APSIPA Transactions on Signal and Information Processing, 2023, 12, e9

Original Paper

Design of Multiple Routing Configurations Considering Load Distribution for Network Slicing

Takeru Misugi[1], Hideyoshi Miura[1], Kouji Hirata[2*] and Takuji Tachibana[3]

[1] *Graduate School of Science and Engineering, Kansai University, Osaka, Japan*

[2] *Faculty of Engineering Science, Kansai University, Osaka, Japan*

[3] *Graduate School of Engineering, University of Fukui, Fukui, Japan*

ABSTRACT

In recent years, the network slicing technology has attracted much attention because it can provide virtual networks called slices according to service requirements. This paper proposes a load-balanced fast failure recovery method based on Multiple Routing Configurations (MRC) for network slicing environments. MRC ensures the availability of routing paths for any possible single link/node failures by preparing multiple backup routing configurations corresponding to the failures. MRC can construct backup routing configurations for physical networks, but it does not consider network slicing environments. The proposed failure recovery method extends the concept of MRC. In the proposed method, dedicated backup routing configurations are constructed for each slice only with necessary physical nodes and links, instead of using all the physical nodes and links. By doing so, we expect to avoid making inefficient detour paths. Through numerical experiments, we show the effectiveness of the proposed method. In addition, we implement our proposed method with Programming Protocol-Independent Packet Processors (P4) in

*Corresponding author: Kouji Hirata, hirata@kansai-u.ac.jp. This research was supported by SCOPE of the Ministry of Internal Affairs and Communications, Japan, under Grant No. 191605004.

Received 28 July 2022; Revised 21 November 2022

ISSN 2048-7703; DOI 10.1561/116.00000148

software-defined networking environments. We conduct demonstration experiments using Mininet to confirm our P4-based implementation.

Keywords: Network slicing, failure recovery, multiple routing configurations, P4

1 Introduction

In recent years, network slicing has attracted much attention as a technology that meets the requirements for various types of Internet services [1, 12]. The network slicing technology virtually divides a communication network into multiple networks (called slices), each of which is independent of each other. Each slice is allocated a certain amount of network resources such as link bandwidth and switch capacity. This slicing technology enables different types of services, e.g., high-speed, high-capacity services and high-reliability, low-latency services, to be provided on a common physical network [32]. Because of these features, network slicing is expected to be used in services that require high-quality communications.

In communications using the network slicing technology, in order to provide high-quality services to users, it is necessary to continue the services of each slice in the case of failures. In traditional IP networks, when the network states change due to a network failure, the routing table of each router is dynamically modified by a traditional routing protocol such as Open Shortest Path First (OSPF) [13]. In such a routing protocol, the routing table is modified by exchanging network state information among routers. The convergence time during which the routing table is unstable is long. Thus a lot of packets forwarded during this period could be dropped. In order to resolve this problem, the Multiple Routing Configurations (MRC) algorithm has been introduced [17]. MRC is a fast failure recovery mechanism for a single link/node failure of a network. MRC ensures path availability in the case of a link/node failure by preparing multiple backup routing configurations in advance. In each backup routing configuration, some nodes and links are assumed to be failed. These nodes and links are called isolated nodes and links, respectively. Although the isolated nodes and links are not used for data transmission after a failure occurrence on the backup routing configuration, the path availability between each node pair is ensured by other nodes and links. When a link/node failure is detected, MRC immediately switches a normal routing configuration to a backup routing configuration in which the failed point is selected as an isolated link/node. As a result, MRC can continue data transmission. In MRC, each link/node is treated as an isolated link/node in one of backup routing configurations to ensure failure recovery from any single link/node failures.

MRC does not use isolated links and nodes as routing paths on backup routing configurations. When MRC is applied to a large-scale network, the number of isolated links/nodes unused for routing paths in a backup routing configuration becomes large. This feature can make inefficient detour paths that cause an increase in the number of hops and unbalanced load on network links leading to network congestion. Furthermore, this problem becomes pronounced in the case of applying MRC to network slicing environments where there are multiple slices on a physical network. When common backup routing configurations are constructed for the slices, many unnecessary physical links and nodes are included in backup routing paths for the slices even though the size of the slices is small. Therefore, the length of some backup routing paths could become long, compared with the size of the slices. Furthermore, in the case of failures, each slice tends to use the same backup routing paths, which concentrates load on specific links and nodes. Therefore, it is preferred that we construct different backup routing configurations for respective slices. However, there are no works that discuss how to make backup routing configurations of MRC for network slicing environments, taking load distribution into account.

In this paper, we propose a load-balanced fast failure recovery method based on MRC for network slicing environments. The proposed method extends the concept of MRC to construct backup routing configurations for slices. In the proposed method, dedicated backup routing configurations are constructed for each slice only with necessary physical nodes and links, instead of using all the physical nodes and links. By doing so, the number of isolated links and nodes per routing configuration is reduced. Thus, we expect to avoid making inefficient detour paths for each slice, which suppresses network congestion after a failure occurrence. The proposed method provides an algorithm to construct backup routing configurations for each slice with the use of minimum necessary physical links and nodes. Through numerical experiments, we show the effectiveness of the proposed method. In addition, we implement our proposed method with Programming Protocol-Independent Packet Processors (P4) [6] in Software-Defined Networking (SDN) environments. P4 is a programming language for SDN and it allows us to flexibly define the functions of network devices. By confirming the operation using Mininet [20], we show that routing paths are ensured by switching routing configurations in the case of a single link/node failure.

This paper is an extended version of our conference paper [21], where we have provided the concept of our MRC design for network slicing. In the conference paper, we have introduced a simple method to apply the MRC algorithm using common backup routing configurations for each slice and implemented its basic operation with the use of P4. In this present paper, we extend our work by providing the algorithm to construct backup routing paths for each slice. Furthermore, we conduct numerical experiments for examining the performance of our proposed method in more detail.

The contributions of this paper are as follows:

- We introduce the algorithm to design backup routing configurations for network slicing environments. The algorithm enables us to construct dedicated backup routing configurations for each slice with the use of minimum necessary physical nodes and links.
- Through numerical experiments, we show the effectiveness of the proposed method. Specifically, the proposed method alleviates unbalanced load on network links, which is expected to avoid network congestion after a failure occurrence.
- We provide an example of the implementation of our proposed method with P4. Through demonstration experiments, we confirm that our proposed method ensures the path availability in the case of a failure.

The rest of this paper is organized as follows. Section 2 discusses related works. In Section 3, we explain the outline of network slicing and MRC. Section 4 explains our proposed method. In Section 5, we examine the performance of our proposed method through numerical experiments. In Section 6, we conduct demonstration experiments by implementing the proposed method with P4. We state the conclusion of this paper in Section 7.

2 Related Works

Failure recovery methods switch normal routing paths to backup routing paths that do not pass through failed points after failures occur. They are categorized into reactive and proactive methods according to ways to switch routing paths [3, 27]. Reactive methods calculate backup routing paths after failures occur, without preparing backup routing paths in advance. For instance, SDN controllers calculate backup routing paths according to the failure in SDN environments. Thus, network nodes do not need to maintain backup routing paths. Although the reactive methods can reduce the number of routing entries to be stored in network nodes, they require more recovery time than proactive methods. On the other hand, proactive methods prepare backup routing paths in advance. When failures occur, the proactive methods immediately switch normal routing paths to backup routing paths. Therefore, the proactive methods can quickly recover from failures, but they need to store many routing entries in network nodes to maintain backup routing paths. MRC is a proactive method because it prepares backup routing configurations in advance.

In the past, various methods for recovering from link/node failures in SDN environments including network slicing have been proposed. Chu *et al.* [10] have proposed a failure recovery method for hybrid SDNs in which traditional IP

routers and SDN switches co-exist. When a failure occurs, IP routers redirect traffic flows to SDN switches along pre-configured tunneling paths in order to bypass the failed point. Lin *et al.* [19] have introduced a recovery method that periodically collects information about the global network topology using link layer discovery protocol packets. Based on this information, a controller proactively establishes backup routing paths, which are installed on network nodes. Wang *et al.* [30] have presented a ring-based failure recovery method to reduce the number of routing entries in SDN switches. It constructs a ring in such a way that each backup routing path includes a part of the ring. Baumgartner *et al.* [4] have introduced a mathematical model for a general network slice design problem. They have provided network slice protection against single node/link failures to add survivability aspects to their model. Wen *et al.* [31] have investigated robust network slicing mechanisms by addressing the slice recovery and reconfiguration in a unified framework. They have designed an optimal joint slice recovery and reconfiguration algorithm for stochastic traffic demands by exploiting robust optimization. Mohan and Gurusamy [23] have proposed an approach for resilient network slice embedding such that virtual nodes providing network services are placed on appropriate physical nodes to minimize the number of affected network services in the case of failures. They have developed an optimization programming formulation for the proposed approach. Zhang *et al.* [35] have introduced the Progressive Slice Recovery (PSR) problem that decides the best sequence of repairs during recovery for network slicing environments. They have proposed a comprehensive PSR method achieving fast recovery of slices based on an optimization model.

