frac{64+4}{1500}$, of the proposed scheme. The packet header ratio in the SR scheme linearly increases with SID-list length, but in the proposed single tag scheme the ratio is independent of SID-list length. When SID-list holds 16 entries, the proposed single tag scheme improves the packet header ratio by 47%, which is derived as follows

$$\left(\frac{\frac{(64+(4\times16))}{1500} - \frac{64+4}{1500}}{\frac{64+(4\times16)}{1500}} \right) \times 100 = 46.875\% \approx 47\%. \quad (2)$$

It should be noted that the ratio of packet header of the single tag scheme has an affect if the packet size exceeds the MTU size due to the SID-list. If the SID-list does not make the packet size exceed the MTU size, the ratio of packet header between the SR and single tag schemes are the same.

The number of swapping nodes increases with the network size. The original data is split into chunks. Each chunk is put into the payload of a packet. If the packet size exceeds the MTU size due to the SID-list, data chunk size (in bytes), d, must be reduced to fit the MTU. d is computed by the MTU size without the

packet header.

$$d = (1500 - (64 + 4n)) = 1436 - 4n, \quad (3)$$

where n is the number of tags in the SID-list. n in the proposed scheme is the same as that in the MPLS-TE, i.e., one. A slight increasing in a packet header size may trigger the use of additional packets. The number of required packets, P, for payload length (in bytes), D, is:

$$P = \frac{D}{d}. \quad (4)$$

In the proposed scheme, n is one so that d is

$$d = (1500 - (64 + 4)) = 1432 \text{ bytes}. \quad (5)$$

The required number of packets in the proposed scheme, P_{prop}, is obtained by substituting Eq. (5) into Eq. (4) as follows

$$P_{\text{prop}} = \frac{D}{1432} \text{ packets}. \quad (6)$$

The required number of packets in the SR scheme, P_{SR}, increases with n, which is obtained by substituting Eq. (3) into Eq. (4) as follows

$$P_{\text{SR}} = \frac{D}{(1436 - 4n)} \text{ packets}. \quad (7)$$

The number of additional packets, E, in the SR scheme compared to the proposed scheme is obtained from the difference between Eq. (6) and Eq. (7), which is given by:

$$E = P_{\text{SR}} - P_{\text{prop}}. \quad (8)$$

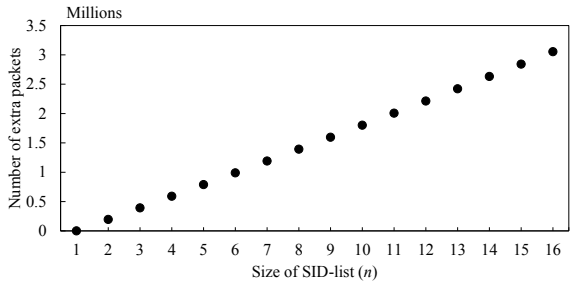


Fig. 9 Number of additional packets needed in SR scheme compared to single tag scheme.

We compare the SR and proposed single tag schemes in terms of the number of additional packets. Figure 9 plots the number of additional packets when 100 Gbytes of data is sent, which is obtained by Eq. (8). The number of additional packets increases with SID-list size in the SR scheme, whereas the single tag scheme needs no additional packets. If the SID-list has six entries,

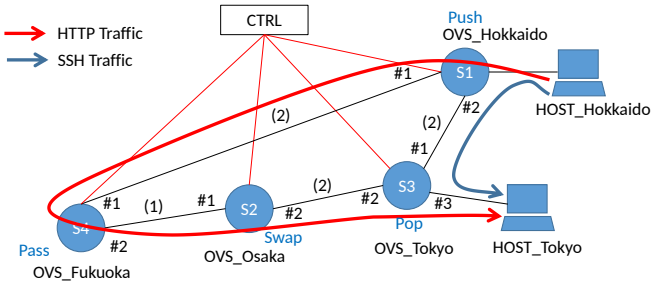


Fig. 10 Topology for demonstration.

the SR schemes needed approximately one million additional packets than the single tag scheme. If the SID-list has 16 entries, the SR scheme needs three million more packets than the single tag scheme. This inefficiency in packet transmission is a significant problem.

