PAPER *Special Section on Technology and Architecture for Sustainable Growth of the Internet*

Selective Precomputation of Alternate Routes Using Link-State Information for IP Fast Restoration

Kazuya SUZUKI[†,††a)], ***Student Member***, **Masahiro JIBIKI**[†], ***and*** **Kenichi YOSHIDA**[††], ***Members***

SUMMARY The availability of IP networks has increased its importance due to the evolving use of real-time and mission-critical applications on IP networks. Methods for preparing alternate routing tables that can be used for fast restoration from link failures have been investigated. In such methods, each node has to compute a number of alternate routing tables in advance since they have to prepare for each potential failure. The resulting huge number of alternate routing tables has prevented these methods from being deployed. In this paper, we propose a method for reducing the number of alternate routing tables for link failure. It analyzes three types of shortest path trees on the basis of link-state information. We show that the number of alternate routing tables can be reduced to 1/100, on average, from that with the conventional method, and that they are small enough to be stored in the memory of IP routers.

key words: *IP fast restoration, link-state routing*

1. Introduction

The Internet has become an important social infrastructure, and its importance is increasing daily. Various business applications, such as VoIP and e-commerce, are becoming widely used as are various personal applications. Since business applications tend to require high availability, today's IP networks have to provide services that are as highly available as leased-line services.

Although required availability is significantly high, e.g. leased-line services achieve restoration times of less than 50 ms [1], IP networks cannot achieve such high availability. For example, recent studies [2], [3] have revealed a high frequency of failures in well managed IP networks. The weakness of the basic mechanism used to achieve IP network availability worsens this drawback. This mechanism, i.e., the IP restoration mechanism, depends on redundant links and intra-domain routing protocols. When a failure occurs, redundant links enable intra-domain routing protocols, e.g., OSPF [4] and IS-IS [5], to calculate an alternate routing table without using failed equipments. Although this alternate routing table can be used to restore IP availability, calculating them typically takes from 100 to 400 ms [6], [7]. Thus, IP networks cannot achieve sufficient availability.

A naive method for restoring IP networks calculates alternate routing tables from link-state information in advance, and uses them when a failure occurs. However, the huge number of alternate routing tables has prevented such naive methods from becoming practical since there are too many potential failures and resulting alternate routing tables. In this paper, we propose a method for reducing the number of alternate routing tables. By analyzing three types of shortest path trees, the proposed method reduces the number of alternate routing tables needed for potential link failures.

This paper is structured as follows. In Sect. 2, we briefly survey related work. In Sect. 3, we present the basic idea for reducing the number of alternate routing tables and describe an algorithm for implementing it. In Sect. 4, we describe how we experimentally evaluated our proposed method and show that it can reduce the amount of memory to store alternate routing tables to a practical size. We show that the calculation cost is also sufficiently low for practical use. After Sect. 5 discusses future work, Sect. 6 summaries our findings.

2. Related Work

Much work has been studied to achieve fast restoration from failures in IP network.

One approach to IP network restoration is to use a routing protocol. Restoration using a routing protocol can be divided into four processes: failure detection, failure notification, alternate route calculation, and routing table update. To shorten the time for calculating alternate routes, incremental SPF algorithm [8] was proposed. Since the incremental SPF algorithm only reconstructs the difference from the original shortest path tree after failure, the calculation time is reduced. However, the calculation time cannot be reduced to zero if only the incremental SPF algorithm is used. There is limit on the reduced restoration time. Nevertheless, the concept of the incremental SPF algorithm is useful. It can be used as a process to reduce the number of excess alternate routing tables. Section 3.1 explains the process of reduction using the incremental SPF algorithm in detail. We also adopted the incremental SPF algorithm to reduce the calculation cost for preparing alternate routing tables, which we discuss in Sect. 4.4.

Sub-second restoration after a failure has been achieved by tuning the routing protocol parameters [6], [7]. Iannaccone et al. [6] measured the time for each restoration process using an actual Tier-1 network, and estimated the restoration time for a large network. Francois et al. [7] analyzed the relationship between the restoration time and var-

Manuscript received September 15, 2009.
Manuscript revised December 28, 2009.
[†]The authors are with the System Platforms Research Laboratories, NEC Corporation, Kawasaki-shi, 211-8666 Japan.
[††]The authors are with the Graduate School of Business Sciences, University of Tsukuba, Tokyo, 112-0012 Japan.
a) E-mail: kazuya@ax.jp.nec.com
DOI: 10.1587/transcom.E93.B.1085

ious timers for the routing protocol. They reported an optimum value of the timers for attaining fast restoration. However, even with parameter tuning, a restoration time of less than 50 ms has not been achieved.

IP fast reroute methods [9], [10] have been proposed for shortening the restoration time after a failure in an IP-based network. Several alternate routing tables are prepared in advance, and the routers switch to one immediately after a failure. These methods use original forwarding mechanisms that differ from those of conventional IP forwarding for quicker failure notification. One such method [9] 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. [10] 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-based networks.

Previous work [11] showed that routing loops occur more frequently when a node is close to the location of a failure. This finding suggests that, for achieving packet reachability, it is sufficient for nodes that are close to the failure location to update their own routing table. Our proposed method is based on this finding.

3. Reducing Number of Alternate Routing Tables

3.1 Reducing Processes in Naive Method

Various related techniques have been proposed to achieve fast IP restoration. For example, the incremental SPF algorithm [8] uses a procedure that computes routing tables incrementally. This incremental procedure can be used as a naive method for computing alternate routing tables.

Let us assume the network topology in Fig. 1(a) and that there is a link failure between nodes D and F. OSPF, a typical intra-domain routing protocols, computes the whole routing table from scratch. The computation cost can be reduced by using the incremental SPF algorithm to compute a subset of the routing table. For example, a packet going from nodes A to G uses the link between nodes D and F (see Fig. 1(b)). Thus, the incremental SPF algorithm computes a routing table of node A. It does not compute a routing table of node E since a packet from E does not use the link between D and F (see Fig. 1(c)). By omitting the calculation related to node E, the incremental SPF algorithm reduces the computing cost.

We can use this incremental procedure to create alternate routing tables. For each possible failures, we can use this procedure to calculate alternate routing tables after failure. By storing these alternate routing tables in each nodes and use these tables in case of failures, each node can restore IP availability.