Furthermore, some implementation of fast failure recovery with P4 have been introduced in the past. Cascone *et al.* [7] have proposed a failure detection and recovery method. They have implemented the failure detection mechanism and the path-based failure recovery mechanism, using OpenState [5] and P4. Sedar *et al.* [26] have discussed P4-based implementation of fast reroute algorithms. They have presented a mechanism that reduces the number of flow entries in SDN switches by applying a wildcard match table to fast reroute algorithms. Also, Chiesa *et al.* [9] have introduced P4-based implementation of a fast-reroute mechanism. It provides fast failure recovery by avoiding packet recirculation in switches for finding active output ports. Xu *et al.* [33] have presented P4-based implementation of a proactive failure recovery method. It creates a custom packet header that contains backup routing path information. When a failure occurs, packets are forwarded to backup routing paths based on the custom packet header. Hirata and Tachibana [16] have designed P4-based implementation of MRC. They have provided some mechanisms to realize MRC in SDN environments. These works have implemented failure recovery mechanisms with P4, but they have not considered network slicing environments. In this paper, we provide P4-based implementation of fast failure recovery for network slicing environments.

3 Outline of Background Technologies

3.1 Network Slicing

Network slicing is a technology that embeds virtual networks called slices into a physical network consisting of physical links and nodes. The physical network into which slices are embedded is referred to as the *substrate network*. A slice consists of virtual nodes and virtual links. A virtual node is placed on one of physical nodes and it implements a specific network functionality such as router and firewall. This can be implemented by the Network Function Virtualization (NFV) technology [28]. A virtual link between two virtual nodes represents communication requirements between them to provide some network services. The virtual link can be realized as a multihop physical path. The structure of slices depends on service requirements of clients.

Figure 1 is an example of the construction of slices on a substrate network. We assume that there are two slices to be embedded into the substrate network. We consider the case where each virtual node is mapped to the physical node having the same number as the virtual node. Each virtual node and link requires a certain amount of network resources, i.e., node capacity and link bandwidth, respectively. In Figure 1, the numbers next to nodes and links represent the amount of required network resources. For example, the amount of required node capacity of virtual node 1 on the red-colored slice is 2 (in a square). Similarly, the amount of required bandwidth of virtual link 1-3 on the red-colored slice is 3. These two slices are embedded into the substrate

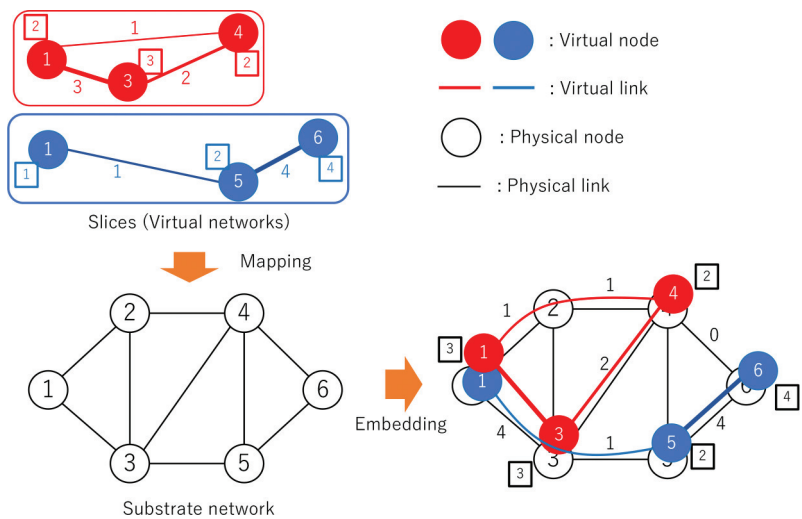


Figure 1: Example of network slicing.

network as shown in the figure. In this example, virtual node 1 of each slice is embedded into the same physical node, and thus the amount of used capacity of physical node 1 is $1 + 2 = 3$. As long as the amount of used capacity is less than the capacity of the physical node, different virtual nodes can be embedded into the physical node. Similarly, the other virtual nodes are embedded into corresponding physical nodes. Virtual links are mapped to physical links. Note that some virtual links consist of multihop physical paths, e.g., virtual link 1–4 of the red-colored slice consists of physical path 1-2-4. The physical path is calculated based on routing algorithms such as the shortest path routing. The physical links along the physical path allocate bandwidth to the virtual link. For instance, on physical link 1–3, the link bandwidth is allocated to virtual links 1–3 and 1–5 of the slices (i.e., $1 + 3 = 4$).

3.2 Multiple Routing Configurations (MRC)

MRC is a technology to realize fast recovery from a single link/node failure in a biconnected network [17]. MRC prepares K ($K > 1$) backup routing configurations on which backup routing paths are established in advance, covering all possible single link/node failures in the network as shown in Figure 2. The value of K is determined according to the size of the network.

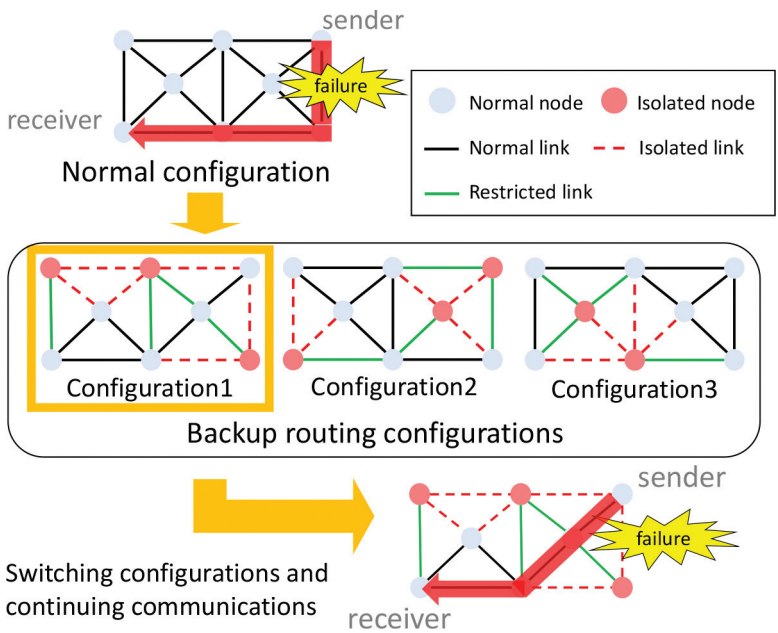


Figure 2: Workflow of MRC.

Note that the value of K is generally much smaller than the number of links and nodes in the network. When a node or link failure occurs along a normal routing path on a normal routing configuration, the previous node of the failed point detects the failure. Then, MRC immediately switches the normal routing configuration to a backup routing configuration covering the failed point. By using an alternate routing path on the backup routing configuration, data transmission can continue without packet losses.

On backup routing configurations, nodes are classified into normal nodes and isolated nodes. Also, links are classified into normal links, isolated links, and restricted links. The normal nodes and the normal links can be used for packet transmission on the backup routing configuration, while the isolated nodes and the isolated links do not carry any traffic. The restricted links are used only for the first hop or the last hop of packet flows, whose source or destination on the backup routing configuration is an isolated node, to enable the packet flows to reach the isolated node.

After a failure occurs, MRC uses a backup routing configuration where the failed point is selected as an isolated link/node. MRC guarantees tolerance for all possible single link/node failures by isolating each link and each node on one of backup routing configurations. On each backup routing configuration, each normal node pair must be connected by a path that does not pass through any isolated links and nodes. Therefore, each backup routing configuration must satisfy the following constraints.