6 Demonstration

The proposed scheme was demonstrated on Science Information Network 5 (SINET5) [20], provided by the National Institute of Informatics (NII) [21] to confirm its functionality. Seven virtual machines (VMs) were deployed across Japan. One VM in Tokyo acted as a controller. Two VMs in Hokkaido and Tokyo were hosts. The other four VMs acted as OpenFlow switches, located in Hokkaido, Tokyo, Osaka, and Fukuoka. Open vSwitch version 2.5.2 [22] was installed on each switch. Information of IP address and switch_ID is listed in Table 5 The network topology used in the demonstration is illustrated in Fig. 10. The numbers in brackets represent link cost. The controller ran an MySQL database.

Table 5 Information of VMs for demonstration.

VM name	Location	IP address	switch_ID
HOST_Hokkaido	Hokkaido	192.168.0.34	-
HOST_Tokyo	Tokyo	192.168.0.17	-
OVS_Hokkaido (s1)	Hokkaido	192.168.1.37	101
OVS_Osaka (s2)	Osaka	192.168.1.4	102
OVS_Tokyo (s3)	Tokyo	192.168.1.17	103
OVS_Fukuoka (s4)	Fukuoka	192.168.1.29	104
CTRL	Fukuoka	192.168.1.24	-

The database used in the demonstration had three tables, as shown in Fig. 11. The first table, named "path", kept the information of paths and switches for each source-destination pair and protocol. The second table "port" kept the information of port number in each switch. The third table "sw_detail" stored the datapath id (dpid), and MPLS id number for each switch.

```
mysql> select * from path;
+----+-------------+-------------+-------+-------------+----------+
| id | nw_src      | nw_dst      | proto | path        | set_sw   |
+----+-------------+-------------+-------+-------------+----------+
|  1 | 192.168.0.34 | 192.168.0.17 |    22 | s1,s3       | s1,s3    |
|  2 | 192.168.0.34 | 192.168.0.17 |    80 | s1,s4,s2,s3 | s1,s2,s3 |
+----+-------------+-------------+-------+-------------+----------+
```

```
mysql> select * from port;
+----+------+----------+--------+
| id | sw   | neighbor | output |
+----+------+----------+--------+
|  1 | s1   | s3       |      2 |
|  2 | s1   | s4       |      1 |
|  3 | s1   | host     |      3 |
|  4 | s2   | s3       |      2 |
|  5 | s2   | s4       |      1 |
|  6 | s3   | s1       |      1 |
|  7 | s3   | s2       |      2 |
|  8 | s3   | host     |      3 |
|  9 | s4   | s1       |      1 |
| 10 | s4   | s2       |      2 |
+----+------+----------+--------+
```

```
mysql> select * from sw_detail;
+----+------+------------------+----------+
| id | sw   | dpid             | mpls_id |
+----+------+------------------+----------+
|  1 | s1   | 63866552046920   |     101 |
|  2 | s2   | 108297769848384  |     102 |
|  3 | s3   | 152739915257422  |     103 |
|  4 | s4   | 178844403451471  |     104 |
+----+------+------------------+----------+
```

Fig. 11 Database structure.

We created a scenario in which HOST_Hokkaido transferred data to HOST_Tokyo. Two types of traffic, hypertext transfer protocol (HTTP) and secure shell (SSH), were transmitted along different routes, as shown in Fig. 10. Normally the controller would compute the path between source and destination, based on several parameters such as traffic condition. However, path computation was dropped in the demonstration. Paths between HOST_Hokkaido and HOST_Tokyo with different traffic characteristics were assumed and manually input into the database. HTTP traffic packets were sent from HOST_Hokkaido via OVS_Hokkaido, OVS_Fukuoka, OVS_Osaka, and OVS_Tokyo, to reach HOST_Tokyo. SSH traffic packets were sent from HOST_Hokkaido via OVS_Tokyo to HOST_Tokyo. The initial flow entries in each switch are shown in Fig. 12.

