IEICE TRANS. COMMUN., VOL.E105–B, NO.2 FEBRUARY 2022

PAPER *Special Section on Emerging Communication Technologies in Conjunction with Main Topics of ICETC2020*

# An Efficient Calculation for TI-LFA Rerouting Path*

**Kazuya SUZUKI**[†a)], ***Senior Member***

**SUMMARY** Recently, segment routing, which is a modern forwarding mechanism, and Topology Independent Loop-free Alternate, which is an IP fast-reroute method using segment routing, have been proposed and have begun to be applied to real networks. When a failure occurs in a network, TI-LFA quickly restores packet forwarding without waiting for other nodes to update their routing tables. It does so by using segment routing to forward sections that may cause loops in the rerouting path. However, determining the segment routing sections has a high computational cost because it requires computation for each destination. This paper therefore proposes an algorithm to determine the egress node that is the exit of the segment routing section for all destination nodes with only three shortest-path tree calculations. The evaluation results of the proposed algorithm showed that the average tunnel lengths are at most 2.0 to 2.2 hops regardless of the size of the network. I also showed that the computational complexity of the proposed algorithm is $O(N \log N)$.

***key words:*** *IP routing, segment routing, IP fast-reroute*

## 1. Introduction

The spread of IoT applications has increased the importance of IP networks carrying their data. When a failure occurs in an IP network, nodes (routers) detecting the failure notify other nodes in the network. The nodes receiving the failure notification restore packet forwarding by calculating a route to bypass the failure point. However, packet forwarding stops from failure occurrence until the restoration, so packet loss occurs in communication at the failure point. Various IP fast-reroute methods to solve this problem have been proposed [2]–[8]. These methods reduce downtime by preparing alternative routing tables in advance.

Packets may loop if only the node detecting the failure performs the rerouting operation. Therefore, Loop-free Alternate (LFA) [4], an IP fast-reroute method, determines whether a loop occurs for each destination and reroutes only packets that do not loop. In an IP fast-reroute method for all destinations [7], the node detecting a failure identifies nodes that need to update their routing tables to prevent loops and notifies these nodes of the failure. The nodes receiving the failure notification switch to alternative routing tables prepared in advance. This method requires the introduction of a new control message that notifies the nodes of failure and instructs to switch them to the alternative routing tables. These methods have the benefit of being able to be applied to IP networks without changing the IP forwarding mechanism.

Recently, a new forwarding mechanism called segment routing [9] has been proposed for fast rerouting and traffic engineering, and it has begun to be applied to real networks. In segment routing, nodes forward packets by using segment ID (IP address or MPLS label), which indicates nodes on the path to the destination of a packet. The segment ID is added to packets by a source node or ingress node of a segment routing domain. In segment routing, the path on which packets are forwarded can be explicitly specified by specifying the nodes.

Topology Independent Loop-free Alternate (TI-LFA) [6] is an IP fast-reroute method using segment routing. In TI-LFA, a node detecting a failure uses segment routing to pass packets through nodes that cause the loop. TI-LFA calculates a set of nodes (called Q-Space) that does not include a failed link on the shortest path for a destination node. The method avoids routing loops by calculating a rerouting path to a node in the Q-Space (called an egress node) and forwarding packets on the rerouting path. The Q-Space is calculated using the reverse-shortest-path tree with the destination node as the sink of the tree. The Q-Space needs to be calculated for each destination node in the network, necessitating the calculation of many shortest-path trees, the computational cost of which is high.

To solve this problem, this paper proposes a method to determine the egress node for each destination node in the network by calculating the shortest-path tree only three time at most. The proposed method is based on an idea presented in our previous study [7]. This study has proposed a method in which a node detecting a failure uses only three shortest path tree calculations to determine whether or not the other nodes need to update their routing table. If the node detecting a failure reroutes packets when the other nodes that need to update their routing table does not update them, loops occur and the packets does not reach their destination. This paper proposes a method, which is based on our previous study, to determine the egress node of TI-LFA efficiently.

The rest of this paper is organized as follows. Section 2 briefly surveys related works. In Sect. 3, after describing conditions for updating routing table and a method checking the condition, which are proposed in [7], I describe the proposed method to determine the egress node of TI-LFA efficiently. Section 4 evaluates the proposed algorithm.

Manuscript received February 18, 2021.
Manuscript revised June 14, 2021.
Manuscript publicized August 5, 2021.

[†]The author is with Faculty of Systems Science and Technology, Akita Prefectural University, Yurihonjyo-shi, 015-0055 Japan.

[*]A preliminary version of this paper was published in the Proceedings of International Conference on Emerging Technologies for Communications (ICETC), 2020 [1].

a) E-mail: kazuya-suzuki@akita-pu.ac.jp

DOI: 10.1587/transcom.2021CEP0003

**Table 1** Comparison of IP fast-reroute methods.

| Methods | Forwarding mechanism | Rerouting targets | Additional failure notification |
|---|---|---|---|
| MRC [2] | Original | All | - |
| FIR [3] | Original | All | - |
| LFA [4] | IP | Partial | - |
| rLFA [5] | IP with tunneling | Partial | - |
| TI-LFA [6] | IP with segment routing | All | - |
| SPAR [7] | IP | All | Necessary |

Section 5 describes discussions on the proposed method. Section 6 summarizes our study.

## 2. Related Works

IP fast-reroute methods are technologies to reduce downtime of packet forwarding in the event of failure by quickly switching to alternative routing tables prepared in advance. Early studies on IP fast-reroute [2], [3] use original forwarding mechanisms that differ from those of conventional IP forwarding for quicker failure notification. One such method [2] marks packets that cannot be forwarded due to failure and forwards them using an alternate routing table instead of the normal routing table. Nelakuditi et al. [3] proposed a method that infers the locations of failures from interfaces where packets have been received. Different prepared routing tables are used to forward packets in accordance with the receiving interfaces. Because these methods require changes in the forwarding mechanisms, it is difficult to deploy them in conventional IP networks.