The criteria under which the incremental SPF algorithm omits calculations are based on the shortest path tree. For example, the network in Fig. 1 has eight links, and the alternate routing table needed for each node depends on the failed link. Without calculating eight alternate routing tables of node A, the incremental SPF algorithm calculates only six tables, which corresponds to the links in the shortest path tree from node A (see Fig. 1(b)). The calculations for two links, between nodes C–E and E–G, are omitted because these links are not on the shortest path from node A. Hereafter, we call this the *on shortest path tree* method.

Although the *on shortest path tree* method reduces the number of alternate routing tables compared with the exhaustive approach, it still requires an large number of routing tables (see Sect. 4.5). In our proposed method, the number of alternate routing tables is reduced even further. The key idea is to concentrate on packet reachability, and to give up the shortest path.

To create the shortest path from node A with failure between D and F, we have to prepare an alternate routing table of node A. However, if we consider only availability, i.e. reachability from node A to G in Fig. 2(a), we can use the original routing table for node A if node D has an appropriate routing table for link failure between nodes D and F. For the same reason, node A can use the original routing table after link D–E, E–G, or F–G has failed. However, a routing loop occurs between nodes A and B if node A does not update its routing table after link B–C has failed (see Fig. 2(b)). In short, node A needs alternate routing tables for only three potential link failures (A–B, A–D and B–C). The calculation of the five alternate routing tables can be omitted.

The following sections describe and discuss our

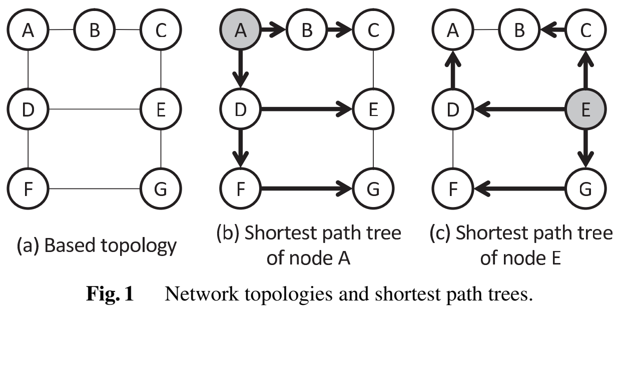

A
B
C
D
E
F
G
(a) Based topology
(b) Shortest path tree of node A
(c) Shortest path tree of node E

Fig. 1 Network topologies and shortest path trees.

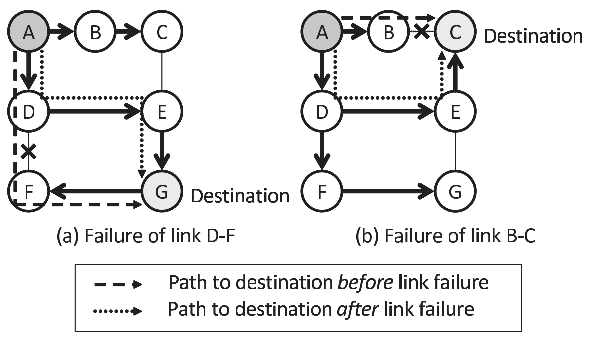

A
B
C
D
E
F
G
Destination
(a) Failure of link D-F
(b) Failure of link B-C
Path to destination *before* link failure
Path to destination *after* link failure

Fig. 2 Link failures and routing table updates.

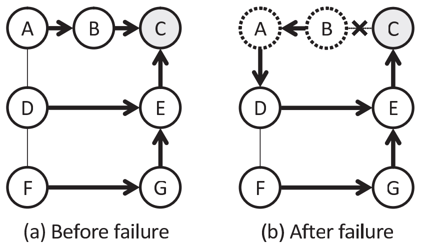


Fig. 3 Reverse shortest path tree.

method for reducing the number of alternate routing tables for a link failure based on this idea. Section 3.2 presents the criteria for updating a routing table. Section 3.3 presents a distributed approach for identifying the nodes that satisfy the criteria. Section 3.4 describes our proposed procedure for IP fast restoration using the distributed approach.

3.2 Criteria for Routing Table Update

We specify the criteria for identifying those nodes that need to update routing table after a particular link failure.

Since each packet is forwarded in accordance with the destination in its header, it is important to investigate the change in the path to a destination after a failure. The change is investigated by using the reverse shortest path trees, as illustrated in Fig. 3. The shortest path tree indicates the shortest paths from a specific source node, whereas the reverse shortest path tree indicates the shortest paths to a specific destination node.

If the link between nodes B and C fails, as illustrated in Fig. 3, the direction of the path between nodes A and B changes. If only a part of these node does not update its routing table after the failure, a routing loop occurs between them. Therefore, nodes A and B should both update their routing tables. These nodes have two characteristics in common. 1) The shortest path from these nodes to node C before failure includes the failed link (see Fig. 3(a)). 2) These nodes are on the shortest path from B, which is connected with the failed link, to C after failure (see Fig. 3(b)). Consequently, to identify which nodes need to update routing table after a link failure, we simply need to determine which nodes have these two characteristics.

Let us summarize these characteristics. Assume that e is the failed link, n_r is a node connected with e, and n_d is the destination node. If n_i satisfies the following criteria, n_i needs to update its routing table after e fails.

Criterion 1: Before a failure at e, the shortest path from n_i to n_d uses link e.

Criterion 2: After a failure at e, the shortest path from n_r to n_d uses n_i.

Criterion 1 is a naive standard used to enumerate the candidate nodes for updating the routing table. Criterion 2 is used to avoid routing loops. Note that the *on shortest path tree* method essentially uses criterion 1. It calculates unnecessary alternate routing tables because it does not use criterion 2.

With our proposed method, only those nodes satisfying both criteria 1 and 2 for n_d update their route to n_d in their routing table after failure of e. We prove that it is sufficient to update the routes of these nodes to restore the packet reachability in the network in Appendix.

We will now discuss the calculation cost involved in checking the criteria for all nodes in the network. An exhaustive approach requires calculation of the shortest path trees for all possible combination of source nodes n_i and link failures e. Thus, the total calculation cost with the exhaustive approach is $N \times L$ times SPF calculations, where N is the number of nodes, and L is the number of links in the network. Since the *on shortest path tree* method checks only criterion 1, the total calculation cost with this method is N times SPF calculations. In contrast, a method which checks both criteria requires $N + 2L$ times SPF calculations. This can be broken down into N times SPF calculations for source nodes, and $2L$ times SPF calculations for both ends of the links. The former is used to check criterion 1, and the latter to check criterion 2.