The scenario was performed as follows. HTTP traffic packets were sent from HOST_Hokkaido using the command "curl http://192.168.0.14/demonstration.html". Upon receiving the first HTTP packet, OVS_Hokkaido, OVS_Hokkaido sent OpenFlow *packet_in* to the controller, to determine how to deal with this packet. The controller retrieved the path "$s1, s4, s2, s3$" and the set of switches to be configured, "$s1, s2, s3$", from Table "path", datapath ID and switch ID from Table "sw_detail", and an output port for each switch from Table "port". The controller sent an OFPFlowMod message, which is a flow configuration message, to every switch that needed to be configured by the addition of a flow entry, as shown in Fig. 13. It should be noted that MPLS swapping is used as an action in the demonstration since there is only one pair of source and destination. After switch configuration, the flow entry at $s1$ matched the packet with the destination IP address and destination tcp port. The action pushed MPLS tag 102 to the packet, and specified packet release from port #1, which was connected to OVS_Fukuoka. The flow entry

OVS_Hokkaido

```
root@ovs-hokkaido2:/home/ubuntu/single_tag# ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=88.785s, table=0, n_packets=0, n_bytes=0, idle_age=88,
tcp,nw_dst=192.168.0.34,tp_src=22 actions=output:3
 cookie=0x0, duration=88.773s, table=0, n_packets=0, n_bytes=0, idle_age=88,
tcp,nw_dst=192.168.0.34,tp_src=80 actions=output:3
 cookie=0x0, duration=88.798s, table=0, n_packets=0, n_bytes=0, idle_age=88,
priority=0 actions=CONTROLLER:65535
root@ovs-hokkaido2:/home/ubuntu/single_tag#
```

OVS_Fukuoka

```
root@ovs-fukuoka:/home/ubuntu/single_tag# ovs-ofctl dump-flows s4
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=69.875s, table=0, n_packets=0, n_bytes=0, idle_age=69,
ip,nw_dst=192.168.0.34 actions=output:1
 cookie=0x0, duration=69.781s, table=0, n_packets=0, n_bytes=0, idle_age=69,
mpls,mpls_label=102 actions=output:2
 cookie=0x0, duration=69.634s, table=0, n_packets=0, n_bytes=0, idle_age=69,
priority=0 actions=CONTROLLER:65535
root@ovs-fukuoka:/home/ubuntu/single_tag#
```

OVS_Osaka

```
root@ovs-tokyo2:/home/ubuntu/single_tag# ovs-ofctl dump-flows s3
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=84.769s, table=0, n_packets=0, n_bytes=0, idle_age=84,
tcp,nw_dst=192.168.0.34,tp_src=22 actions=output:1
 cookie=0x0, duration=84.768s, table=0, n_packets=0, n_bytes=0, idle_age=84,
tcp,nw_dst=192.168.0.34,tp_src=80 actions=output:2
 cookie=0x0, duration=84.771s, table=0, n_packets=0, n_bytes=0, idle_age=84,
priority=0 actions=CONTROLLER:65535
root@ovs-tokyo2:/home/ubuntu/single_tag#
```

OVS_Tokyo

```
root@ovs-osaka:/home/ubuntu/single_tag# ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=79.668s, table=0, n_packets=0, n_bytes=0, idle_age=79,
ip,nw_dst=192.168.0.34 actions=output:1
 cookie=0x0, duration=79.656s, table=0, n_packets=0, n_bytes=0, idle_age=79,
priority=0 actions=CONTROLLER:65535
root@ovs-osaka:/home/ubuntu/single_tag#
```

Fig. 12 Initial flow entries in each switch.