When using the original IP forwarding, the node detecting the failure starts the rerouting operation before the other nodes update their routing tables, so a loop may occur between the node detecting the failure and the other nodes before they update their routing tables. In LFA [4], each node determines whether a loop occurs for each destination in the event of failure and reroutes only packets that do not loop.

Remote LFA (rLFA) [5] is an IP fast-reroute method using tunnels. A previous paper [8] proposed an efficient method for calculating the endpoint of the rerouting tunnel. When using a tunnel, packets that are added to tunnel headers going toward the tunnel endpoint are forwarded in the tunnel. The endpoint node of the tunnel removes the tunnel headers and forwards packets to their original destinations. However, these methods require that packets with tunnel headers be able to reach the tunnel endpoint node without a routing loop. The methods cannot be applied to destinations for which a tunnel endpoint node can not be determined. In other words, these methods may not be able to reroute packets for all destinations in a network. To forward packets to tunnel a endpoint without looping, TI-LFA [6] uses segment routing [9]. Segment routing that explicitly specifies the forwarding path can forward tunnel packets to a tunnel endpoint (called an egress node) without routing loops.

Selective precomputation of alternate routes (SPAR) [7] is an IP fast reroute method for all destinations using an approach different from TI-LFA. In this method, nodes that need to update their own routing tables when failure occurs prepare alternative routing tables for quickly restoring packet forwarding. To achieve this, [7] gave conditions for updating routing tables when failure occurs and a method for efficiently determining whether these conditions are satisfied. Since the proposed method in this paper is based on this study, these conditions and the determining method are explained in detail in 3.2 and 3.3.

Table 1 summarizes the characteristics of various IP fast-reroute methods. Even if there is a rerouting path to a destination, LFA and rLFA may not be able to reroute to that destination, whereas TI-LFA and SPAR can reroute for all destinations. LFA and SPAR are applicable to general IP networks, whereas rLFA and TI-LFA require tunneling and segment routing, respectively. TI-LFA can target all destinations for rerouting; however, TI-LFA requires nodes supporting segment routing.

## 3. Proposed Method

The proposed method is based on conditions required to update the routing table at the time of failure and the method for determining whether these conditions are satisfied, which a previous paper [7] showed. After explaining prerequisites, these conditions and method in 3.1, 3.2 and 3.3, I propose an efficient method for calculating the egress node of a rerouting path using segment routing in 3.4.

### 3.1 Prerequisites

The proposed method is applied to IP networks with dynamic routing by routing protocol such as OSPF [10]. This IP networks are assumed to have at least one path between any two nodes after a failure. The proposed method is not necessary for IP networks where there is no alternative path after a failure. In other words, the proposed method targets IP networks in which a failure does not split the IP network into two parts.

To simplify the discussion, I applied the following prerequisites. These prerequisites will be discussed in detail in 5.4.

1. All links are to be point-to-point and bi-directional.
2. There is only one link failure that occurs in network.
3. There is only one shortest path between any two nodes in network.

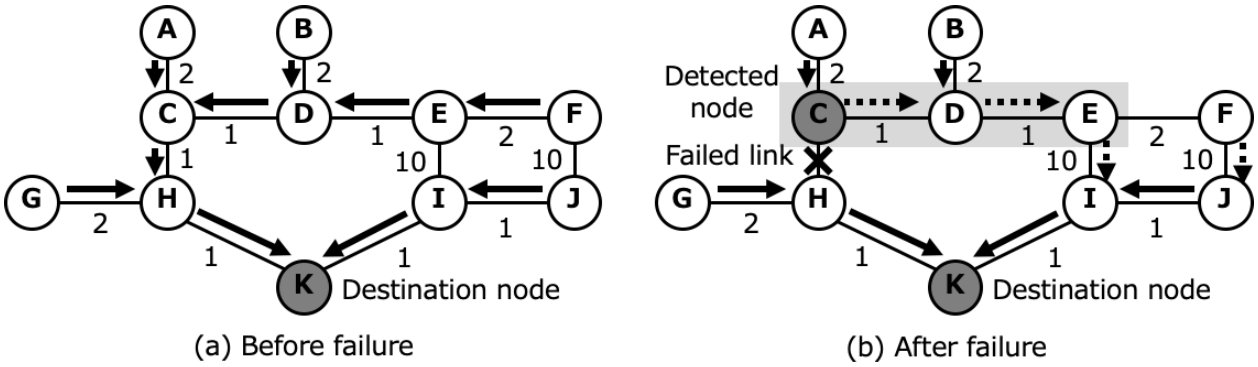


**Fig. 1** Nodes requiring route update.

### 3.2 Conditions for Updating Routing Table

I explain the conditions for determining whether nodes need to update their routing tables when a link failure occurs. Figure 1(a) and (b) show before and after the failure of link between nodes C and H. The arrows in Fig. 1 designate routes from each node to the destination node K. The numbers in Fig. 1 represent metrics of links for shortest path calculation (costs in OSPF [10]). Although the proposed method supports the case where the metric of the link is not symmetric, Fig. 1 takes the example network where the metric is symmetric to simplify the discussion.

The routing tables of the nodes upstream of the failed link may change before and after the failure. These nodes correspond to nodes A–F in Fig. 1. Of those nodes, the routes of nodes C–F have changed. For nodes C–E of those nodes, the shortest path after failure (C → D → E) has the opposite direction of the shortest path before failure (E → D → C). In other words, if these nodes (nodes C–E) do not update their own routing tables, a routing loop occurs. However, if node F does not update its routing table, a routing loop does not occur. Therefore, node F does not necessarily have to update its routing table.