1. Each backup routing configuration includes a connected graph composed of all the normal nodes and the normal links.
2. Each link/node becomes an isolated link/node on one of backup routing configurations.
3. Isolated nodes are connected with isolated or restricted links, while they are not connected with normal links.
4. At least one of links connected to an isolated node is a restricted link.
5. At least one of nodes connected to an isolated link is an isolated node.
6. Nodes connected by a restricted link are a normal node and an isolated node.

Kvalbein *et al.* [17] have introduced the heuristic algorithm to construct backup routing configurations satisfying these constraints. The heuristic algorithm takes as input a biconnected network and the number K of backup routing configurations that are intended created. In this paper, we use the heuristic algorithm to make K backup routing configurations. In what follows, we briefly explain the procedure of the heuristic algorithm. Note that detailed explanations on the heuristic algorithm have been discussed in [17].

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a biconnected substrate network where $\mathcal{V}$ and $\mathcal{E}$ denote sets of physical nodes and links, respectively. Let $v_i \in \mathcal{V}$ denote i-th physical nodes ($i = 1, 2, \ldots, |\mathcal{V}|$). Let $\mathcal{C}$ denote a set of backup routing configurations and $\mathcal{C}_k$ denote the k-th configuration ($k = 1, 2, \ldots, K$). The outline of the MRC algorithm is as follows.

MRC Algorithm:

Input: Biconnected substrate network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the number K of backup routing configurations to be constructed

Output: Set $\mathcal{C} = \{\mathcal{C}_k \mid k = 1, 2, \ldots, K\}$ of backup routing configurations

Step (1) For each $k = 1, 2, \ldots, K$, $C_k \leftarrow \mathcal{G}$. Then, $k \leftarrow 1$ and $i \leftarrow 1$.

Step (2) If configuration $\mathcal{C}_k$ does not satisfy the constraint 1 of MRC discussed above in the case where node v_i is removed from $\mathcal{C}_k$, go to Step (4).

Step (3) If the constraints 2-6 of MRC for each configuration are satisfied after node v_i becomes an isolated node in configuration $\mathcal{C}_k$, the following sub-procedures (3-1)–(3-3) are done; otherwise go to Step (4).

- **(3-1)** In configuration C_k, node v_i becomes an isolated node.
- **(3-2)** In configuration C_k, at least one link of node v_i becomes a restricted link, and the other links of node v_i become isolated links.
- **(3-3)** $i \leftarrow i + 1$. If $i \leq |\mathcal{V}|$, go to Step (2); otherwise, the algorithm terminates.

Step (4) $k \leftarrow (k \mod K) + 1$. If there are no configurations in which node v_i can become an isolated node, the algorithm terminates; otherwise, go to Step (2).

The MRC algorithm sequentially adopts nodes as isolated nodes in backup routing configurations. Step (1) is the initialization. In Steps (2) and (3), a node becomes an isolated node and corresponding links become restricted or isolated links in one of backup routing configurations while satisfying the constraints of MRC. By repeating these steps, the MRC algorithm constructs backup routing configurations. Note that the case where the MRC algorithm terminates in Step (4) means that the algorithm fails to construct K backup routing configurations. In this case, we need to appropriately increment the value of K and apply the MRC algorithm again to be able to construct backup routing configurations.

In MRC, in order to ensure the reachability of all node pairs on each backup routing configuration, a connected sub-graph consisting of all the normal links and the normal nodes must be included in the backup routing configuration as shown in Figure 2. The structure of networks applied MRC needs to be a biconnected graph to meet this condition.

4 Proposed Method

4.1 System Model

Figure 3 represents the system model assumed in this paper. In this paper, we consider the situation where there is a substrate network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and multiple slices are embedded into the substrate network. Let $\mathcal{S}$ denote a set of slices to be embedded. Let $\mathcal{S}_i = (\mathcal{V}_i, \mathcal{E}_i)$ denote the i-th slice, where $\mathcal{V}_i$ and $\mathcal{E}_i$ denote sets of virtual nodes and links constituting the slice, respectively. The slices have different structures as shown in Figure 3, and they are given as the problem input. Each virtual node of a slice is embedded into the corresponding physical node of the substrate network as discussed in Section 3.1. On the other hand, each virtual link of a slice is mapped to a multihop physical path on the substrate network. We assume that the physical path is the shortest path in terms of the number of hops. Each virtual node of a slice communicates with other virtual nodes connected by virtual links, which require a certain amount of link bandwidth along their physical paths as discussed in Section 3.1. We assume that the capacity of each physical node and link is sufficiently large to accommodate all slices. Under these assumptions, the proposed method applies the MRC algorithm to a biconnected graph constructed for each slice.

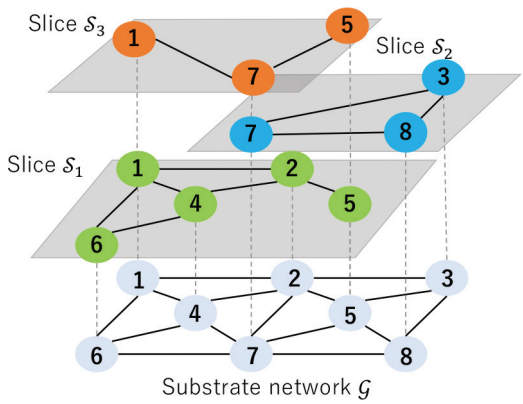


Figure 3: System model.

4.2 Application of MRC to Slices

4.2.1 Background and Overview

In this paper, we examine how to construct efficient backup routing configurations in network slicing environments. When applying MRC to a large-scale network, each backup routing configuration includes a lot of isolated links

and isolated nodes which cannot be used for data transmission. As a result, inefficient long paths could be established on the backup routing configuration as shown in Figure 4. In this figure, we consider a flow from node 8 to node 13. In the normal routing configuration, the shortest path of the flow is 8-11-13 whose hop count is 2. On the other hand, the shortest path of the backup routing configuration is 8-6-7-10-12-13 whose hop count is 5, which is much larger than that of the normal routing configuration.

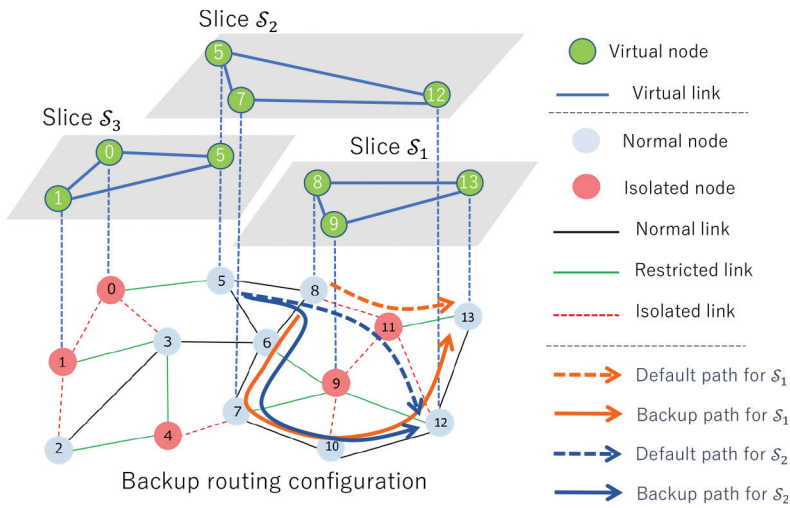


Figure 4: Common backup routing configuration for slices.