OVS_Hokkaido

```
 cookie=0x0, duration=8.748s, table=0, n_packets=6, n_bytes=498, idle_age=7,
tcp,nw_dst=192.168.0.17,tp_dst=80 actions=push_mpls:0x8847,load:0x66-
>OXM_OF_MPLS_LABEL[],output:1
```

OVS_Osaka

```
 cookie=0x0, duration=11.620s, table=0, n_packets=6, n_bytes=522, idle_age=10,
mpls,mpls_label=103 actions=pop_mpls:0x0800,output:3
```

OVS_Tokyo

```
 cookie=0x0, duration=13.447s, table=0, n_packets=6, n_bytes=522, idle_age=12,
mpls,mpls_label=102 actions=load:0x67->OXM_OF_MPLS_LABEL[],output:2
```

Fig. 13 Additional flow entries sent from controller to the switches handling HTTP traffic.

at $s2$ (OVS_Osaka) matched MPLS 102 of the packet and swapped it with MPLS 103 before sending it to output port #2, which was connected to OVS_Tokyo. The flow entry at $s3$ (OVS_Tokyo) matched MPLS 103 of the packet and popped the MPLS tag before sending it to output port #3, which was connected to HOST_Tokyo. It should be noted that no additional flow entry is needed in $s4$ (OVS_Fukuoka) since $s4$ is on the shortest path between $s1$ and $s3$. The packet is forwarded using the initial flow entries in $s4$. After the flow entries were sent to the corresponding switches, subsequent packets traversed the path $s1 \rightarrow s4 \rightarrow s2 \rightarrow s3$ to HOST_Tokyo using the above flow entries. Upon successfully receiving an HTTP packet, HOST_Tokyo returns a response message to HOST_Hokkaido along the same route.

OVS_Hokkaido

```
 cookie=0x0, duration=22.540s, table=0, n_packets=41, n_bytes=5331, idle_age=7,
tcp,nw_dst=192.168.0.17,tp_dst=22 actions=push_mpls:0x8847,load:0x67-
>OXM_OF_MPLS_LABEL[],output:2
```

OVS_Tokyo

```
 cookie=0x0, duration=27.136s, table=0, n_packets=41, n_bytes=5495, idle_age=12,
mpls,mpls_label=103 actions=pop_mpls:0x0800,output:3
```

Fig. 14 Additional flow entries sent from controller to each corresponding switch for SSH traffic.

The initial SSH traffic packet from HOST_Hokkaido was sent to HOST_Tokyo by using the command "ssh ubuntu@192.168.0.17". The packet was sent to the controller as an OpenFlow *packet_in*, as for the HTTP traffic. The controller retrieved the path "$s1, s3$", the set of switches that need to be configured ("$s1, s3$"), datapath ID, switch ID, and output port of each switch from the database. The controller sent an OFPFlowMod message, which is a flow configuration message, to add flow entries to $s1$ and $s3$, as shown in Fig. 14. At $s1$, the destination IP address and destination tcp port of the packet were matched. The action pushed MPLS tag 103 to the packet, and released the packet to port #2, which was connected to OVS_Tokyo. The flow entry at $s3$ matched MPLS 103 of the packet and the MPLS tag was popped before sending it to output port #3, which was connected to HOST_Tokyo. Subsequent SSH packets traversed the path $s1 \rightarrow s3$ to HOST_Tokyo using the above flow entries. Upon receiving an SSH packet, HOST_Tokyo returned a prompt for password input back to HOST_Hokkaido along the same route. Roundtrip time of SSH packets was measured as the period from when the packet left HOST_Hokkaido to when acknowledgement was received by the same host. The first packet took 1.01 seconds. This time includes link propagation delay, processing at the controller, and flow setup at corresponding switches on the path. Once switches flow entries were added, later packets took only 10 ms.