3.3 Distributed Processing for Checking Criteria

This section presents a distributed approach for each node in network to identify the nodes that satisfy the criteria. We can simply design a distributed processing method in which each node checks criteria 2 in parallel. This approach requires $N + D_r$ times SPF calculations at each node, where D_r is the number of links that node n_r has[†]. The SPF calculations incur heavy costs and they are huge for all nodes.

We have developed a method that can check both criteria with only a few SPF calculations by using the characteristics of the shortest path length, i.e., a metric. To explain the characteristics, we use the notation shown in Table 1. The last seven notations in Table 1 define important metrics, and Fig. 4 illustrates them graphically.

First, we present a necessary condition to satisfy criterion 1. Since node n_r, which is connected with e, is on the shortest path between n_i and n_d, the following equation holds:

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

This equation is a necessary condition for satisfying criterion 1, but not a sufficient one. This is because the shortest path from n_i to n_d may not include e, but n_r. The enumerating algorithm explained in the next section has a step to avoid this case. A necessary and sufficient condition to satisfy criterion 2 is as follows. If node n_r satisfies criterion

[†]If each node connected with e checks the criteria as node n_r, all nodes have to check criterion 1 independently. Therefore, each node requires N times SPF calculations for checking criterion 1 and D_r times that for checking criterion 2. In contrast, if we check criterion 1 in parallel, each node has to calculate SPF $1 + 2L$ times as source node n_i. Here, all nodes have to check criterion 2 independently. Since $L > N$ and $L > D_r$ in general, we have selected the former, which requires $N + D_r$ calculations by node.

Table 1 Notations.

$\mathcal{V}$	=	Nodes in network
$\mathcal{E}$	=	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 $\{\mathcal{V}, \mathcal{E}\}$
$Metric(n_s, n_d, \mathcal{T})$	=	Metric from n_s to n_d calculated from normal or reverse shortest path tree $\mathcal{T}$
$Nexthop(n_s, n_d, \mathcal{T})$	=	Nexthop for destination node n_d on routing table of n_s calculated from shortest path tree $\mathcal{T}$
$Link(n_i, n_j)$	=	Link between n_i and n_j
$\mathcal{T}_r$	=	$SPF(n_r, \mathcal{V}, \mathcal{E})$
$\mathcal{T}_i$	=	$SPF(n_i, \mathcal{V}, \mathcal{E})$
$\mathcal{T}'_r$	=	$SPF(n_r, \mathcal{V}, \mathcal{E} - \{e\})$
$\widetilde{\mathcal{T}}_r$	=	$RevSPF(n_r, \mathcal{V}, \mathcal{E})$
$m_{i\to d}$	=	$Metric(n_i, n_d, \mathcal{T}_i)$
$m_{i\to r}$	=	$Metric(n_i, n_r, \mathcal{T}_i)$
$m_{r\to d}$	=	$Metric(n_r, n_d, \mathcal{T}_r)$
$m'_{r\to d}$	=	$Metric(n_r, n_d, \mathcal{T}'_r)$
$m'_{r\to i}$	=	$Metric(n_r, n_i, \mathcal{T}'_r)$
$m'_{i\to d}$	=	$Metric(n_i, n_d, \mathcal{T}'_i)$
$\widetilde{m}_{i\to r}$	=	$Metric(n_i, n_r, \widetilde{\mathcal{T}}_r)$

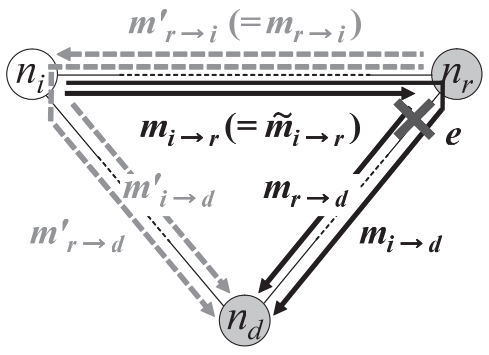

$m'_{r\to i}\ (= m_{r\to i})$
n_i
n_r
$m_{i\to r}\ (= \widetilde{m}_{i\to r})$
e
$m'_{i\to d}$
$m_{r\to d}$
$m'_{r\to d}$
$m_{i\to d}$
n_d

Fig. 4 Metrics between nodes related to criteria.

2,

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

We use the following inequality to illustrate the relationship between Eqs. (1) and (2). Since it is trivial for a metric between the source and destination nodes before a failure to be less than or equal to that after a failure, the following inequality holds:

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

Substituting Eqs. (1) and (2) into (3) results in the following inequality:

$$m_{i\to r} + m_{r\to d} \le m'_{r\to d} - m'_{r\to i}. \tag{4}$$

Thus,

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

We modify Inequality (5) by using the following equations. Since failure e does not change the path between nodes n_r and n_i,

$$m'_{r\to i} = m_{r\to i}. \tag{6}$$

The $m_{i\to r}$ calculated from $\mathcal{T}_i$ (i.e., the shortest path tree with source node n_i) is the same value as $\widetilde{m}_{i\to r}$ calculated from $\widetilde{\mathcal{T}}_r$ (i.e., the reverse shortest path tree with destination node n_r).

$$m_{i\to r} = \widetilde{m}_{i\to r}. \tag{7}$$

Finally, substituting Eqs. (6) and (7) into (5) gives:

$$\widetilde{m}_{i\to r} + m_{r\to i} \le m'_{r\to d} - m_{r\to d}. \tag{8}$$

Link failure e changes the shortest path between nodes n_r and n_d, and changes the metric between these nodes. The right-hand side of Inequality (8) shows this difference, which we can calculate by using the shortest path trees with and without the failure. We can also calculate the second term on the left-hand side using the shortest path tree without failure. To calculate the first term on the left-hand side, a reverse shortest path tree to node n_r is necessary. Thus, we can evaluate Inequality (8) from these three shortest path trees (i.e., $\mathcal{T}_r$, $\mathcal{T}'_r$, and $\widetilde{\mathcal{T}}_r$). Note that we can calculate these metrics related to Inequality (8) using the shortest path trees and the reverse shortest path tree for a specific node, n_r.