This problem of MRC becomes pronounced in network slicing environments where there are multiple slices on the substrate network. When each slice uses common backup routing configurations constructed based on the whole substrate network, inefficient detour paths could be established, regardless of the size of slices. As shown in Figure 4, the size of slice $\mathcal{S}_1$ is relatively small, but the backup routing path (i.e., 8-6-7-10-12-13) from node 8 to node 13 becomes long in the case of the failure discussed above. Furthermore, in the case where multiple slices use the same backup routing path, traffic concentrates on specific links and nodes. For example, in Figure 4, we assume that node 5 communicates with node 12 on slice $\mathcal{S}_2$. Its routing path on the backup routing configuration is 5-6-7-10-12. In this case, the flows of slices $\mathcal{S}_1$ and $\mathcal{S}_2$ pass through many common links and nodes, which could cause network congestion. Therefore, it is preferred that we construct different backup routing configurations for respective slices. Figure 5 shows an example of making dedicated backup routing configurations for each slice. In this example, each backup routing configuration is constructed with a sub-graph consisting of a small number of physical links and nodes, instead of using all the physical links

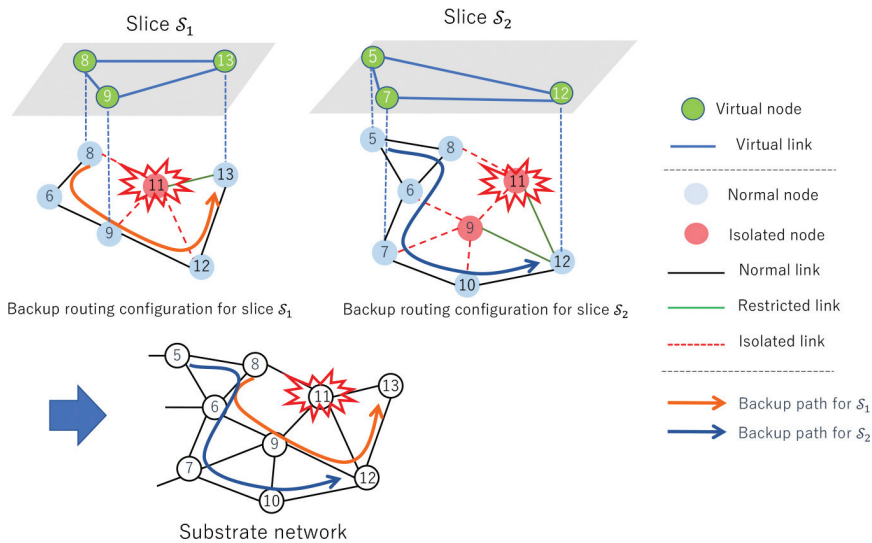


Figure 5: Construction of dedicated backup routing configurations for respective slices.

and nodes of the substrate network. As a result, for slice $\mathcal{S}_1$, the backup routing path from node 8 to node 13 is 8-6-9-12-13, the length of which is smaller than that in the case of using a common backup routing configuration shown in Figure 4. Furthermore, by using dedicated backup routing configurations for respective slices, backup routing paths tend to pass through different links and nodes. As a result, we expect to distribute the load on network links.

However, there is a case where a sub-graph of the substrate network corresponding to a slice is not a biconnected graph. Therefore, MRC cannot be directly applied to the sub-graph. To resolve this problem, the proposed method in this paper constructs a biconnected graph by adding minimum necessary components, i.e., physical links and nodes, to the sub-graph of each slice. To do so, the proposed method detects nodes that are articulation points in the sub-graph corresponding to the slice. A node in a network is referred to as an *articulation point* if the network becomes disconnected by removing the node, as shown in Figure 6. For example, in this figure, the red-colored nodes are articulation points. When any of the nodes are removed, the graph is divided into two graphs. We can construct a biconnected graph by eliminating the articulation points. By applying MRC to the biconnected graph, we can reduce the size of backup routing configurations for the slice. As a result, we expect to make efficient detour paths for each slice, which suppresses network congestion after a failure occurrence.

As discussed above, MRC ensures the path availability between each node pair for any possible single link/node failures by preparing K backup routing

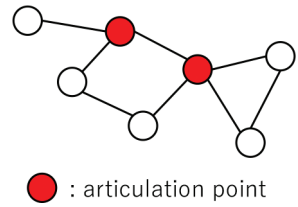


Figure 6: Articulation point.

configurations. When a link or node failure occurs, MRC selects the backup routing configuration where the link or node is isolated. These backup routing configurations are shared by each node. Each node maintains common routing entries for the backup routing configurations, and thus MRC can realize fast recovery within a few tens of milliseconds [17, 22]. In the proposed method, we apply this concept to network slicing environments. Specifically, each node maintains backup routing entries for each slice in order to ensure the path availability for any possible single link/node failures on the sub-graph for the slice. Therefore, we prepare K backup routing configurations for each slice. In what follows, we explain the procedure of the proposed method in detail.

4.2.2 Articulation Point Detection

A biconnected graph has no articulation points. Thus the graph is connected even after any one of the nodes is removed [2]. We can determine whether a target network is a biconnected graph by detecting articulation points. As a detection method, Depth First Search (DFS) is generally used [11, 18].

DFS constructs a spanning tree T called DFS tree on the network. If node v of the DFS tree T meets the following condition A or B, v is an articulation point, where v_r denotes the root node of T.

A. In the case of $v = v_r$, the number of child nodes of v_r is greater than or equal to 2.

B. In the case of $v \neq v_r$, v has a child node v_c such that there are no back edges, which are not included in T, from v_c or any descendant nodes of v_c to any ancestor nodes of v.

Figure 7 illustrates an example of articulation point detection using DFS. In this figure, nodes 1 and 3 are articulation points. Node 1 is the root node and the number of child nodes is 2 (i.e., nodes 2 and 5), which meets the condition A. On the other hand, node 3 is not the root node. It has a child node (i.e., node 4) and there are no back edges from node 4 to nodes 1 or 2 on the original network. Thus node 3 meets the condition B. Actually, if node 1 or 3 is removed, the network becomes disconnected.

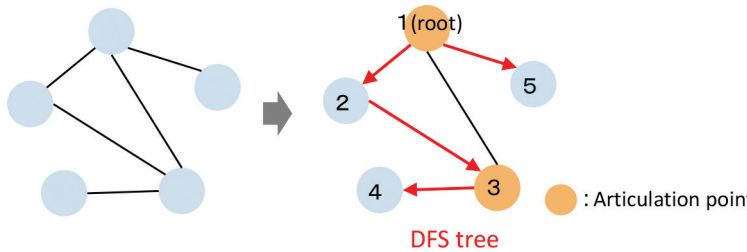


Figure 7: Articulation point detection using DFS.

4.2.3 Procedure of the Proposed Method

We here discuss the detailed procedure of the proposed method to construct backup routing configurations for each slice on the substrate network. Let $\mathcal{B}(\mathcal{S}_i) = (\mathcal{V}_B(\mathcal{S}_i), \mathcal{E}_B(\mathcal{S}_i))$ denote a biconnected graph to be constructed for slice $\mathcal{S}_i$, where $\mathcal{V}_B(\mathcal{S}_i)$ and $\mathcal{E}_B(\mathcal{S}_i)$ denote sets of nodes and links, respectively, included in the biconnected graph. Let $\mathcal{N}(p)$ and $\mathcal{L}(p)$ denote sets of physical nodes and links, respectively, which are included in physical path p. The procedure of the proposed method is as follows.

Configuration Construction Algorithm:

Input: Substrate network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and set $\mathcal{S} = \{\mathcal{S}_i = (\mathcal{V}_i, \mathcal{E}_i) \mid i = 1, \ldots, |\mathcal{S}|\}$ of slices

Output: Backup routing configurations for each slice $\mathcal{S}_i \in \mathcal{S}$

Step (1) $i \leftarrow 1$.

Step (2) $\mathcal{V}_B(\mathcal{S}_i) \leftarrow \emptyset$ and $\mathcal{E}_B(\mathcal{S}_i) \leftarrow \emptyset$.

Step (3) For each virtual link $e_s \in \mathcal{E}_i$, physical nodes and links included in the physical path p_{e_s} corresponding to e_s are added to $\mathcal{B}(\mathcal{S}_i)$. Specifically, $\mathcal{V}_B(\mathcal{S}_i) \leftarrow \mathcal{V}_B(\mathcal{S}_i) \cup \mathcal{N}(p_{e_s})$ and $\mathcal{E}_B(\mathcal{S}_i) \leftarrow \mathcal{E}_B(\mathcal{S}_i) \cup \mathcal{L}(p_{e_s})$.

Step (4) Articulation points are detected by applying the DFS algorithm, in which the root node is randomly selected, to $\mathcal{B}(\mathcal{S}_i)$. If an articulation point is selected as the root node, the DFS algorithm using another node as the root node is applied to $\mathcal{B}(\mathcal{S}_i)$ again. Until an articulation point is not selected as the root node, this operation is repeated.

Step (5) If there are no articulation points, $i \leftarrow i + 1$. If $i > |\mathcal{S}|$, it goes to Step (10); otherwise, it goes to Step (2).

Step (6) For each articulation point v, the length $l(v_c, v_a)$ of the shortest path from its child node v_c to each ancestor node v_a of v on the substrate network without using the DFS tree is calculated.