Packets were captured at the output ports of switches to confirm the functioning of the proposed scheme. For the HTTP traffic, we captured the transmission packets at port #1 of OVS_Hokkaido, port #2 of OVS_Fukuoka, port #2 of OVS_Osaka, and port #3 of OVS_Tokyo, as shown in Fig. 15. Packets for the flow configuration messages and *packet_in* were captured at the controller, as shown in Fig. 16. This data confirmed that *packet_in* was received from OVS_Hokkaido, and the correct flow configuration messages were sent to OVS_Hokkaido, OVS_Osaka, and OVS_Tokyo. We also confirmed that the MPLS label 102 was added to the packet at OVS_Hokkaido. The packet with MPLS label 102 was passed through OVS_Fukuoka. MPLS label 102 was swapped to MPLS label 103 at OVS_Osaka. MPLS label 103 was popped at OVS_Tokyo. For the SSH traffic, the transmission packets were captured at OVS_Hokkaido and OVS_Tokyo, as shown in Fig. 17. The flow configuration messages and *packet_in* were captured at the controller, as shown in Fig. 18. This data confirmed that *packet_in* was received from OVS_Hokkaido, and that the correct flow configuration messages were sent to OVS_Hokkaido and OVS_Tokyo. MPLS label 103 was added to the packet at OVS_Hokkaido, and the MPLS label 103 was popped at OVS_Tokyo. Roundtrip times of the SSH packets were measured. The first packet took 1.09 seconds, subsequent packets took 31.8 ms.

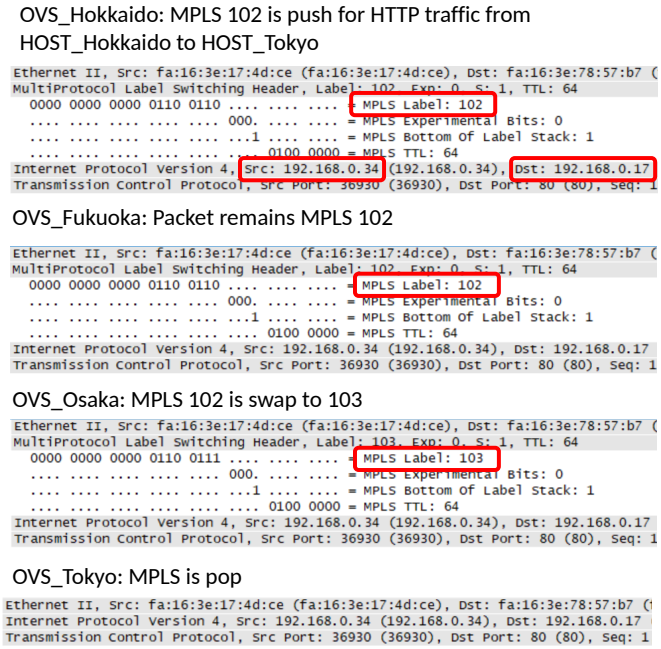


Fig. 15 Handling of HTTP packets at each switch.

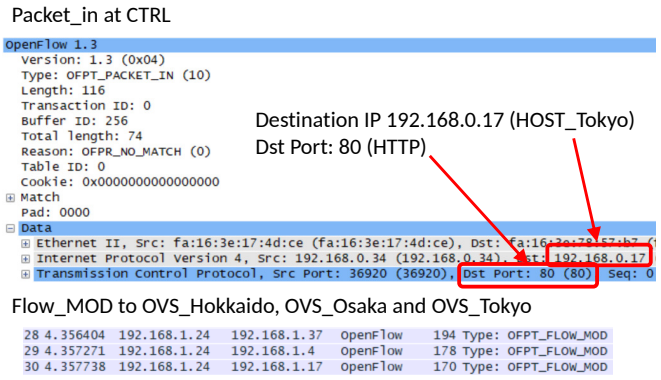


Fig. 16 HTTP packets captured at controller.

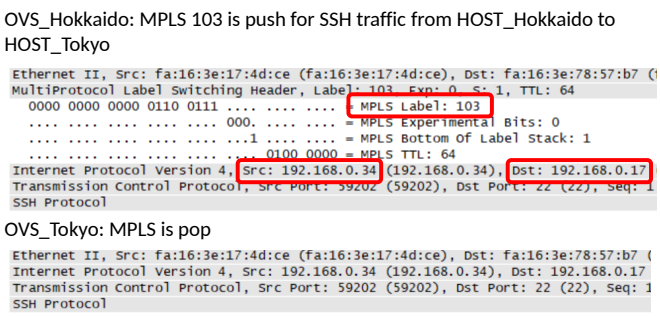


Fig. 17 Handling of SSH packets at each switch.