All the nodes whose routing tables should be updated satisfy Inequality (8). This inequality requires only $2 + D_r$ SPF calculations, where D_r is the number of links that the node has. This can be broken down into two SPF calculations for $\mathcal{T}_r$ and $\widetilde{\mathcal{T}}_r$, and D_r SPF calculations for $\mathcal{T}'_r$ with each failure in links that n_r has. Since all routers regularly calculate $\mathcal{T}_r$ for creating the routing tables they usually use. Briefly, n_r additionally requires $1 + D_r$ SPF calculations for checking the criteria. By using Inequality (8), we can finally reduce SPF calculations needed for checking in parallel from $N + D_r$ to $1 + D_r$.

Next we consider the total number of SPF calculations for parallel processing by all nodes in the network, $\mathcal{N}$. Since all nodes requires $1 + D_r$ times SPF calculations, the total of that is showed as the following.

$$\sum_{i\in N}(1 + D_i) = N + \sum_{i\in N} D_i = N + 2L \tag{9}$$

This equation shows that total number of SPF calculations for parallel processing is the same as that for concentrated processing. In brief, parallel processing using Inequality (8) does not require any overhead compared with concentrated processing.

3.4 Procedure for IP Fast Restoration

The criteria discussed above are used in the procedure for fast IP network restoration.

The algorithm used to count $\mathcal{V}_u$, which is the set of

```
Enumerating algorithm( $n_r$, $e$ )
1:  $\mathcal{T}_r \Leftarrow SPF(n_r, \mathcal{V}, \mathcal{E})$
2:  $\mathcal{T}'_r \Leftarrow SPF(n_r, \mathcal{V}, \mathcal{E} - \{e\})$
3:  $\widetilde{\mathcal{T}}_r \Leftarrow RevSPF(n_r, \mathcal{V}, \mathcal{E})$
4:  for $\forall n_d \in \mathcal{V}$ do
5:    $n_k \Leftarrow Nexthop(n_r, n_d, \mathcal{T}_r)$
6:    next if $e \neq Link(n_r, n_k)$
7:    $\mathcal{V}_u^{(d)} \Leftarrow \phi$
8:    $n_i \Leftarrow Nexthop(n_r, n_d, \mathcal{T}'_r)$
9:    $t_d \Leftarrow Metric(n_r, n_d, \mathcal{T}'_r) - Metric(n_r, n_d, \mathcal{T}_r)$
10:   while $Metric(n_i, n_r, \widetilde{\mathcal{T}}_r) + Metric(n_r, n_i, \mathcal{T}_r) \leq t_d$ do
11:     $\mathcal{V}_u^{(d)} \Leftarrow \mathcal{V}_u^{(d)} \cup \{n_i\}$
12:     $n_i \Leftarrow Nexthop(n_i, n_d, \mathcal{T}'_r)$
13:   end
14: end
15: return $\mathcal{V}_u$
```

Fig. 5 Enumerating algorithm.

nodes satisfying both criteria 1 and 2, from node n_r and link e is shown in Fig. 5. Here, n_r is an end node of link e. Steps 5 and 6 are for avoiding cases that satisfy Eq. (1), but criterion 1. Step 5 calculates the nexthop of n_r on the shortest path to n_d before failure. If the link between n_r and its nexthop is not e, no nodes satisfies criterion 1 for the combination of n_r, n_d, and e. Then, steps 7 to 13 are skipped for this n_d.

Using this algorithm, we can identify the nodes that need to update routing tables after failure, and have them prepare alternate routes. This is done in four steps by each node n_r.

1. Calculate normal routing table.
2. Enumerate $\mathcal{V}_u^{(d)}$ using the algorithm in Fig. 5.
3. Notify all nodes n_u in $\mathcal{V}_u^{(d)}$.
4. Repeat steps 1 and 2 for all $\mathcal{V}_u^{(d)}$.

This procedure assumes that all nodes have a function for creating an alternate routing table for a specified link failure. Each node n_u, which notified above, then performs the following procedure.

5. Sort messages received from n_r by failed link e.
6. Calculate the shortest path tree without e, and create alternate routes to nodes that is as n_d in the sorted messages for e.
7. Store the alternate routes in an alternate routing table for e.
8. Repeat steps 6 and 7 for all e in the sorted messages.

When node n_r detects a failure in connecting link e, n_r sends a message requesting to switch to the appropriate alternate routing table. This notification can be achieved with a conventional process, such as flooding a link-state advertisement (LSA) in OSPF.

The exhaustive and *on the shortest path* approaches do not require steps 1 to 5. In other word, these steps are overhead of our proposed method. Step 3 is a notification step. Since this step must be repeated for all nodes in $\mathcal{V}_u$, the number of nodes in $\mathcal{V}_u$ affects the amount of overhead. A later evaluation showed that the number of nodes is only a few percent of the total number in the network, as shown in Fig. 11. This overhead is therefore acceptable amount.

4. Experimental Results

We evaluated the number of alternate routing tables and the amount of computing resources required to assess the effect of our proposed method. We used both artificially generated topologies and those of real networks.

4.1 Topologies Used in Experiments

We used topologies generated by the BRITE topology generator [12] to investigate how our proposed method worked in networks with various parameters. BRITE is widely used for generating topologies for simulating networks. Its use enabled us to analyze the effect of such parameters as the number of nodes and their average degree. Although we could select the model for generating topologies with BRITE, we used the Barabasi-Albert (BA) model [13] to generate the topologies in our experiments.

To simplify the discussion, we made three assumptions about the topologies generated with BRITE.

- All links are point-to-point and bi-directional.
- All links have symmetric costs.
- All nodes in the network have a single address, and no links have a network address. In other words, the destinations in the routing tables that all nodes have are nodes, not links.

Sections 4.2 to 4.5 present the results with BA topologies and discuss our analysis of the effect of various parameters. In Sect. 4.6, we use the topologies of actual networks, i.e. GÉANT2 [14], Internet2(Abilene) [15] and SINET3 [16], to complement our analysis. We excluded stub nodes (nodes with degree 1) from the analysis since a link failure does not change their path.

4.2 Number of Alternate Routing Tables

To estimate the number of alternate routing tables that have to be prepared on each node for a single link failure, we generated various networks by changing the assumed number of nodes and their average degrees. Figure 6 plots the results.

Figure 6(a) plots the average number of alternate routing tables. Although an increase in the number of nodes and their degree both increased the required number of alternate routing tables, the average number was relatively small. For example, with the proposed method, only 19 alternate routing tables are needed for networks with 1000 nodes and an average node degree of 12. Since the actual network degrees are less than 4 (see Sect. 4.6), this number is less than we expected.