To summarize, the node satisfying the following conditions is the node that causes a routing loop if it does not update its routing table.

**Condition 1:** A failed link is on the path to the destination in the shortest-path tree rooted in itself before the failure.

**Condition 2:** After failure, the node exists on the path to the destination in the shortest-path tree rooted at the node detecting the failure.

A node satisfying both of these conditions causes a routing loop unless it updates its routing table. In the proposed method, I use segment routing to set up tunnels through these nodes. By using a node that satisfies Condition 2 but not Condition 1 as the egress node, packets can be sent to their destination without a routing loop.

### 3.3 Checking Conditions

In setting up the rerouting tunnel by segment routing, the node that detects the failure (node C in Fig. 1) forwards the packets originally forwarded via the failure link to the rerouting tunnel. Therefore, a method is required in which node detecting failure efficiently determines the egress node for the destinations of packets that satisfy the conditions shown in 3.1. To find nodes that satisfy Condition 2, the detecting node calculates the shortest-path tree whose root is itself using Dijkstra's algorithm and enumerates the nodes on the path to the destination. The simple method to determine whether these nodes satisfy Condition 1 is to calculate the shortest-path tree rooted from each node, respectively. Alternatively, by calculating the reverse-shortest-path tree with each node as a sink, it is possible to determine whether each node satisfies Condition 1. The computational complexity of Dijkstra's algorithm is $O((N + L) \log N)$, where $N$ and $L$ are the number of nodes and links. The computational complexity of the simple method is $O(N(N + L) \log N)$. I describe an efficient method to determine whether a node satisfying Condition 2 satisfies Condition 1 or not where the computational complexity is $O((N + L) \log N)$.

Let $n_r$ be the node (the detecting node) adjacent to the failed link $e$ and $n_d$ be the destination node. The metric on the shortest path from node $n_i$ to node $n_j$ in the network before the failure is denoted as $m_{i\to j}$, and the same metric in the network after the failure is denoted as $m'_{i\to j}$. The relationship between these nodes and metrics is shown in Fig. 2. The solid and dotted arrows in Fig. 2 represent the shortest paths before and after the failure, respectively.

Assume that node $n_i$ satisfies Condition 1. Since the detecting node $n_r$ connected to the failed link e is a node on the shortest path from $n_i$ to $n_d$, the following formula holds.

$$m_{i\to d} = m_{i\to r} + m_{r\to d}. \tag{1}$$

Next, I show the necessary and sufficient conditions for Condition 2 to hold. If node $n_i$ satisfies Condition 2, the following formula holds, and vice versa.

$$m'_{r\to d} = m'_{r\to i} + m'_{i\to d}. \tag{2}$$

Since the metric on the shortest path from a source node to a destination node before the failure is smaller or equal to that after the failure, the following formula holds.

$$m_{i\to d} \le m'_{i\to d}. \tag{3}$$

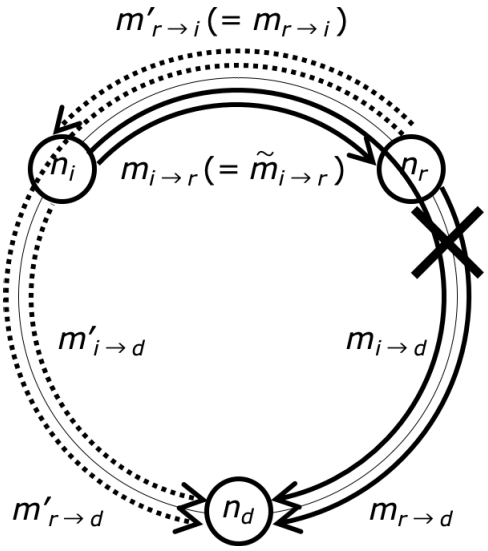


**Fig. 2** Metrics between nodes.

Since the failure of $e$ does not affect the shortest path from $n_r$ to $n_i$, the following formula holds.

$$m_{r\to i} = m'_{r\to i}. \tag{4}$$

The following formula is derived from the above.

$$m_{i\to r} + m_{r\to i} \le m'_{r\to d} - m_{r\to d}. \tag{5}$$

The shortest-path tree $\mathcal{T}_r$ calculated with $n_r$ as the root based on the network before the failure can be used to find $m_{r\to d}, m_{r\to i}$ for an arbitrary node $n_i$, $n_d$ in the network. Similarly, the shortest-path tree $\mathcal{T}_r'$ based on the network after the failure can be used to find $m'_{r\to d}$. Further, $m_{i\to r}$ can be found from the reverse-shortest-path tree $\widetilde{\mathcal{T}_r}$ calculated with $n_r$ as sink. In other words, by calculating the shortest-path trees only three times ($\mathcal{T}_r, \mathcal{T}_r', \widetilde{\mathcal{T}_r}$), any node $n_i$ satisfying the Condition 2 can be judged for whether or not it satisfies Condition 1.

### 3.4 Proposed Algorithm

The proposed algorithm and its notations are shown in Fig. 3 and Table 2. Lines 1–6 in Fig. 3 are the find_egress function for finding egress nodes for all destination nodes. The arguments of the function are source node $n_r$ and failed link $e$. Lines 1–3 calculate the shortest-path tree $\mathcal{T}_r$ before failure, shortest-path tree $\mathcal{T}_r'$ before failure, and reverse-shortest-path tree $\widetilde{\mathcal{T}_r}$. The root or sink of these shortest-path trees are all $n_r$. Line 5 computes a dictionary $\mathcal{D}$ to store the relationship between a destination node and an egress node by the rec_func function.