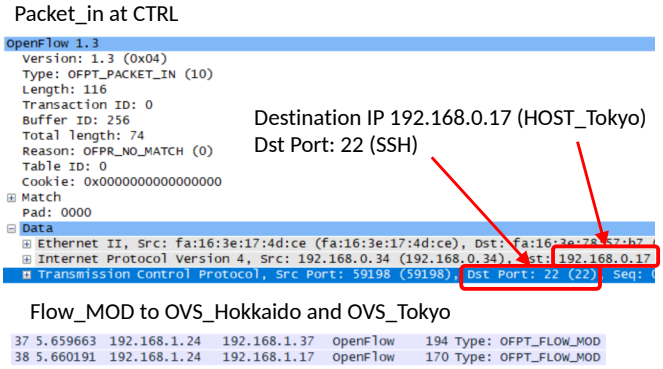


Fig. 18 SSH packets captured at controller.

7 Conclusions

This paper proposed a scheme to reduce a size of a packet header for segment routing architecture in SDN. The proposal uses only single tags to specify given paths instead of using SID-lists holding multiple SIDs. The tag is swapped with a new tag at an edge node of each segment, called a swapping node. An ILP formulation was introduced to minimize the maximum number of flows that need to be held in each swapping node. Analyses showed that the ILP formulation needs fewer flow entries in each swapping node, on average, than MFPR, but a greater total number of flow entries in the network, on average. Our analyses indicate that the proposal reduces the packet header by 47 percent compared to SR scheme when the SID-list holds 16 entries. The proposed scheme was implemented on SINET5 and used to realize HTTP and SSH transmission. All functions of the proposed scheme were confirmed.

References

1. Rosen, E., Viswanathan, A., & R. Callon (2001). Multiprotocol Label Switching Architecture. RFC 3031.
2. Giorgetti, A., Sgambelluri, A., Paolucci, F., Cugini, F., & Castoldi, P. (2017). Segment routing for effective recovery and multi-domain traffic engineering. In *IEEE/OSA Journal of Optical Communications and Networking, 9*(2), A223-A232. https://doi.org/10.1364/JOCN.9.00A223
3. Cianfrani, A., Listanti, M., & Polverini, M. (2017). Incremental Deployment of Segment Routing Into an ISP Network: a Traffic Engineering Perspective. In *IEEE/ACM Transactions on Networking, 25*(5), 3146-3160. https://doi.org/10.1109/TNET.2017.2731419
4. Schüller, T., Aschenbruck, N., Chimani, M., Horneffer, M. & Schnitter, S. (2017). Traffic engineering using segment routing and considering requirements of a carrier IP network. In *IFIP Networking Conference (IFIP Networking) and Workshops*.
5. Cianfrani, A., Listanti, M., & Polverini, M. (2016). Translating Traffic Engineering outcome into Segment Routing paths: The Encoding problem. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 245-250).
6. Filsfils, C., Previdi, S., Bashandy, A., Decraene, B., Litkowski, S., & Shakir, R. (2017). Segment Routing with MPLS data plane. IETF draft-ietf-spring-segment-routing-mpls-10.
7. Kitsuwan, N., McGettrick, S., Slyne, F., Payne, D.B., & Ruffini, M. (2015). Independent transient plane design for protection in OpenFlow-based networks. In *IEEE/OSA Journal of Optical Communications and Networking, 7*(4), 264-275. https://doi.org/10.1364/JOCN.7.000264.
8. (2014). IEEE Standard for Local and metropolitan area networks - Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks - Amendment 22: Equal Cost Multiple Path (ECMP). IEEE Std 802.1Qbp-2014 (Amendment to IEEE Std 802.1Q-2011). https://doi.org/10.1109/IEEESTD.2014.6783684
9. Configuring the Maximum Number of MPLS Labels. Retrieved February 28, 2018 from https://www.juniper.net/documentation/en_US/junos/topics/task/configuration/interfaces-mpls-maximum-labels.html.
10. Maximum-labels. Retrieved February 28, 2018 from https://www.juniper.net/documentation/en_US/junos/topics/reference/configuration-statement/maximum-labels-edit-interfaces-unit-family-mpls.html.
11. Configuring MPLS. Retrieved February 28, 2018 from http://www.pica8.com/wp-content/uploads/2015/09/v2.9/html/ovs-configuration-guide/#8195477.
12. Chunduri, U., Clemm, A., & Li, R. (2018). Preferred Path Routing - A Next-Generation Routing Framework Beyond Segment Routing. In *IEEE Globbal Communications Conference (GLOBECOM)*.
13. Farrel, A., Vasseur, J.P., & Ash, J. (2006). A path computation element (PCE)-based architecture. RFC 4655.
14. Lee, Y., Le Roux JL., King, D., & Oki, E. (2009). Path computation element communication protocol (PCECP) requirements and protocol extensions in support of global concurrent optimization. RFC 5557.
15. Oki, E., Takeda, T., Farrel, A., & Zhang, F. (2017). Extensions to the Path Computation Element Communication Protocol (PCEP) for Inter-Layer MPLS and GMPLS Traffic Engineering. RFC 8282.
16. Oki, E., Le Roux, JL., & Farrel, A. (2009). Framework for PCE-Based Inter-Layer MPLS and GMPLS Traffic Engineering. RFC 5623.
17. Japan Photonic Network Model. Retrieved September 04, 2019 from http://www.ieice.org/cs/pn/jpn/jpnm.html.
18. Arakawa, S., Sakano, T., Tsukishima, Y., Hasegawa, H., Tsuritani, T., Hirota, Y., & Tode, H. (2013). Topological characteristic of japan photonic network model. In *IEICE Technical Report, 113(91)*, 7-12.