Figure 6(c) plots the maximum number of alternate routing tables. Although up to 350 alternate routing tables are needed for networks with 1000 nodes and an average degrees of 12, only of 5% of the nodes require more than 50 alternate routing tables. Figure 6(b) shows that 95% of the

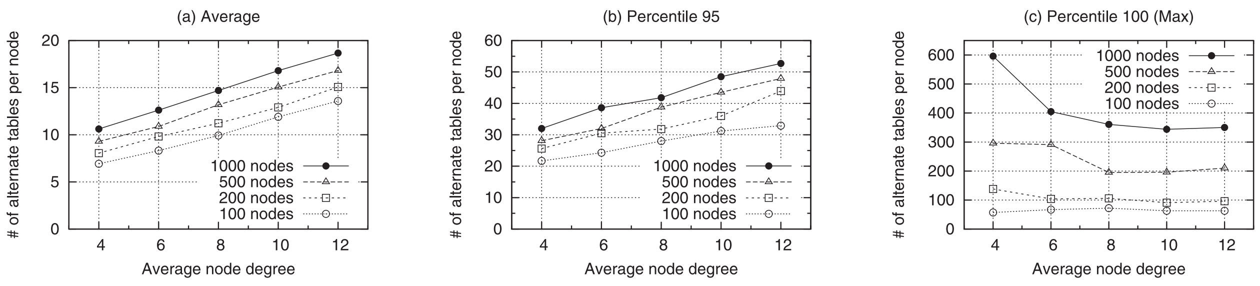


Fig. 6 Number of alternate tables per node. (100–1000 nodes; average node degree of 4–12)

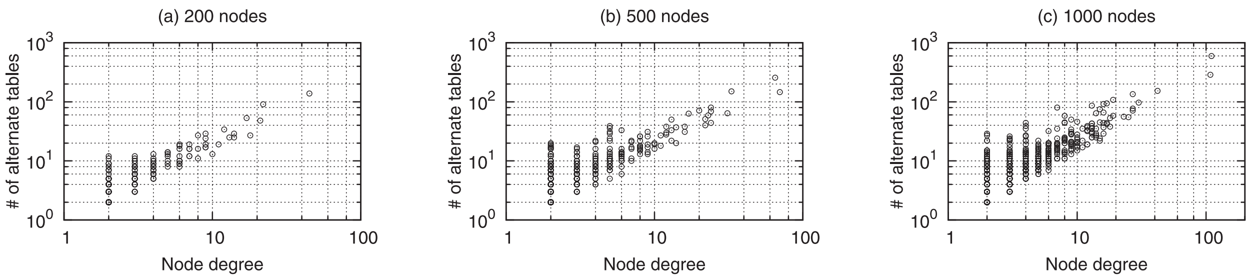


Fig. 7 Relationship between number of alternate routing tables and node degree. (average node degree of 4)

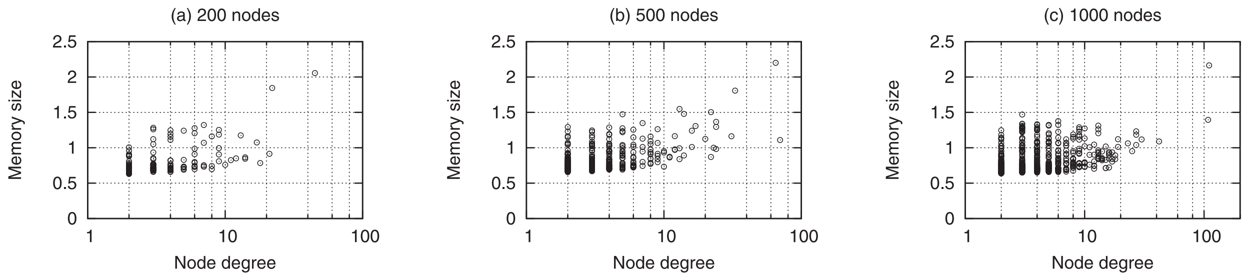


Fig. 8 Memory size for alternate routing tables. (average node degree of 4)

nodes require less than 50 alternate routing tables. In short, only a few nodes need many alternate routing tables.

Figure 7 shows the characteristics of such nodes under the condition that the average node degree is 4. As shown in Fig. 7, as the node degree increases, the number of alternate routing tables increases. In an actual network, nodes with a large degree correspond to core routers and nodes with a small degree correspond to edge routers. This means that core routers tend to require more alternate routing tables. As shown in Fig. 7(c), a core router node with 110 degrees requires 596 alternate routing tables. In the following sections, we discuss that calculating and storing these alternate routing tables are practicable for general IP routers.

4.3 Memory Size

Figure 8 shows the relationship between the memory size required to store alternate routing tables on each node and the node degree. The units on the vertical axis represent the sizes relative to a normal routing table.

Since the alternate routing table stores only changes required for a link failure, they do not need to be full size. Thus, they require memory of less than 2.5 times that of a normal routing table. For example, the total size of alternate routing tables for a core router, i.e. a node with 100 degrees, in a network of 1000 nodes is only 76 kB (35 kB for a normal routing table[†] × 2.2). Those for an edge router (with the degree assumed to be 2) and a middle-range router (with the degree assumed to be 10) are respectively 44 kB and 45 kB. If we assume DRAM memory is used to store the alternate routing tables, this size does not make any practical problem in its implementation.

4.4 Calculation Cost

We estimated the cost of calculating alternate routing tables using the incremental SPF algorithm for networks with 200, 500 and 1000 nodes when the average degree was 4. Figure 9 shows the relationship between cost and the node degree, where the units for cost represent the cost of a full SPF calculation.