Step (7) The pair of v_c and v_a with the smallest value of $l(v_c, v_a)$ is selected, and then its path p is added to $\mathcal{B}(\mathcal{S}_i)$ to eliminate the articulation point. Specifically, $\mathcal{V}_B(\mathcal{S}_i) \leftarrow \mathcal{V}_B(\mathcal{S}_i) \cup \mathcal{N}(p)$ and $\mathcal{E}_B(\mathcal{S}_i) \leftarrow \mathcal{E}_B(\mathcal{S}_i) \cup \mathcal{L}(p)$.

Step (8) If there are physical links e that connect two nodes in $\mathcal{V}_B(\mathcal{S}_i)$ such that $e \notin \mathcal{E}_B(\mathcal{S}_i)$, $\mathcal{E}_B(\mathcal{S}_i) \leftarrow \mathcal{E}_B(\mathcal{S}_i) \cup \{e\}$.

Step (9) It goes to Step (4).

Step (10) The MRC algorithm is applied to the biconnected graph of each slice $\mathcal{S}_i \in \mathcal{S}$ to construct K backup routing configurations for the slice.

In step (3), the resulting graph $\mathcal{B}(\mathcal{S}_i)$ is a connected graph because it consists of physical nodes and links corresponding to the slice. In step (4), to consider only the condition B of the DFS tree discussed in Section 4.2.2, we deal with the case where the root of the DFS tree is not an articulation point. By performing step (7), we can eliminate the articulation point because the resulting graph $\mathcal{B}(\mathcal{S}_i)$ no longer meets the condition B. In Step (10), we use the MRC algorithm introduced in [17] to construct backup routing configurations.

We here explain an example of biconnected graph construction with Figure 8. In this figure, we assume that there is a slice consisting of three virtual nodes and two virtual links. The virtual nodes are mapped to physical nodes 1, 5, and 8. The path between nodes 1 and 5 includes nodes 1, 2, 5, links 1-2, and 2-5. Thus, the resulting graph $\mathcal{B}(\mathcal{S}_i)$ in step (3) has the structure shown in Figure 8(a). In step (4), the DFS tree is constructed. In this case, articulation points are nodes 2 and 5 as shown in Figure 8(b). In step (6), the path length on the substrate network without using the DFS tree is calculated for each node pair. Specifically, for node 2, the path length from node 5, which is a child node of node 2, to node 1, which is an ancestor node of node 2, is calculated. Also, for node 5, the path length from node 8 to nodes 1 and 2 is calculated. In this case, the path length from node 8 to node 2 is the smallest (i.e., $l(5,1) = 3$, $l(8,1) = 3$ and $l(8,2) = 2$). Thus, the path (i.e., 8-7-2) is added to the graph $\mathcal{B}(\mathcal{S}_i)$ in step (7). Then, in step (8), link 5-7 is added to the graph $\mathcal{B}(\mathcal{S}_i)$ (Figure 8(c)). The DFS algorithm is applied to the resulting graph $\mathcal{B}(\mathcal{S}_i)$ to detect articulation points (Figure 8(d)). By repeating this procedure, we can construct a biconnected graph for the slice, using a small number of components (Figure 8(e)). The backup routing configurations that are constructed by applying the MRC algorithm to the resulting graph are shown in Figure 8(f) where $K = 3$. As we can see from this figure, we can construct the backup routing configurations using only necessary components while without using the whole substrate network.

Note that this algorithm can always construct a bi-connected graph $\mathcal{B}(\mathcal{S}_i)$ for each slice $\mathcal{S}_i \in \mathcal{S}$. The algorithm detects an articulation point in Step (4). Then, the detected articulation point is eliminated by Steps (6)-(8) by adding

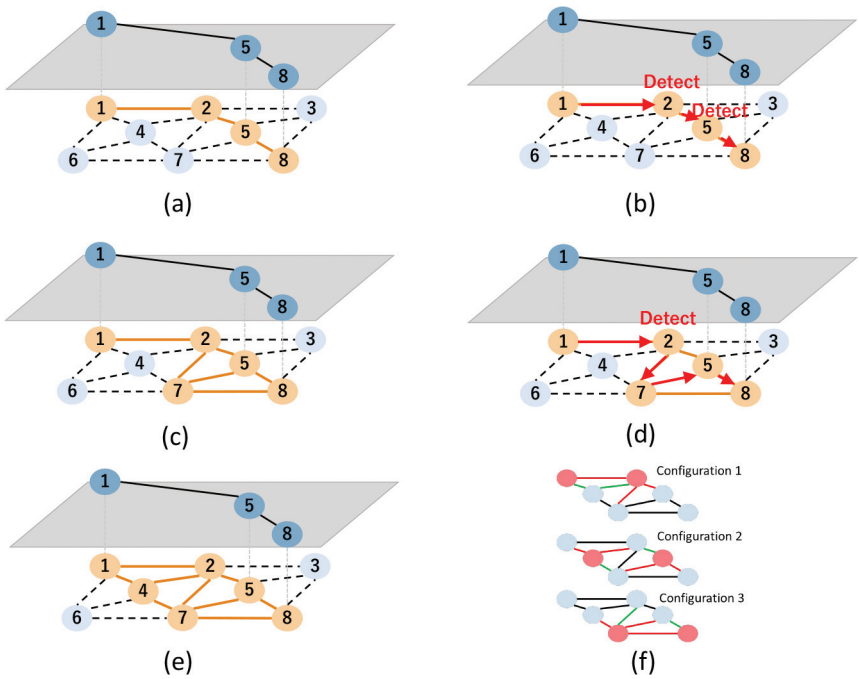


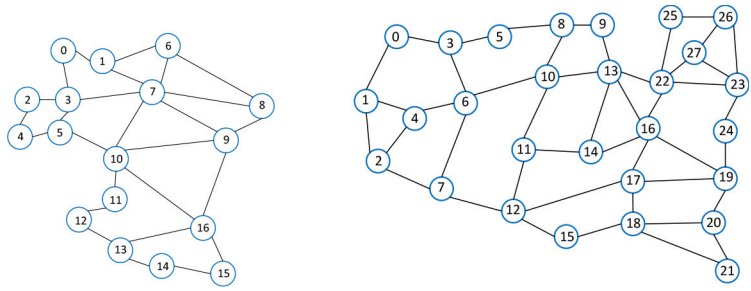
Figure 8: Procedure for creating a bi-connected graph.

some links and nodes of the substrate network. By repeating these steps, the algorithm constructs a bi-connected graph. In the worst case, the resulting bi-connected graph corresponds to the whole substrate network, which is a bi-connected graph.

5 Numerical Experiments

5.1 Model

In order to examine the performance of the proposed method, we conduct numerical experiments. As substrate networks, we use two network models: German Network (GN) and US Backbone Network (UBN) shown in Figure 9(a) and (b), respectively. We randomly make 5 slices on the substrate network for each experiment. The number of virtual nodes on each slice is set to the same value. The virtual nodes of each slice are mapped to randomly selected physical nodes of the substrate network. The structure of each slice is a complete graph. The amount of traffic between each virtual node pair is randomly selected



(a) German network. (b) US backbone network.

Figure 9: Network model.

from among [12, 20]. They communicate with each other using the virtual link, which is the shortest path on the substrate network.

As performance metrics, we use the number of hops of each path and the load of each link after a failure occurrence. The link load is defined as the sum of the amount of traffic passing through the link. For each slice, we construct $K = 4$ backup routing configurations. In each experiment, we measure the number of hops and the link load for all possible node failures. We collect 100 independent samples from experiments and the average is shown in each result.

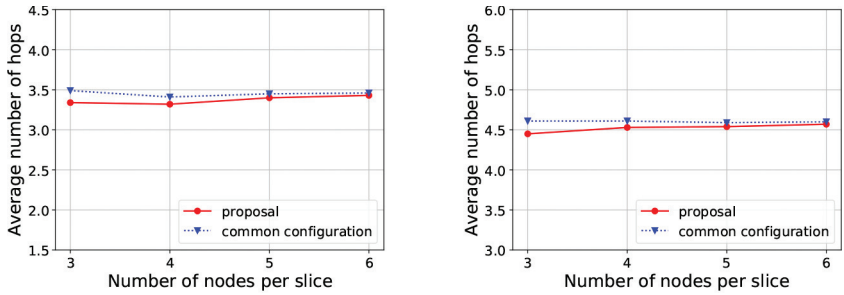
5.2 Results

First, we examine the average number h_{ave} of hops of backup routing paths for slices. It is defined as

$$h_{\text{ave}} = \frac{\sum_{\mathcal{S}_i \in \mathcal{S}} \sum_{k \in \mathcal{K}_i} \sum_{p \in \mathcal{P}_i^{[k]}} h_p^{[k]}}{\sum_{\mathcal{S}_i \in \mathcal{S}} \sum_{k \in \mathcal{K}_i} |\mathcal{P}_i^{[k]}|},$$

where $\mathcal{K}_i$ represents a set of backup routing configurations for slice $\mathcal{S}_i \in \mathcal{S}$. The symbol $\mathcal{P}_i^{[k]}$ represents a set of backup routing paths on backup routing configuration $k \in \mathcal{K}_i$. The symbol $h_p^{[k]}$ represents the length of backup path $p \in \mathcal{P}_i^{[k]}$. For the sake of comparison, we consider the case (labeled with "common configuration") where each slice uses common backup routing configurations, which are constructed by applying the MRC algorithm to the whole substrate network.