19. Filsfils, C., Nainar, N. K., Pignataro, C., Cardona, J. C., & Francois, P. (2015). The Segment Routing Architecture. In *IEEE Global Communications Conference (GLOBECOM)*.
20. Science Information NETwork 5. Retrieved February 28, 2018 from https://www.sinet.ad.jp/en/top-en.
21. National Institute of Informatics. Retrieved February 28, 2018 from http://www.nii.ac.jp/en/.
22. Open vSwitch. Retrieved February 28, 2018 from http://openvswitch.org/.

Nattapong Kitsuwan received B.E. and M.E. degrees in electrical engineering (telecommunication) from Mahanakorn University of Technology, King Mongkut's Institute of Technology, Ladkrabang, Thailand, and a Ph.D. in information and communication engineering from the University of Electro-Communications, Japan, in 2000, 2004, and 2011, respectively. From 2002 to 2003, he was an exchange student at the University of Electro-Communications, Tokyo, Japan, where he performed research regarding optical packet switching. From 2003 to 2005, he was working for ROHM Integrated Semiconductor, Thailand, as an Information System Expert. He was a post-doctoral researcher at the University of Electro-Communications from 2011 to 2013. He worked as a researcher for the Telecommunications Research Centre (CTVR), Trinity College Dublin, Ireland from 2013 to 2015. Currently, he is an assistant professor at the University of Electro-Communications, Tokyo, Japan. His research focuses on optical network technologies, routing protocols, and software-defined networks.