The frequency of SPF calculation is the same as the

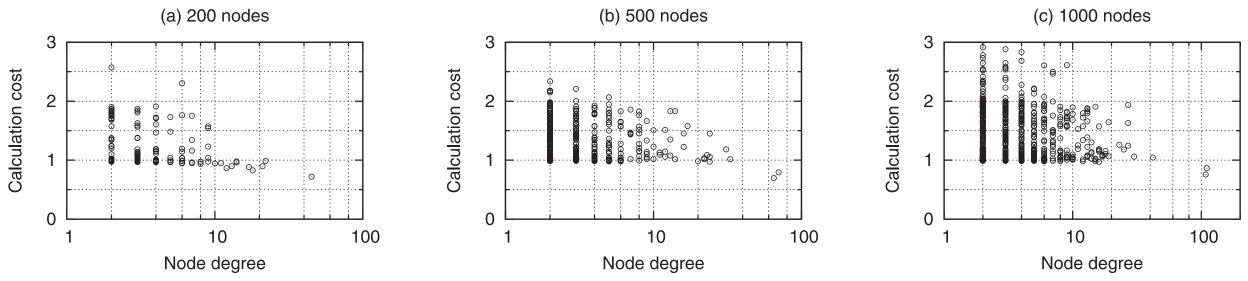


Fig. 9 Cost of computing alternate routing tables. (average node degree of 4)

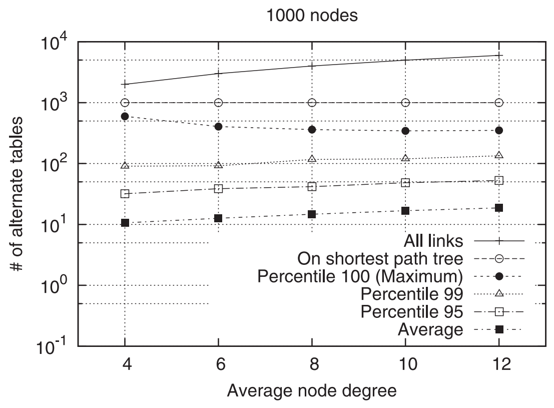


Fig. 10 Advantages over conventional method.

number of alternate routing tables. However, the cost of each calculation can be reduced by using the incremental SPF algorithm [8]. Barbehenn [17] reported that the complexity of Dijkstra's algorithm, on which the SPF calculation is based, is $O(N + L \log N)$ where N and L are the number of nodes and links. Since the incremental SPF algorithm reconstructs only the difference from the original shortest path tree, and the number of nodes and links in the difference is fewer than that of all nodes and links in the network, it can reduce the calculation cost.

As shown in Fig. 9, the cost for the largest degree nodes is less than 1. Since nodes with a large degree, i.e. core routers, have many links, a single-link failure has a relatively small effect on their routing tables. Thus, the use of the incremental SPF algorithm can reduce the cost of calculation.

However, the cost of calculation is relatively large for small degree nodes because the numbers of nodes and links considered by the incremental SPF algorithm are relatively large. However, the total cost of alternate calculation was within three times that of full SPF calculation in all cases.

The maximum values does not differ greatly, regardless of network size. Thus, calculating alternate routing tables for routers with our proposed method does not make problems in term of calculation cost.

4.5 Advantages over Conventional Method

Figure 10 compares the proposed method with the conventional one, i.e., the *on shortest path tree* method. It shows the relationship between the number of alternate routing tables and the average node degree.

The *on shortest path tree* method makes alternate routing tables for each link failure on the shortest path tree. Since the number of links in the shortest path tree is the number of nodes in the network minus 1, the number of alternate routing tables prepared by the *on shortest path tree* method is 999 for the example shown here.

In contrast, the number on average with our proposed method is 1/50 to 1/100 that with the *on shortest path tree* method. Furthermore, the maximum values with our method are less than 60% of those with the *on shortest path tree* method. Therefore, our method needs far fewer alternate routing tables than the *on shortest path tree* method.

Table 2 Results for various topologies.

Name	Network size		# of alternate tables per node		
	Nodes	Deg.	Min.	Ave.	Max.
GÉANT2	32	3.3	0(0%)	3.8(7.3%)	12(23%)
Internet2	11	2.5	2(14%)	4.5(32%)	7(50%)
SINET3	12	2.3	4(29%)	6.3(45%)	10(71%)
BA(10)	10	3.4	2(12%)	3.7(22%)	7(41%)
BA(30)	30	3.8	2(3.5%)	4.8(8.4%)	19(33%)

4.6 Results on Real Network

Table 2 compares the number of alternate routing tables for actual networks, i.e., GÉANT2, Internet2, and SINET3, with those of BA topologies using similar node degrees.

As shown in the table, the smaller the average node degree, the smaller the number of alternate routing tables. This is consistent with the results shown in Fig. 6. Table 2 also shows that the results for actual networks are similar to those for networks generated by BA topologies with similar parameters. Thus, these results support our assertions described in the discussion above.

5. Discussion and Future Work

5.1 Node Failure

Our discussion so far has focused on link failure in this

[†]To store an IPv6 route, 16 bytes each are needed for the prefix and gateway address, 1 byte is needed for the prefix length field, and 2 bytes are needed for the interface index field for each node of the 1000 nodes.

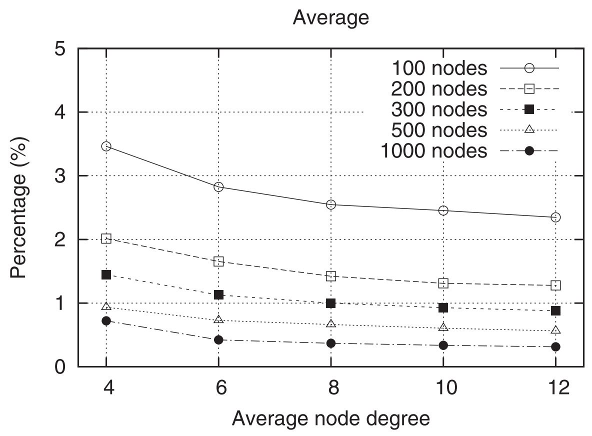


Fig. 11 Percentage of nodes selected with proposed method. (average)

paper, not node failure. It is generally difficult to determine which type of failure has occurred without the failure-detecting function of the lower layer. If the layer 2 network, i.e., Ethernet, does not have this function, the keepalive-based failure detection mechanism of layer 3, i.e., OSPF Hello or BFD [18], is used to report failures. In such cases, link failures may be treated as node failures because these failures cannot be distinguished.

Our proposed method can be modified to handle node failures. 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. To modify the enumerating algorithm for node failure, we can use $\mathcal{T}'_r$ calculated from the following instead of line 2 in Fig. 5.

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

Evaluation and discussion of this case remain for future work.

5.2 Failure Notification

Pei et al. [19] reported the importance of fast calculation of alternate routing tables for quick network convergence. They also reported the importance of fast notification of failure.

Our proposed method reduces the time required to calculate alternate routing tables. However, the underlying concept can also be applied to fast notification. The time for failure notification may be reduced by sending the notification directly to selected nodes rather than using a standard flooding mechanism. If node n_r detects a failure in connecting link e, for example, it can use $\mathcal{V}_u$ to send a message requesting switching to an alternate routing table.