The arguments of the rec_func function are a source node $n_r$, a target node $n_i$, a list $\mathcal{P}$ that includes paths from source node to target node, a dictionary $\mathcal{D}$ and three shortest-path trees $\mathcal{T}_r$, $\mathcal{T}_r'$, and $\widetilde{\mathcal{T}_r}$. Line 8 calculates the right side of formula (5). Lines 9–12 check whether each node on the path

```
1:  def find_egress( n_r, e, V, E ):
2:      T_r ⇐ SPF(n_r, V, E)
3:      T_r' ⇐ SPF(n_r, V, E − {e})
4:      T̃_r ⇐ RevSPF(n_r, V, E)
5:      D ⇐ rec_func(n_r, n_r, [n_r], φ, T_r, T_r', T̃_r)
6:      return D
7:  def rec_func( n_r, n_d, P, D, T_r, T_r', T̃_r ):
8:      t_d ⇐ Met(n_r, n_d, T_r') − Met(n_r, n_d, T_r)
9:      for n_i in P:
10:         if Met(n_i, n_r, T̃_r) + Met(n_r, n_i, T_r) > t_d:
11:             D[n_d] ⇐ n_i
12:             break
13:     for n_j in Children(n_d, T_r'):
14:         P' ⇐ P + [n_j]
15:         D ⇐ rec_func(n_r, n_j, P', D, T_r, T_r', T̃_r)
16:     return D
```

**Fig. 3** Algorithm for finding egress nodes.

**Table 2** Notations.

| Notation | Mean |
|---|---|
| $\mathcal{V}$ | A set of nodes in network |
| $\mathcal{E}$ | A set of links in network |
| $e$ | Failed link |
| $SPF(n, \mathcal{V}, \mathcal{E})$ | Shortest-path tree from source node $n$ with $\{\mathcal{V}, \mathcal{E}\}$ |
| $RevSPF(n, \mathcal{V}, \mathcal{E})$ | Reverse-shortest-path tree to destination node $n$ with $\{\mathcal{V}, \mathcal{E}\}$ |
| $Met(n_i, n_j, \mathcal{T})$ | Metric from $n_i$ to $n_j$ calculated from normal or reverse-shortest-path tree $\mathcal{T}$ |
| $Children(n, \mathcal{T})$ | Children of $n$ on shortest-path tree $\mathcal{T}$ |
| $\mathcal{P}$ | List of nodes on a path |
| $\mathcal{D}$ | Dictionary from destination nodes to egress nodes : $\{\mathcal{V} \to \mathcal{V}\}$ |

from the source node $n_r$ to the destination node $n_d$ satisfies formula (5); if so, $n_i$ is stored in $\mathcal{D}$ as the egress node for the destination node $n_d$. Whether or not a node $n_i$ satisfies the formula (5) is determined in order from the source node $n_r$ along the path $\mathcal{P}$ until the node not satisfying the condition is found. Therefore, even if there are multiple nodes that do not satisfy the condition for the destination node $n_d$, the node closest to the source node $n_r$ is selected as the egress node. Lines 13–15 recursively calculate the rec_func function on the shortest-path tree for which the root is $n_r$, with all the child nodes $n_j$ of the destination node $n_d$ currently processed in the rec_func function.

When the rec_func function is called on Line 5, its second argument specifies $n_r$. When it is called again on Line 15, its second argument specifies a child node of $n_r$. By such recursive calls of the rec_func function, all nodes on the shortest-path tree $\mathcal{T}_r'$ whose root is $n_r$ are processed. In other words, the rec_function is executed as many times as the number of nodes in $\mathcal{T}_r'$. The dictionary $\mathcal{D}$ finally stores egress nodes for all destination nodes in $\mathcal{V}$.

The proof that the proposed method works properly is given in Appendix.