Eiji Oki is a Professor at Kyoto University, Japan. He received the B.E. and M.E. degrees in instrumentation engineering and a Ph.D. degree in electrical engineering from Keio University, Yokohama, Japan, in 1991, 1993, and 1999, respectively. In 1993, he joined Nippon Telegraph and Telephone Corporation (NTT) Communication Switching Laboratories, Tokyo, Japan. He has been researching network design and control, traffic-control methods, and high-speed switching systems. From 2000 to 2001, he was a Visiting Scholar at the Polytechnic Institute of New York University, Brooklyn, New York, where he was involved in designing terabit switch/router systems. He was engaged in researching and developing high-speed optical IP backbone networks with NTT Laboratories. He was with The University of Electro-Communications, Tokyo, Japan from July 2008 to February 2017. He joined Kyoto University, Japan, in March 2017. He has been active in the standardization of the path computation element (PCE) and GMPLS in the IETF. He wrote more than ten IETF RFCs. Prof. Oki was the recipient of several prestigious awards, including the 1998 Switching System Research Award and the 1999 Excellent Paper Award presented by IEICE, the 2001 Asia-Pacific Outstanding Young Researcher Award presented by IEEE Communications Society for his contributions to broadband network, ATM, and optical IP technologies, the 2010 Telecom System Technology Prize by the Telecommunications Advanced Foundation, IEEE HPSR 2012 Outstanding Paper Award, and IEEE HPSR 2014 Best Paper Award Finalist, First Runner Up. He has authored/coauthored four books, Broadband Packet Switching Technologies, published by John Wiley, New York, in 2001, GMPLS Technologies, published by CRC Press, Boca Raton, FL, in 2005, Advanced Internet Protocols, Services, and Applications, published by Wiley, New York, in 2012, and Linear Programming and Algorithms for Communica- tion Networks, CRC Press, Boca Raton, FL, in 2012. He is a Fellow of IEEE.

Takashi Kurimoto graduated from the Tokyo Institute of Technology, Japan, where he received B.E. and M.E. degrees in applied physics 1992 and 1994, respectively. In 1994, He graduated from Keio University where he received the Ph.D. degree in 2012. He worked for NTT Network Service Systems Laboratories and NTT east plant planning department from 1994 to 2014. He has been engaged in researching the switching technology for high-speed computer networks and deployment of the next generation network. He moved to NII in 2015 and is currently involved in the design and implementation of the Science Information Network (SINET). He received the IEICE Switching System Research Award in 1996.

Shigeo Urushidani is a professor and director at the Research Center for Academic Networks of the National Institute of Informatics (NII), Japan. He is also the director at Cyber Science Infrastructure Development Department of NII. He received B.E. and M.E. degrees from Kobe University in 1983 and 1985, respectively, and received a Ph.D. from the University of Tokyo in 2002. He worked for NTT from 1985 to 2006, where he was engaged in the research and development of ATM, AIN, IP/MPLS, and optical switching systems. He moved to NII in 2006 and is currently involved in the design and implementation of the Science Information Network (SINET), as well as in the research and development on network and system architecture for ultra-high-speed green networks.

Responses to the reviewer' comments

We would like to thank the reviewer for the careful and constructive comments. The paper has been improved as a result. We have addressed every point raised by the reviewer. The revised parts are marked with blue color.

Reviewer

Comment 1

There is no clarification about the equation 2 and 3 still.

Answer 1: We clarified how to obtain these equations, which become Eqs. (3)-(8) in the revised version. We explained how to obtaine every value of these equations at paragraph 3 of section 5.2.

Comment 2

Authors are requested to justify the 47% improvement of packet header ratio. There was no any strong background to justify this improvement.

Answer 2: We derived how to obtain the 47% improvement when $n = 16$ at Eq. (7) as follows: $\left(\frac{\frac{(64+(4\times16))}{1500} - \frac{64+4}{1500}}{\frac{64+(4\times16)}{1500}} \right) \times 100 = 46.875\% \approx 47\%$. We added the background for this improvement at the end of paragraph 2 of section 5.2.

Comment 3

Proposed equations and equations were not correlated.

Answer 3: First, we moved the explanation of Fig. 8 to paragraph 2 of section 5.2 before Eqs. (3)-(8), which are used for Fig. 9, to avoid misunderstanding of the equations and result in Fig. 8. Second, we showed the computation of each variable as an equation, and explained how to derive each equation by substituting from which equation into which equation to show that Eqs. (3)-(8) are correlated.