Figure 11 plots the average percentage of nodes that receive a failure notification under this scheme. On average, a single-link failure affects only a small percent of the nodes in the network. This percentage decreases as the average node degree increases. When the average node degree increases, the number of links increases, so the possibility of each link being on the shortest path tree decreases. Then, the percentage of nodes that receive a failure notification decreases.

In summary, these results show that it is sufficient for only a few nodes to be notified of a failure for the network to be restored. This is even more so for large networks due to their many links and nodes. The details of a direct notification mechanism remain for future work.

5.3 Equal-Cost Multi-Path

In actual IP networks, there may be more than one shortest path. The *equal-cost multipath (ECMP)* [4] is used in such cases. ECMP can be supported by extending our algorithm. Since steps 8 to 13 in our algorithm (Fig. 5) examine each node on the shortest path with respect to criterion 1, all we have to do is apply these steps to each shortest path.

Since we used networks without ECMP in the evaluation described in Sect. 4, ECMP may affect our evaluation results. For example, as the number of ECMP increases, the possibility that a node is on a shortest path increases. Therefore, each node may require more alternate routing tables than were given by our evaluation. This depends on the number of shortest paths, the network topology, etc. The quantitative evaluation of this remains for future work.

5.4 Deployment Issue

Assuming the use of the conventional IP forwarding mechanism, we described a method that can be used to create loop-free alternate routes. This method does not require any change of the forwarding mechanism. Note that the routes calculated by the normal router after a failure are the same routes created by our method. Thus, routers supporting our method can be in the same network as ones that do not. It is possible to gradually replace the normal routers with ones supporting our method. Transient loops, which degrade packet reachability, may occur between the two types of routers. This is because these two types of routers update the routes at different times after a failure. However, transient loops can occur regardless of whether or not our method is deployed. The time until the end of the transient loops is not increased by the gradual deployment of our method. The occurrence of transient loops is not a disadvantage of our method.

6. Conclusion

Our proposed method reduces the number of alternate routing tables for IP fast restoration. It does this by analyzing three types of shortest path trees.

The proposed method has three particular advantages.

- The number of alternate routing tables can be reduced to 1/100, on average, from that the conventional method.
- Both the calculation cost and memory size are small enough to enable it to be implemented in general IP routers.
- Since the proposed method does not require any change

of the forwarding mechanism, it can be deployed gradually in actual working networks.

We also described 1) a method based on the proposed approach for attaining faster failure notification and 2) a method for handling node failures. However, in-depth discussion related to these methods remains for future work.

Acknowledgments

This work was partly supported by the Ministry of Internal Affairs and Communications, Japan.

References

[1] J. Manchester, D. Saha, and S. Tripathi, "Protection, restoration, and disaster recovery," IEEE Netw., vol.18, no.2, pp.3–4, 2004.
[2] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of link failures in an IP backbone," Proc. ACM SIGCOMM Workshop on Internet Measurement, pp.237–242, 2002.
[3] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot, "Characterization of failures in an IP backbone," Proc. IEEE INFOCOM 2004, vol.4, pp.2307–2317, 2004.
[4] J. Moy, "OSPF Version 2," RFC 2328, Internet Engineering Task Force, 1998.
[5] D. Oran, "OSI ISIS Intradomain routing protocol," RFC 1142, Internet Engineering Task Force, 1990.
[6] G. Iannaccone, C. Chuah, S. Bhattacharyya, and C. Diot, "Feasibility of IP restoration in a tier 1 backbone," IEEE Netw., vol.18, no.2, pp.13–19, 2004.
[7] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second IGP convergence in large IP networks," ACM SIGCOMM Computer Communication Review, vol.35, no.3, pp.35–44, 2005.
[8] J.M. McQuillan, I. Richer, and E.C. Rosen, "The New Routing Algorithm for the ARPANET," IEEE Trans. Commun., vol.COM-28, no.5, pp.711–719, 1980.
[9] 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.
[10] S. Nelakuditi, S. Lee, Y. Yu, Z. Zhang, and C. Chuah, "Fast local rerouting for handling transient link failures," IEEE/ACM Trans. Netw., vol.15, no.2, pp.359–372, 2007.
[11] K. Suzuki and M. Jibiki, "Formalization and analysis of routing loops by inconsistencies in IP forwarding tables," IEICE Trans. Commun., vol.90, no.10, pp.2755–2763, 2007.
[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] A.L. Barabasi and R. Albert, "Emergence of scaling in random networks," Science, vol.286, pp.509–512, 1999.
[14] "GÉANT2 Website." http://www.geant2.net/
[15] "The Internet2 Network." http://www.internet2.edu/
[16] "SINET3." http://www.sinet.ad.jp/topology/
[17] M. Barbehenn, "A Note on the complexity of dijkstra's algorithm for graphs with weighted vertices," IEEE Trans. Comput., vol.47, no.2, p.263, 1998.
[18] D. Katz and D. Ward, "Bidirectional forwarding detection," draft-ietf-bfd-base-09, Internet Engineering Task Force, 2009.
[19] D. Pei, L. Wang, D. Massey, S. Wu, and L. Zhang, "A study of packet delivery performance during routing convergence," Proc. IEEE International Conference on Dependable Systems and Networks (DSN), pp.183–192, 2003.

Appendix: Proof of Criteria

Our proposed method makes only nodes satisfying both Criterion 1 and Criterion 2 for n_d update a route to n_d on their own routing tables after a failure. Assume that a link in the network has failed. We prove that it is sufficient to update the route of these nodes to restore the packet reachability in the network.

We discuss the case where a specific node, n_d, is the destination node. Let e denote a failed link. Let $\mathcal{V}$ be the set of all nodes in the network. It is then divided into three subsets that are pairwise disjointed.

- Let $\mathcal{V}_u^{(d)}$ be a set of nodes that satisfy both criterion 1 and 2 for n_d.
- Let $\mathcal{V}_a^{(d)}$ be a set of nodes that satisfy only criterion 1 for n_d, not criterion 2.
- Let $\mathcal{V}_b^{(d)}$ be a set of nodes that are not included in either $\mathcal{V}_u^{(d)}$ or $\mathcal{V}_a^{(d)}$.

With our proposal method, only nodes in $\mathcal{V}_u^{(d)}$ need to update the route to n_d in their routing table after the failure of e. Figure A· 1 illustrates the relationships among these sets.

Now, let us assume the network that is a connected graph when e fails. This means that the failure of e does not divide the network into two parts.