Figure 10(a) shows the average number h_{ave} of hops of backup routing paths as a function of the number of virtual nodes in each slice in the GN model. From this figure, we observe that the average number h_{ave} of hops of backup routing paths in the proposed method is slightly smaller than that in



(a) GN model. (b) UBN model.

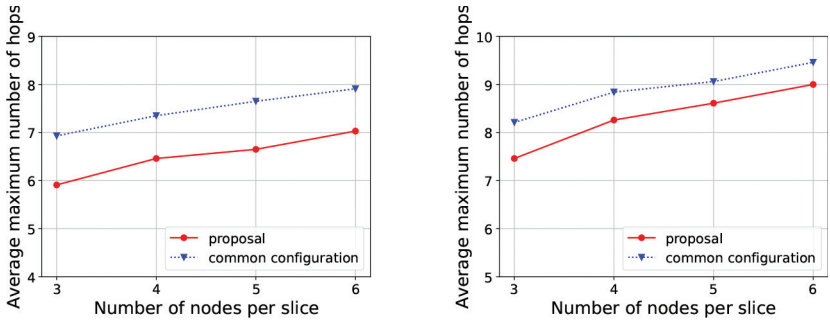
Figure 10: Average number of hops of backup routing paths.

the case where each slice uses common backup routing configurations. Also, Figure 10(b) shows the average number h_{ave} of hops of backup routing paths as a function of the number of virtual nodes in each slice in the UBN model. As shown in this figure, the proposed method slightly improves the average number h_{ave} of hops of backup routing paths, compared with the case of using the common configurations. This result is very similar to that in the GN model. These results mean that the length of most backup routing paths is not much reduced by the proposed method. However, the length of some backup routing paths is remarkably reduced by the proposed method as discussed next.

We then examine the maximum number of hops of backup routing paths for each slice. We define the average maximum number $h_{\mathrm{ave\_max}}$ of hops as

$$h_{\mathrm{ave\_max}} = \frac{\sum_{\mathcal{S}_i \in \mathcal{S}} \left\{ \max_{k \in \mathcal{K}_i, p \in \mathcal{P}_i^{[k]}} h_p^{[k]} \right\}}{|\mathcal{S}|}.$$

Figure 11(a) shows the average maximum number $h_{\mathrm{ave\_max}}$ of hops as a function of the number of virtual nodes in each slice in the GN model. From this figure, we observe that the proposed method efficiently reduces the average maximum number of hops. The reason is that making long detoured paths tends to be suppressed by constructing dedicated backup routing configurations for respective slices on small substrate networks as shown in Figure 5. On the other hand, as discussed above, the improvement of the average number h_{ave} of hops of the proposed method is relatively small. These results imply that the proposed method tends to smooth the variance of the length of backup routing paths. Figure 11(b) shows the average maximum number $h_{\mathrm{ave\_max}}$ of hops as a function of the number of virtual nodes in each slice in the UBN model. As shown in this figure, the proposed method also works well in the UBN model.



(a) GN model. (b) UBN model.

Figure 11: Average maximum number of hops.

Next, we examine the maximum link load after a failure occurs. The maximum link load is generally used for a performance metric of load-balancing. The high maximum link load indicates that traffic concentrates on a specific network link. On the other hand, the low maximum link load indicates that traffic is distributed to various links. In this paper, we define the maximum link load $l_{\max}^{[v]}$ after node v is failed as

$$l_{\max}^{[v]} = \max_{e \in \mathcal{E}} l_e^{[v]},$$

where $l_e^{[v]}$ represents the load of link $e \in \mathcal{E}$ after node v is failed. $l_e^{[v]}$ is given by

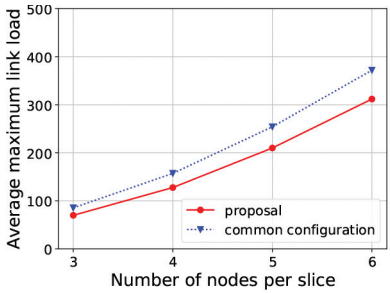
$$l_e^{[v]} = \sum_{\mathcal{S}_i \in \mathcal{S}} \sum_{p \in \mathcal{P}_i^{[b(i,v)]}: e \in p} \lambda_p,$$

where λ_p represents the traffic volume of path p and $b(i, v)$ represents the routing configuration for slice $\mathcal{S}_i \in \mathcal{S}$ when node v is failed. In this paper, we assume that each virtual node pair uses the shortest path on the selected routing configuration to communicate with each other after the failure occurs. Note that if the backup routing configurations for the slice do not include the failed node v, the normal routing configuration is used.

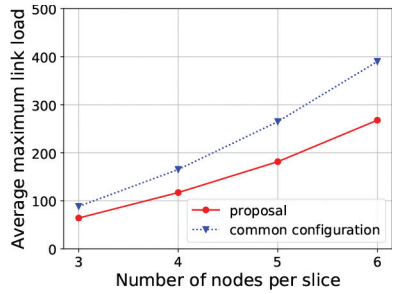
Figure 12(a) and (b) show the average maximum link load for each failure case as a function of the number of virtual nodes in each slice in the GN and UBN models, respectively. The average maximum link load $l_{\text{ave\_max}}$ for each failure case is defined as

$$l_{\text{ave\_max}} = \frac{\sum_{v \in \mathcal{V}} l_{\max}^{[v]}}{|\mathcal{V}|}.$$

As we can see from these figures, the proposed method efficiently reduces the maximum link load, regardless of the number of virtual nodes in each slice.



(a) GN model.

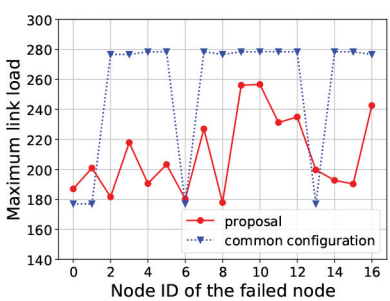


(b) UBN model.

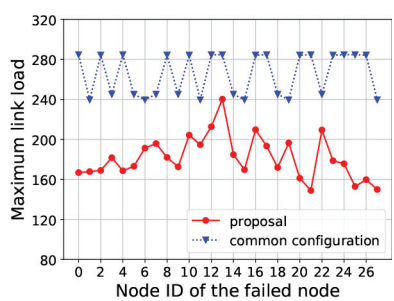
Figure 12: Maximum link load.

This is because when the slices use a common backup routing configuration, the backup routing paths tend to share many links, as discussed in the example of Figure 4. On the other hand, in the proposed method, the slices tend to use different links for backup routing paths on their respective backup routing configurations, as discussed in the example of Figure 5. As a result, the proposed method avoids the concentration of multiple backup routing paths on a specific link after a failure occurs. This result means that the proposed method efficiently distributes the load on network links.

Figure 13(a) and (b) show the maximum link load $l_{\max}^{[v]}$ when node v is failed, in the GN and UBN models, respectively, where the number of virtual nodes in each slice is 5. As we can see from these figures, the proposed method efficiently reduces the maximum link load in most failure cases. In the UBN model, especially, the proposed method works well because the size of the substrate network is large. We thus conclude that the proposed method can distribute the load on network links after a failure occurs.



(a) GN model.



(b) UBN model.

Figure 13: Maximum link load for each node failure.

6 Implementation Demonstration with P4

We here demonstrate simple implementation of our proposed method. In this paper, we confirm that the implementation of the failure recovery mechanism of MRC in our proposed method works for network slicing environments through demonstration experiments using P4. Note that in this paper, we do not discuss the detailed performance evaluation similar to the numerical experiments. We leave the performance evaluation using the implementation with P4 as future work. P4 is a language to program the behavior of data planes (i.e., functions used for packet forwarding in network switches) [6]. P4 enables us to implement various kinds of network applications such as traffic offloading, duplicate address detection, bandwidth management, denial of service mitigation, traffic metering, routing control, and service protection [8, 14, 15, 24, 29, 34].