## 4. Evalution

I evaluate the tunnel lengths in the segment routing calculated

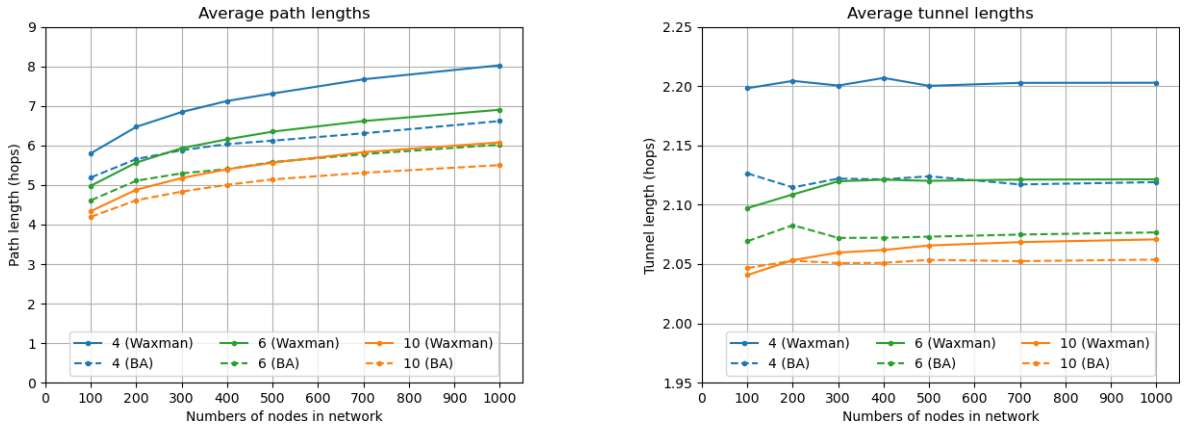


**Fig. 4** Average path lengths and average tunnel lengths.

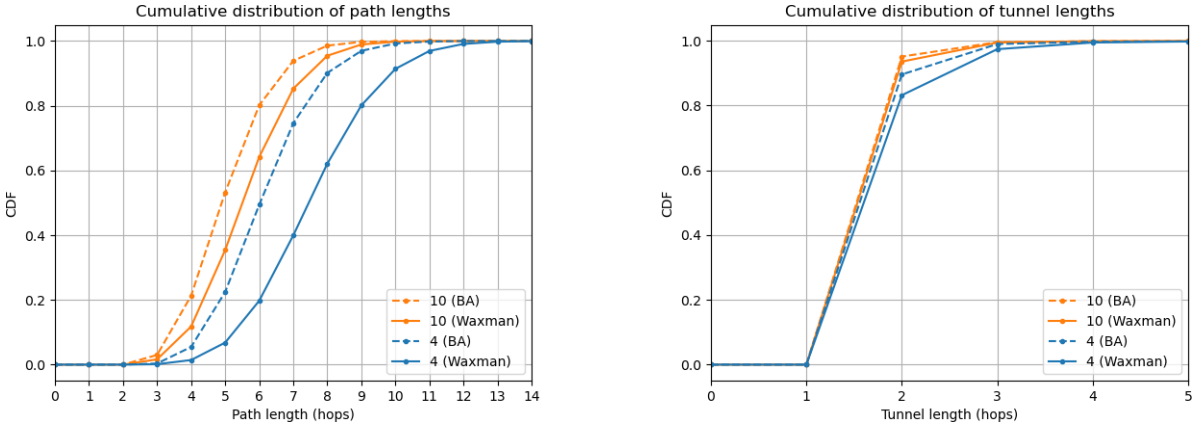


**Fig. 5** Cumulative distribution of path lengths and tunnel lengths.

with the proposed method.

The proposed algorithm looks for an endpoint node $n_i$ along the path $\mathcal{P}$ in the loop part from line 9 to line 12 of Fig. 3. This loop part is repeated until the endpoint is found. In other words, the number of iterations in this loop part is equal to the tunnel length.

In segment routing, the detecting node adds headers for the number of hops in a tunnel, and the packet size increases accordingly. For example, when SRv6 (segment routing over IPv6) is used, the packet length increases by 128 bit (16 octet) for each additional tunnel length [11].

A shorter tunnel length is desirable for both viewpoint of the computational cost and the packet size. Therefore, I evaluated the tunnel lengths in the segment routing calculated with the proposed method using simulation. The evaluation results using the artificially generated topologies and those of the actucal networks are shown in 4.1 and 4.2, respectively.

### 4.1 Results on Waxman and BA Model

I show the results of the simulation using the Waxman model [13] and the Barabasi-Albert (BA) model [14], which are widely used as topology models in network evaluation. The topologies used in the simulation were generated using BRITE [12]. The numbers of nodes were from 100 to 1000, and the average node degrees were 4, 6, and 10. The average path lengths from the detecting node to the destination node and the average tunnel lengths are shown in Fig. 4.

In Fig. 4, the average path lengths in the simulation topologies were about 4 to 8 for the Waxman model and 4 to 7 for the BA model. In both cases, the average path lengths increased as the number of nodes increased and as the node degree decreased.

Figure 4 shows that the average tunnel lengths are about 2.0 to 2.2. When the node degrees are 6 and 10 in the Waxman model, as the number of nodes increases from 100 to 500, the average tunnel lengths tends to increase slightly. However, when the number of nodes is 500 or more, the average tunnel lengths are almost constant regardless of the number of nodes. In the BA model, although the average tunnel lengths slightly increase or decrease when the number of nodes is small, they are almost constant regardless of the number of nodes. These results demonstrate that the average tunnel lengths of the proposed method are at most 2.0 to 2.2 regardless of the size of the network.

Next, Fig. 5 shows the cumulative distribution of path length and tunnel length for networks with 1000 nodes and

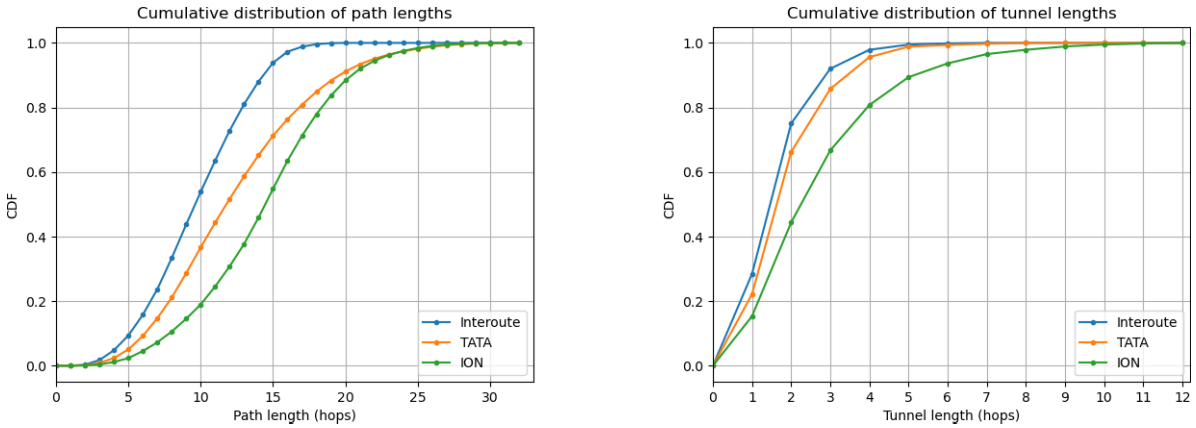


**Fig. 6** Cumulative distribution of path lengths and tunnel lengths of actual networks.

average node degree 4 and 10 in the Waxman, BA model. While Fig. 4 shows that the average path length of Waxman model with an average node degree of 4 is about 8.0, Fig. 5 shows that about 40% of the node pairs have path lengths above the average. On the other hand, when I look at the tunnel length, 83% of all node pairs are less than 2 and 97% are less than 3. Thus, for most node pairs, even if the path length increases, the tunnel length does not increase significantly.