Lemma 1: n_d is an element of $\mathcal{V}_b^{(d)}$.

Proof 1: Because n_d does not satisfy criterion 1 for n_d, n_d is a element of $\mathcal{V}_b^{(d)}$. □

Lemma 2: Let n_r be an end node of link e, where the shortest path from n_r to n_d includes e. Then, n_r is an element of $\mathcal{V}_u^{(d)}$.

Proof 2: According to the definition of n_r, n_r satisfies criterion 1 for n_d. It is trivial that n_r satisfies criterion 2 for n_d. Therefore, n_r is an element of $\mathcal{V}_u^{(d)}$. □

We prove three lemmas for packets starting from any node in $\mathcal{V}_u^{(d)}$, $\mathcal{V}_a^{(d)}$, or $\mathcal{V}_b^{(d)}$.

Lemma 3: A packet that starts from any node in $\mathcal{V}_b^{(d)}$ to n_d can reach the destination node, n_d.

Proof 3: According to lemma 1, n_d is an element in $\mathcal{V}_b^{(d)}$.

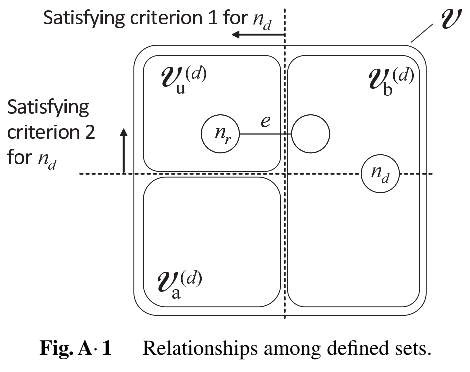


Fig. A· 1 Relationships among defined sets.

A shortest path from any node in $\mathcal{V}_b^{(d)}$ to n_d does not include e, because all nodes in $\mathcal{V}_b^{(d)}$ do not satisfy criterion 1 for n_d. Briefly, the failure of e do not change this shortest path. Therefore, a packet that starts from any nodes in $\mathcal{V}_b^{(d)}$ can reach n_d. □

Lemma 4: A packet that starts from any node in $\mathcal{V}_u^{(d)}$ to n_d can reach a node in $\mathcal{V}_b^{(d)}$.

Proof 4: We discuss a shortest path from any node in $\mathcal{V}_u^{(d)}$ to n_d after the failure of e. Since all nodes in $\mathcal{V}_u^{(d)}$ update routes to n_d, a node in $\mathcal{V}_a^{(d)}$ forwards a packet destined to n_d along this shortest path tree. Therefore, a routing loop between nodes in $\mathcal{V}_u^{(d)}$ does not occur.

We show that this shortest path does not include any node in $\mathcal{V}_a^{(d)}$. According to lemma 2, n_r is in $\mathcal{V}_u^{(d)}$. Because all nodes in $\mathcal{V}_u^{(d)}$ are on the shortest path from n_r to n_d, the shortest path from any node in $\mathcal{V}_u^{(d)}$ to n_d is part of the shortest path from n_r to n_d. According to the definition of $\mathcal{V}_a^{(d)}$, all nodes in $\mathcal{V}_a^{(d)}$ do not satisfy criterion 2. If nodes are on the shortest path from n_r to n_d, they satisfy criterion 2. Therefore, all nodes on the shortest path from any node in $\mathcal{V}_u^{(d)}$ to n_d are in $\mathcal{V}_u^{(d)}$ or $\mathcal{V}_b^{(d)}$, not $\mathcal{V}_a^{(d)}$.

Therefore, a packet that starts from any node in $\mathcal{V}_u^{(d)}$ can reach a node in $\mathcal{V}_b^{(d)}$. □

Lemma 5: A packet that starts from any node in $\mathcal{V}_a^{(d)}$ to n_d can reach a node in $\mathcal{V}_u^{(d)}$ or $\mathcal{V}_b^{(d)}$.

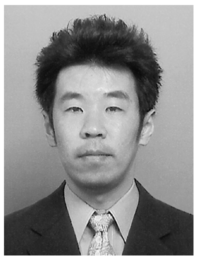
Proof 5: Since all nodes in $\mathcal{V}_a^{(d)}$ do not update routes to n_d, a node in $\mathcal{V}_a^{(d)}$ forwards a packet destined to n_d along the shortest path to n_d before the failure. Therefore, a routing loop between nodes in $\mathcal{V}_a^{(d)}$ does not occur. According to lemma 1, $\mathcal{V}_a^{(d)}$ does not include n_d.

Briefly, a packet that starts from any node in $\mathcal{V}_a^{(d)}$ can reach a node in $\mathcal{V}_u^{(d)}$ or $\mathcal{V}_b^{(d)}$. □

We can prove the following theorem using the lemmas above.

Theorem 6: The packet reachabilities for all pair nodes in the network can be guaranteed by only updating the routing tables of the nodes in $\mathcal{V}_u$.

Proof 6: According to lemma 5, a packet that starts for n_d from any node in $\mathcal{V}_a^{(d)}$ can reach a node in $\mathcal{V}_u^{(d)}$ or $\mathcal{V}_b^{(d)}$. According to lemma 4, a packet that starts for n_d from any node in $\mathcal{V}_u^{(d)}$ can reach a node in $\mathcal{V}_b^{(d)}$. According to lemma 3, a packet that starts for n_d from any node in $\mathcal{V}_b^{(d)}$ can reach the destination node. Therefore, a packet which starts from any node in the network can reach the destination node after a failure. □



Kazuya Suzuki received an M.E. degree in Electrical Engineering from Tokyo Metropolitan University. He joined NEC Corporation in 1997, where he has been undertaking developing network equipment. He is currently with the System Platforms Research Laboratories, NEC Corporation, and is a doctoral candidate at the University of Tsukuba, Tokyo, Japan. His research interests include availability improvement in unicast and multicast routing.

Masahiro Jibiki received the Ph.D. degree in systems management from the University of Tsukuba, Tokyo, Japan. He is currently a Researcher in Central Research Laboratories, NEC Corporation, and is also Visiting Professor in the University of Wakayama, Japan, from 2006. His research interests include networking, distributed system and software science.

Kenichi Yoshida received his Ph.D. from Osaka University in 1992. In 1980, he joined Hitachi Ltd., and is working for University of Tsukuba from 2002. His current research interest includes application of Internet and application of machine learning techniques.