We can define and add the functions required for packet forwarding in network slicing environments by using P4. In this paper, we mainly discuss how to implement packet forwarding on each slice, while the MRC implementation is based on [16]. Note that we do not consider failure detection. We focus on the behavior of each P4-enabled switch after a failure occurs.

6.1 The Concept of P4

P4 adopts the concept of match-action pipelines. Packet forwarding in P4-enabled switches is performed by table lookups and corresponding actions. Figure 14 shows the procedure of packet forwarding in a P4-enabled switch. An incoming packet is first handled by the parser. The parser handles only the packet header, while buffering data in the packet. It extracts some fields from the packet header based on the programmed parse graph.

The extracted header fields are then passed to the ingress match+action tables consisting of lookup keys (e.g., IP addresses and MAC addresses), corre-

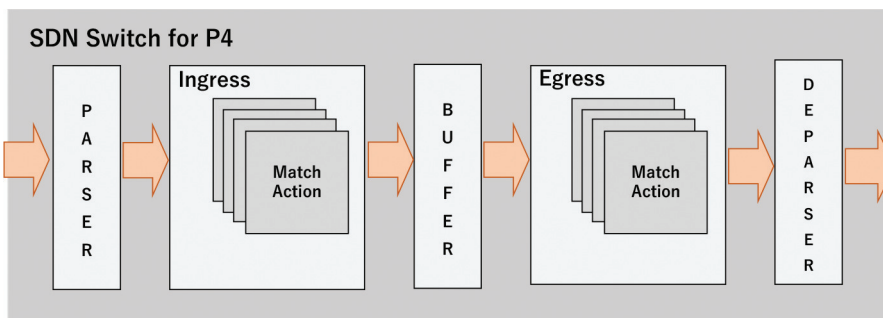


Figure 14: Packet forwarding process in P4 devices.

sponding actions (e.g., forward and drop), and parameters. The match+action tables process the packet header according to the lookup keys and the actions, which can be modified programmatically. Moreover, it determines an output port and a queue into which the packet is placed. Then the packet header is passed to the egress match+action tables, which can be also defined. After processing the packet header there, the switch forwards the packet to the selected output port.

6.2 Implementation of Network Slicing and MRC

In P4, incoming packets can be forwarded using user-defined headers. In order to identify slices, we define a header named MySlice between the Ethernet header and the IP header, as shown in Figure 15. The proto_id field is used to identify whether the IP header follows the MySlice header. The slice field is used to select a slice on which the packet is transmitted.

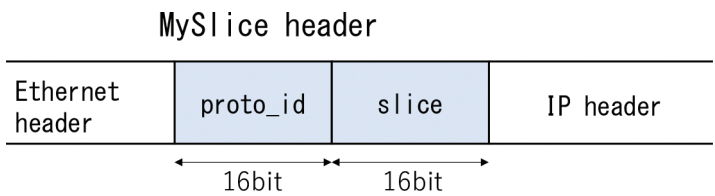


Figure 15: Structure of the MySlice header.

At a P4-enabled switch, an incoming packet is processed by the parser with the following procedure. The parser first extracts the Ethernet header, and then identifies whether the MySlice header is valid by checking the type field of the Ethernet header. If the MySlice header is not valid, the parser extracts the IP header to forward the packet with the use of a default routing table. If the MySlice header is valid, the parser extracts it, and then identifies the slice to be used for the packet transmission by checking the slice field of the MySlice header. In our implementation, an individual routing table is created for each slice in the P4-enabled switch. By processing the incoming packet based on the MySlice header and the routing table, it is possible to forward the packet to an appropriate output port corresponding to its slice.

Furthermore, in our implementation, routing tables for backup routing configurations in addition to the normal routing configuration are created for each slice as shown in Figure 16. After switching to a backup routing configuration in the case of a failure, each P4-enabled switch needs to identify the configuration to be used for forwarding incoming packets. To do so, we use the Type Of Service (TOS) field in the IP header of the incoming packets, which has been introduced in [16]. In the TOS field, the configuration ID that indicates the backup routing configuration to be used is recorded. Each

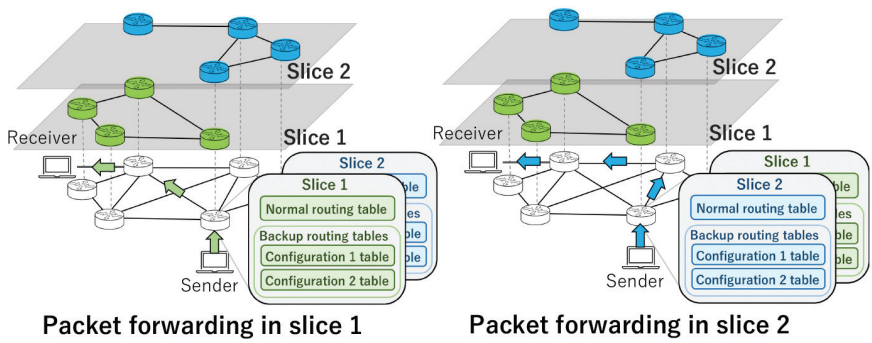


Figure 16: Routing table for each slice and each backup routing configuration.

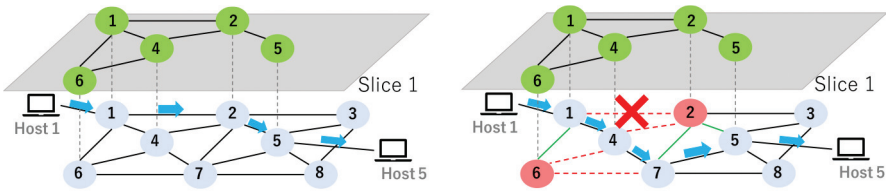
P4-enabled switch forwards the incoming packets according to the routing table for the backup routing configuration corresponding to the configuration ID.

6.3 Demonstration Experiments

In order to confirm our P4-based implementation, we conduct demonstration experiments using Mininet [20]. We construct one substrate network and two slices $\mathcal{S}_1$ and $\mathcal{S}_2$ as shown in Figure 3, where each node represents a P4-enabled switch connected to one host (e.g., host 1 connects to switch 1).

We first confirm the operation of network slicing without failures. We here check the behavior of packet forwarding with the MySlice header defined in our implementation by analyzing packets using Scapy [25]. We assume that packets are transmitted from host 1 to host 5 on slice 1, as shown in Figure 17(a). In this case, host 1 sends packets with the slice field set to 1 in the MySlice header. Figure 18 shows the header information of a transmitted packet at sender host 1 and receiver host 5. From this figure, we observe that the MySlice header is inserted between the Ethernet header and the IP header. Each switch along the routing path recognizes the value of the slice field, and processes the packets according to the routing table of slice 1. As a result, the packets can arrive at receiver host 5.

Next, we assume that packets are transmitted from host 1 to host 5 on slice 2. To do so, the slice field in the MySlice header of the packets is set to 2. Switch 1 connected to host 1 recognizes that the slice field of the incoming packets is 2 and refers to the routing table of slice 2. However, there is no action in the routing table for the packets whose destination address is host 5 on slice 2 because slice 2 does not include switch 1 as shown in Figure 3. Therefore, the packets are dropped at switch 1. We observed this behavior, but we omit this result to save space on the paper. From these results, we



(a) Normal routing configuration. (b) Backup routing configuration.

Figure 17: Routing path.

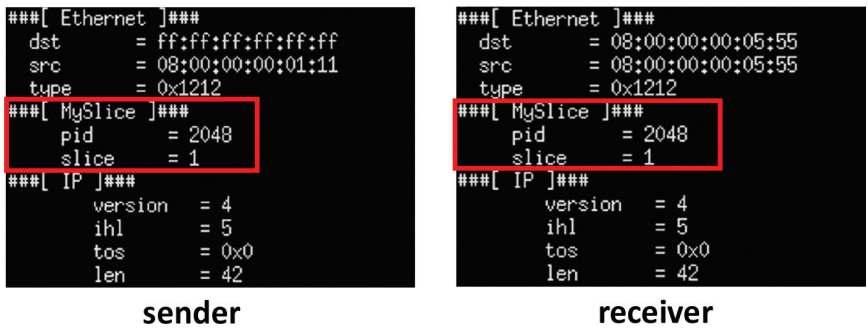


Figure 18: Header information of packets on slice 1.