### 4.2 Results on Actual Networks

Next, I show the evaluation results using the topologies of actual networks. I obtained the topology data which are collected and published by The Internet Topology Zoo project [15], and used the topologies which degree is 2 or higher among them. I excluded stub nodes (nodes with degree 1) from evaluation since the stub node has no rerouting path in case of link failure.

Table 3 shows the number of nodes in network, the average degree of the nodes, the average path length between all node pairs, and the average tunnel length for each network. Except for Interoute, ION, and TATA, the average tunnel length is less than 1.67, which is shorter than the results in 4.1. The average path length of Interoute, ION, and TATA is more than 10, which is longer than the other networks. The average degrees of these three networks are 2.4–2.7 and the number of links relative to the number of nodes is smaller than that of other networks. Therefore, the average path length of these three networks is considered to be longer. However, even the ION, which has the longest average tunnel length, has the average tunnel length of only 3.17.

Next, Fig. 6 shows the cumulative distribution of path lengths and tunnel lengths in the Interoute, ION, and TATA. In ION and TATA, there are node pairs with path length of 32. Since there are node pairs with long path lengths, the tunnel lengths in Fig. 6 are also longer than those in Fig. 5. However, in ION with the longest average tunnel length, more than 80% of the node pairs have a path length of 4 or less. In other words, in a network with long average path length, there are some node pairs with long tunnel lengths, however many node pairs have sufficiently short tunnel lengths compared to their path lengths.

**Table 3** Average path and tunnel lengths of actual networks.

| Name | Network size | | Average path length | Average tunnel length |
|---|---|---|---|---|
| | Nodes | Deg. | | |
| Airtel | 8 | 7.3 | 2.60 | 1.28 |
| ATT North America | 25 | 4.6 | 4.02 | 1.62 |
| Belnet (2004) | 19 | 4.1 | 3.17 | 1.17 |
| Belnet (2006) | 20 | 4.1 | 3.20 | 1.18 |
| BT North America | 36 | 4.2 | 4.46 | 1.64 |
| China Telecom | 20 | 4.4 | 3.49 | 1.49 |
| GoodNet | 13 | 4.2 | 3.31 | 1.53 |
| GridNet | 9 | 4.4 | 3.22 | 1.67 |
| Highwinds | 17 | 6.1 | 3.71 | 1.16 |
| IIJ | 27 | 4.1 | 3.70 | 1.43 |
| InternetMCI | 19 | 4.7 | 3.72 | 1.27 |
| Interoute | 104 | 2.9 | 10.18 | 2.07 |
| ION | 116 | 2.4 | 14.85 | 3.17 |
| Janet Lense | 17 | 4.4 | 2.92 | 1.16 |
| NTT | 28 | 15.1 | 3.43 | 1.03 |
| TATA | 137 | 2.7 | 12.91 | 2.32 |

## 5. Discussion

### 5.1 Computational Complexity

This section discusses the computational complexity of the proposed algorithm. The proposed algorithm performs three shortest-path tree calculations (lines 2–4 of Fig. 3). It is known that the computational complexity of Dijkstra's algorithm for calculating the shortest-path tree is $O((N + L) \log N)$, where $N$ and $L$ are the number of nodes and links. In the artificially generated topologies used in 4.1, the number of links is roughly proportional to the number of nodes. Therefore, to simplify the discussion, I assume that the computational complexity of the shortest-path tree is $O(N \log N)$.

Next, I consider the computational complexity of the rec_func function in line 5 of Fig. 3. This function has two iterations, lines 9–12 and lines 13–15. As mentioned at

**Table 4** Comparison of the number of calculations of shortest-path trees.

| Nodes | 100 | 100 | 100 | 500 | 500 | 500 | 1000 | 1000 | 1000 |
|---|---|---|---|---|---|---|---|---|---|
| Average degree | 4 | 6 | 10 | 4 | 6 | 10 | 4 | 6 | 10 |
| The conventional method($A$) | 104 | 106 | 110 | 504 | 506 | 510 | 1004 | 1006 | 1010 |
| The proposed method ($B$) | 6 | 8 | 12 | 6 | 8 | 12 | 6 | 8 | 12 |
| $B/A \times 100$ (%) | 5.8% | 7.6% | 10.9% | 1.2% | 1.6% | 2.4% | 0.60% | 0.80% | 1.2% |

the beginning of 4, the number of times the first iteration is equal to the tunnel length. The second iteration is called recursively by the rec_func function. As mentioned at the end of 3.4, the number of calls to the rec_func function is equal to the number of nodes in the network. From the above, the computational complexity of the rec_func function is $O(MN)$, and that of the find_egress function is $O(N \log N + MN)$, where the average tunnel length is $M$.

According to the evaluation results in Sect. 4, the values of $M$ are 2.0 to 2.2 for artificially generated topologies and 3.17 for ION with smallest degree. The evaluation in 4.1 has confirmed that the tunnel lengths remain constant even when the number of nodes in the network increases. Therefore, the computational complexity of rec_func function can be considered as $O(N)$. The computational complexity of the find_egress function is $O(N \log N)$ because the computational complexity of the Dijkstra algorithm is dominant compared to that of rec_func function.

### 5.2 Comparison with the Conventional Method

This section compares the number of shortest-path tree computations using Dijkstra algorithm in the proposed method and the conventional method. Both of these methods require three types of shortest-path trees, respectively. First, I consider shortest-path trees commonly used by both methods. Both require shortest-path tree $\mathcal{T}_r$ rooted $n_r$ before failure and shortest-path tree $\mathcal{T}_r'$ rooted $n_r$ after failure, where $n_r$ is the node that calculates the shortest-path trees. When the number of links connected to $n_r$ is $L$, there are $L$ failure patterns, so $\mathcal{T}_r'$ are calculated $L$ times for each failure. Therefore, including the computation of $\mathcal{T}_r$, the shortest-path trees are calculated $L + 1$ times.