Figure 19: Packets after switching the routing configuration.

conclude that the switches adequately process incoming packets based on the routing tables configured for the slices.

Finally, we confirm the operation of the MRC with the backup routing configurations for the slices. We assume that a failure of link (1, 2) occurs, which is done by the "link down" command of Mininet. Under this assumption, we send packets from host 1 to host 5 on slice 1. On the normal routing configuration, the routing path is 1-2-5 as shown in Figure 17(a). However, the

path includes the failed link, so that the packets cannot arrive at host 5 on the normal routing configuration. On the other hand, by using the appropriate backup routing configuration illustrated in Figure 17(b), the packets whose TOS is set to the corresponding value (1 in this example) can arrive at host 5 as shown in Figure 19. From these results, we can see that the proposed method ensures routing paths in the case of a failure.

7 Conclusion

In this paper, we proposed a load-balanced fast failure recovery method based on MRC in communications using network slicing. The proposed method constructs backup routing paths for each slice by applying MRC only with a small number of physical nodes and links. By doing so, the proposed method can avoid making inefficient detour paths for each slice. Through numerical experiments, we showed the effectiveness of the proposed method. In addition, we implemented our proposed method with P4. Through demonstration experiments, we confirmed our P4-based implementation of the proposed method. In this paper, we focused on the behavior of our proposed method after a failure occurrence. As future work, we will consider further implementation including failure detection and route selection at each router to examine the performance such as failure recovery time in our implementation.

References

[1] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions," *IEEE Communications Surveys & Tutorials*, 20(3), 2018, 2429–53.

[2] M. Ahuja and Y. Zhu, "An Efficient Distributed Algorithm for Finding Articulation Points, Bridges, and Biconnected Components in Asynchronous Networks," in *The Ninth Conference on Foundations of Software Technology and Theoretical Computer Science*, 1989, 99–108.

[3] J. Ali, G.-M. Lee, B.-H. Roh, D. K. Ryu, and G. Park, "Software-Defined Networking Approaches for Link Failure Recovery: A Survey," *Sustainability*, 12(10), 2020.

[4] A. Baumgartner, T. Bauschert, A. M. C. A. Koster, and V. S. Reddy, "Optimisation Models for Robust and Survivable Network Slice Design: A Comparative Analysis," in *2017 IEEE Global Communications Conference (GLOBECOM)*, 2017, 1–7.

[5] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming Platform-Independent Stateful Openflow Applications Inside the Switch," *ACM SIGCOMM Computer Communication Review*, 44(2), 2014, 44–51.

[6] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, 44(3), 2014, 87–95.

[7] C. Cascone, D. Sanvito, L. Pollini, A. Capone, and B. Sanso, "Fast Failure Detection and Recovery in SDN with Stateful Data Plane," *International Journal of Network Management*, 27(2), 2017, e1957.

[8] Y.-W. Chen, L.-H. Yen, W.-C. Wang, C.-A. Chuang, Y.-S. Liu, and C.-C. Tseng, "P4-Enabled Bandwidth Management," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2019, 1–5.

[9] M. Chiesa, R. Sedar, G. Antichi, M. Borokhovich, A. Kamisiński, G. Nikolaidis, and S. Schmid, "PURR: A Primitive for Reconfigurable Fast Reroute: Hope for the Best and Program for the Worst," in *the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, 1–14.

[10] C.-Y. Chu, K. Xi, M. Luo, and H. J. Chao, "Congestion-Aware Single Link Failure Recovery in Hybrid SDN Networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, 1086–94.

[11] G. Cong and D. Bader, "An Experimental Study of Parallel Biconnected Components Algorithms on Symmetric Multiprocessors (SMPs)," in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[12] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," *IEEE Communications Magazine*, 55(5), 2017, 94–100.

[13] M. Goyal, M. Soperi, E. Baccelli, G. Choudhury, A. Shaikh, H. Hosseini, and K. Trivedi, "Improving Convergence Speed and Scalability in OSPF: A Survey," *IEEE Communications Surveys & Tutorials*, 14(2), 2012, 443–63.

[14] X. Guo, N. Liu, X. Hou, S. Gao, and H. Zhou, "An Efficient NDN Routing Mechanism Design in P4 Environment," in *2021 2nd Information Communication Technologies Conference*, 2021, 28–33.

[15] L. He, P. Kuang, Y. Liu, G. Ren, and J. Yang, "Towards Securing Duplicate Address Detection Using P4," *Computer Networks*, 198, 2021, 108323.

[16] K. Hirata and T. Tachibana, "Implementation of Multiple Routing Configurations on Software-Defined Networks with P4," in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2019, 13–6.

[17] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Multiple Routing Configurations for Fast IP Network Recovery," *IEEE/ACM Transactions on Networking*, 17(2), 2009, 473–86.

[18] L. Leydesdorff, "Clusters and Maps of Science Journals Based on Bi-connected Graphs in Journal Citation Reports," *Journal of Documentation*, 60(4), 2004, 371–427.

[19] Y.-D. Lin, H.-Y. Teng, C.-R. Hsu, C.-C. Liao, and Y.-C. Lai, "Fast Failover and Switchover for Link Failures and Congestion in Software Defined Networks," in *2016 IEEE International Conference on Communications (ICC)*, 2016, 1–6.

[20] "Mininet," http://mininet.org.

[21] T. Misugi, K. Hirata, and T. Tachibana, "Implementation of a Fast Failure Recovery Method Considering Load Distribution for Network Slicing," in *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2021, 1895–8.

[22] H. Miura, K. Hirata, and T. Tachibana, "P4-Based Design of Fast Failure Recovery for Software-Defined Networks," *Computer Networks*, 216, 2022, 109274.

[23] P. M. Mohan and M. Gurusamy, "Resilient VNF Placement for Service Chain Embedding in Diversified 5G Network Slices," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, 1–6.

[24] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 Edge Node Enabling Stateful Traffic Engineering and Cyber Security," *Journal of Optical Communications and Networking*, 11(1), 2019, A84–A95.

[25] "Scapy," https://scapy.net.

[26] R. Sedar, M. Borokhovich, M. Chiesa, G. Antichi, and S. Schmid, "Supporting Emerging Applications with Low-Latency Failover in P4," in *2018 Workshop on Networking for Emerging Applications and Technologies*, 2018, 52–7.

[27] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting Carrier-Grade Recovery Requirements," *Computer Communications*, 36(6), 2013, 656–65.

[28] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The Algorithmic Aspects of Network Slicing," *IEEE Communications Magazine*, 55(8), 2017, 112–9.

[29] S.-Y. Wang, H.-W. Hu, and Y.-B. Lin, "Design and Implementation of TCP-Friendly Meters in P4 Switches," *IEEE/ACM Transactions on Networking*, 28(4), 2020, 1885–98.

[30] Y. Wang, S. Feng, H. Guo, X. Qiu, and H. An, "A Single-Link Failure Recovery Approach based on Resource Sharing and Performance Prediction in SDN," *IEEE Access*, 7, 2019, 174750–63.

[31] R. Wen, G. Feng, J. Tang, T. Q. S. Quek, G. Wang, W. Tan, and S. Qin, "On Robustness of Network Slicing for Next-Generation Mobile Networks," *IEEE Transactions on Communications*, 67(1), 2019, 430–44.

[32] S. Wijethilaka and M. Liyanage, "Survey on Network Slicing for Internet of Things Realization in 5G Networks," *IEEE Communications Surveys & Tutorials*, 23(2), 2021, 957–94.

[33] J. Xu, S. Xie, and J. Zhao, "P4Neighbor: Efficient Link Failure Recovery with Programmable Switches," *IEEE Transactions on Network and Service Management*, 18(1), 2021, 388–401.

[34] G. Zhang, S. Gao, J. Yue, and Z. Zhao, "A Service Protection Mechanism Impelemented on P4 by Packet Replication," in *2021 2nd Information Communication Technologies Conference*, 2021, 1–5.

[35] Q. Zhang, O. Ayoub, J. Wu, F. Musumeci, G. Li, and M. Tornatore, "Progressive Slice Recovery with Guaranteed Slice Connectivity After Massive Failures," *IEEE/ACM Transactions on Networking*, 30(2), 2022, 826–39.