Next, I consider the differences between the proposed method and the conventional method. The conventional method requires reverse-shortest-path trees $\widetilde{\mathcal{T}_d}$ with destination $n_d$ as sink. If the number of nodes in the network is $N$, the shortest-path tree calculations are required $N - 1$ times. In contrast, the proposed method requires only one calculation of the shortest-path tree $\widetilde{\mathcal{T}_r}$ with $n_r$ as sink.

From the above, the number of shortest-path tree calculations are $L + N$ times for the conventional method and $L + 2$ times for the proposed method. Table 4 shows the result quantitatively. This table shows that there is a large difference in the number of shortest-path tree calculations between the conventional and the proposed methods, and the difference becomes larger as the number of nodes in the network increases.

### 5.3 Disadvantage over the Conventional Method

This section discusses the disadvantage of the proposed method over the conventional method. The conventional method calculates the reverse-shortest-path tree $\widetilde{\mathcal{T}_d}$ with $n_d$ as the sink. By combining $\widetilde{\mathcal{T}_d}$ with $\mathcal{T}_r'$, multiple candidates for the rerouting path from $n_r$ to $n_d$ can be found. For example, congestion can be avoided by distributing traffic to the multiple rerouting paths. On the other hand, the proposed method treats only the shortest path calculated from $\mathcal{T}_r'$ as candidate for rerouting path. There are many cases where detour using the shortest path is sufficient, however there are cases where traffic distribution is required. The extension of the proposed method for traffic distribution is a future work.

### 5.4 Discussion on the Prerequisites

As for Prerequisite 1 in 3.1, most of the links in wired IP networks are bi-directional. Most backbone networks that require the proposed method are composed of point-to-point links. This prerequisite, which is also used in the original TI-LFA [6], is not considered to be a major constraint in practice.

As stated in Prerequisite 2, the proposed method does not support multiple failure, which is also not supported by the original TI-LFA. In these methods, packets are forwarded using segment routing immediately after a failure, and then packet forwarding using normal IP routing is restored by a routing protocol such as OSPF. Then, when another failure occurs, these methods will work if there is an alternate path to the destination. However, when multiple failures occur at the same time, even if each node that detects a failure independently forms a path using these methods, there will be cases where packets do not reach their destination. For example, it is the case that a failure exists on a path formed by a node that has detected another failure. Dealing with such cases is a future work not only for the proposed method but also for the TI-LFA itself.

The proposed method targets a link failure in Prerequisite 2. The original TI-LFA also covers a node failure where the hardware of the node equipment fails. The proposed method can be modified to handle the node failure. Let $n_f$ be a failed node, $n_r$ be a neighbor node of $n_f$, and $\mathcal{E}_f$ be the set of all links connected to $n_f$. Instead of Line 3 in Fig. 3, the following $\mathcal{T}_r'$ is used.

$$\mathcal{T}_r' \Rightarrow SPF(n_r, \mathcal{V} - \{n_f\}, \mathcal{E} - \mathcal{E}_f)$$

The hardware failure of a line card causes multiple links accommodated by that line card to be unavailable at the

same time. The following $\mathcal{T}_r'$ can be used to deal with this failure. Let $\mathcal{E}_r$ be the set of links accommodated by the failed line card.

$$\mathcal{T}_r' \Rightarrow SPF(n_r, \mathcal{V}, \mathcal{E} - \mathcal{E}_r)$$

A detailed discussion of these cases remain as future work.

The proposed method can work even when there are multiple shortest paths, which were assumed to be non-existent in Prerequisite 3. Even if there are multiple shortest paths from the failure detection node $n_r$ to the destination node $n_d$, the algorithm in Fig. 3 only needs to select one of the shortest paths in calculation process of $\mathcal{T}_r'$ in Line 3. Even if the branch point between the selected shortest path and the other shortest path exists between $n_r$ and $n_i$ chosen by the proposed method, the other shortest path is not affected because packet is forwarded from $n_r$ to $n_i$ by segment routing. If the branch point is between $n_i$ and $n_r$, packet may be forwarded on the shortest path not chosen by $\mathcal{T}_r'$. Even in that case, the packet can reach its destination $n_d$ because this shortest path does not include the failed link $e$. Thus, the proposed method works even when there are multiple shortest paths in network. However, as mentioned in 5.3, the fact that the proposed method is limited to only one shortest path to be used is a disadvantage over the conventional method.

## 6. Conclusion

This paper proposed a method to efficiently calculate egress nodes for IP fast-rerouting using segment routing on an IP network. The simple method requires $N$ shortest path tree calculations, where the number of nodes in network is $N$. In contrast, the proposed method requires only three shortest path tree calculations.

The results of evaluation using artificially generated topologies showed that the average tunnel lengths calculated with the proposed method are at most 2.0 to 2.2 hops regardless of the size of the network. I also showed that the computational complexity of the proposed algorithm is $O(N \log N)$.

The proposed method assumes that all nodes in network support segment routing. Until segment routing is widespread enough, there may be a network including both nodes that support segment routing and those that do not support it. A future work is to devise an efficient method to determine a loop-free rerouting path in such a network.

### References

[1] K. Suzuki, "An efficient algorithm for TI-LFA rerouting path calculation," Proc. 2020 International Conference on Emerging Technologies for Communications (ICETC2020), D1-4, 2020.
[2] A. Kvalbein, A.F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations," Proc. IEEE INFOCOM 2006, pp.1–11, 2006.
[3] S. Nelakuditi, S. Lee, Y. Yu, Z. hang, and C. Chuah, "Fast local rerouting for handling transient link failures," IEEE/ACM Trans. Netw., vol.15, no.2, pp.359–372, 2007.
[4] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-free alternates," RFC5286, IETF, 2008.
[5] S. Bryant, C. Filsfils, S. Previdi, M. Shand, and N. So, "Remote loop-free alternate (LFA) fast reroute (FRR)," RFC7490, IETF, 2015.
[6] A. Bashandy, C. Filsfils, B. Decraene, S. Litkowski, P. Francois, D. Voyer, F. Clad, and P. Camarillo, "Topology independent fast reroute using segment routing," https://tools.ietf.org/html/draft-francois-segment-routing-ti-lfa-05
[7] K. Suzuki, M. Jibiki, and K. Yoshida, "Selective precomputation of alternate routes using link-state information for IP fast restoration," IEICE Trans. Commun., vol.E93-B, no.5, pp.1085–1094, May 2010.
[8] K. Suzuki, M. Jibiki, and K. Yoshida, "Comparison of proactive and reactive methods for IP fast restoration using localization algorithm," Proc. 4th International Conference on Signal Processing and Communication Systems (ICSPCS), IEEE, 2010.
[9] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment routing architecture," RFC8402, IETF, 2018.
[10] J. Moy, "OSPF version 2," RFC2328, IETF, 1997.
[11] C. Filsfils, D. Dukes, S. Previdi, J. Leddy, S. Matsushima, and D. Voyer, "IPv6 segment routing header (SRH)," RFC8754, IETF, 2020.
[12] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," Proc. Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp.346–353, 2001.
[13] B.M. Waxman, "Routing of multipoint connections," IEEE J. Sel. Area Commun., vol.6, no.9, pp.1617–1622, 1988.
[14] A.L. Barabasi and R. Albert, "Emergence of scaling in random networks," Science, vol.286, no.5439, pp.509–512, 1999.
[15] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," IEEE J. Sel. Areas Commun., vol.29, no.9, pp.1765–1775, 2011.

## Appendix: Proof of Packet Reachability

Node $n_r$ that detected the failure of link $e$ forwards packets destined to $n_d$ using segment routing until $n_i$ obtained by the proposed method. Then, the packets are forwarded from $n_i$ to $n_d$ according to the normal IP routing. In this way, I prove whether the proposed method can correctly forward packets to the destination.

**Lemma 1:** By using segment routing, it is possible to forward packets destined to $n_d$ from $n_r$ that detected the failure of $e$ to $n_i$ calculated by the algorithm shown in Fig. 3.

**Proof 1:** Lines 13 and 14 of Fig. 3 calculates a path from $n_r$ to $n_i$ along the shortest path tree $\mathcal{T}_r'$ which does not contain the failed link $e$. Since this path does not contain $e$, the packet can reach from $n_r$ to $n_i$ by using segment routing. □

**Lemma 2:** Regardless of the failure of link $e$, packets destined to $n_d$ can reach from $n_i$ to $n_d$ by the normal IP routing.

**Proof 2:** Using proof by contradiction, I show that the failed link $e$ does not exist on the shortest path from $n_i$ to $n_d$ before the failure. Assume that $e$ exists on the shortest path before failure from $n_i$ to $n_d$.

The $n_i$ chosen by Line 10 in Fig. 3. satisfies the following inequality (negation of Inequality (5)).

$$m_{i\to r} + m_{r\to i} > m'_{r\to d} - m_{r\to d} \qquad (A\cdot 1)$$

Since $n_i$ is a node on the shortest path after the failure from $n_r$ to $n_d$, Eq. (2) holds. From the assumption, the

shortest path from $n_r$ to $n_i$ does not contain any failed links, so Eq. (3) holds. Substituting Eqs. (2) and (3) into Inequality (A· 1) leads to the Inequality (A· 2).

$$m_{i\to r} + m_{r\to d} > m'_{i\to d} \tag{A· 2}$$

From the assumption that $e$ exists on the shortest path before the failure from $n_i$ to $n_d$, $n_r$ also exists on this shortest path. Since $m_{i\to r} + m_{r\to d} = m_{i\to d}$ holds, the following inequality is derived from Inequality (A· 2).

$$m_{i\to d} > m'_{i\to d} \tag{A· 3}$$

This inequality shows the contradiction that the metric of the shortest path from $n_i$ to $n_d$ is lower after the failure than before the failure. Therefore, the assumption that the failed link $e$ exists on the shortest path before the failure from $n_i$ to $n_d$ is rejected.

Since the failed link $e$ is not on the shortest path from $n_i$ to $n_d$ before the failure, this shortest path does not change even after the failure. Therefore, packets destined to $n_d$ can reach $n_d$ from $n_i$ without updating the route tables after the failure. □

**Theorem 3:** By the proposed method, it is possible to reach packet from any source node $n_s$ to any destination node $n_d$ without each node in network updating its routing table after failure of link $e$.

**Proof 3:** If $e$ is not on the shortest path before the failure from $n_s$ to $n_d$, packets destined to $n_d$ are reachable without updating the route tables. Next, consider the case where this shortest path includes $e$. Let $n_r$ be the node adjacent to $e$ upstream on the shortest path before the failure. A packet destined to $n_d$ reaches from $n_s$ to $n_d$ by normal IP routing. By Lemma 1, this packet reaches from $n_r$ to $n_i$ determined by the proposed method. By Lemma 2, this packet reaches from $n_i$ to $n_d$. From the above, this theorem is proved. □

**Kazuya Suzuki** received M.E. degree in Electrical Engineering from Tokyo Metropolitan University in 1997, and Ph.D. degree in Systems Management from the University of Tsukuba in 2011. He joined NEC Corporation in 1997, where he has been undertaking research and development of networking technologies. He is currently an Associate Professor of Akita Prefectural University. His research interests include internet routing, software-defined networking and IoT applications.