Implementation of multiple routing configurations on software-defined networks with P4

Kouji Hirata* and Takuji Tachibana†
* Faculty of Engineering Science, Kansai University, Osaka, Japan
E-mail: hirata@kansai-u.ac.jp
† Graduate School of Engineering, University of Fukui, Fukui, Japan
E-mail: takuji-t@u-fukui.ac.jp

***Abstract*—In order to maintain high availability of communication networks, we should recover failures immediately after the failures occur. As one of techniques achieving such fast failure recovery, multiple routing configurations (MRC) have been proposed. In MRC, multiple backup routing configurations are prepared in advance to fast repair a single link/node failure. When a failure occurs during data transmission using a normal routing configuration, MRC changes the routing configuration to another routing configuration that does not use the failed point. Thus, MRC can realize fast recovery within few tens of milliseconds and continue the data transmission. In this paper, we implement MRC on software-defined networks with P4 (Programming Protocol-independent Packet Processeros). P4 is a programming language that enables us to define the behavior of the data plane of network equipment. We examine the MRC implementation, using Mininet.**

I. Introduction

Recently, software-defined networking (SDN) has been widely used due to its flexible characteristic. SDN makes it possible to control networks more flexibly and efficiently [5]. The feature of SDN is the separation of the control plane, which is responsible for a routing control function, from the data plane, which is responsible for a packet forwarding function. Communications between these planes are carried out by a communication protocol such as OpenFlow [2]. As the control plane, a centralized controller named SDN controller is used. It translates network management policies into packet forwarding rules, which are called flow entries, and installs them on flow tables of network devices (i.e., data plane) such as SDN switches. The network devices forward packets based on their flow tables. Each flow entry is composed of match field, action, and counter. The match field is used to identify packet flows based on elements in packet headers such as IP address, MAC address, and port number. For instance, in OpenFlow 1.4, the match field can use 41 elements to identify packet flows. We can change the granularity of flows by adjusting the match field. The actions define how switches process flows, e.g., forward and drop. The counter records statistical information such as the number of arrival packets.

In conventional SDN environments, we can modify the behavior of the control plane but not the data plane. Specifically, routing policies can be flexibly defined, but the corresponding actions cannot be flexibly done. In order to resolve this problem, a programming language for the data plane named P4 (Programming Protocol-independent Packet Processeros) has been proposed [3]. P4 can express how incoming packets are processed by the data plane (i.e., SDN switches), while it does not specify the behavior of the control plane, which is managed by SDN controllers. Because P4 is an open-source and permissively licensed language, it is currently widely used. In this paper, we implement multiple routing configurations (MRC) [4], which provide fast recovery from a single node/link failure, on SDN environments with P4.

In order to maintain high availability of communication networks, we should recover failures immediately after the failures occur. MRC is one of techniques achieving such fast failure recovery. In MRC, multiple backup routing configurations are prepared in advance. When a single node/link failure occurs during data transmission using a normal routing configuration, MRC immediately changes the routing configuration to another routing configuration that does not use the failed point, and thus continue the data transmission. MRC can realize fast recovery within few tens of milliseconds. Furthermore, the routing control of MRC is stable at the time of failure recovery because each node has common routing configurations. In this paper, we examine the MRC implementation using Mininet [1], which is an emulator creating SDN environments.

The rest of this paper is organized as follows. Section II briefly explains MRC. In Section III, we discuss the implementation of MRC with P4. Furthermore, we examine the behavior of the MRC implementation, using Mininet. We state the conclusion of this paper in Section IV.

II. Multiple routing configurations

MRC prepares K ($K > 1$) backup routing configurations in advance to realize fast failure recovery. Even when a link/node failure occurs on a normal routing configuration, MRC can continue data transmission by using a backup routing configuration that does not use the failure point. In K backup routing configurations, each node is classified into a normal node and an isolated node. Also, each link is classified into a normal link, an isolated link, and a restricted link. The normal link/node represents a link/node that can be directly used for data transmission when a failure occurs. The isolated link/node represents a link/node that cannot be used for data transmission when a failure occurs. The restricted link represents a link

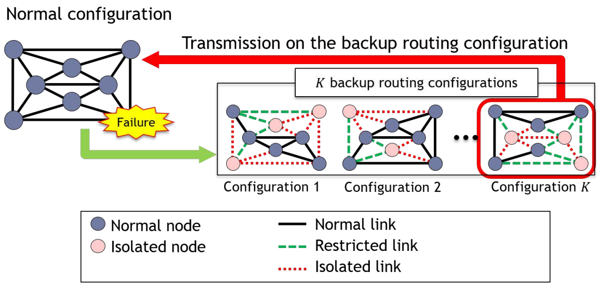

Fig. 1. Routing configurations.

that is used only for the first hop or the last hop of data transmission, which takes into account the possibility that an isolated node connecting to the restricted link is not failed.

Fig. 1 shows an example that a normal configuration and K backup configurations generated form the normal configuration. In MRC, a normal configuration and K backup routing configurations should meed the following conditions.

1) Respective nodes and links become isolated nodes and isolated links in at least one backup routing configurations.
2) In each backup routing configuration, there exists a connected graph consisting of all the normal nodes and the normal links.
3) A restricted link or an isolated link are connected to an isolated node while a normal link is not connected to an isolated node.
4) At least one of links connecting to an isolated node are restricted links.
5) In the case where two nodes are connected by a restricted link, one is a normal node and the other is an isolated node.
6) In the case where two nodes are connected by an isolated link, one is an isolated node.

By using one of K backup routing configurations, MRC maintains high availability. Specifically, when a single node failure occurs, MRC selects a backup routing configuration in which the node is an isolated node. Similarly, when a single link failure occurs, MRC selects a backup routing configuration in which the link is an isolated link. Note that in the selected backup routing configuration, there are cases where non-failed links/nodes are treated as isolated links/nodes. For example, in Fig. 1, when a single node failure occurs in the normal configuration and backup routing configuration 1 is used, there are no failures in two isolated nodes and seven isolated links. These isolated nodes without failures can transmit and receive data, using restricted links. Therefore, MRC can continue data transmission without identification of a node failure or a link failure.

Fig. 2 shows the procedure of MRC when a single link/node failure occurs. When a failure occurs on a routing path during data transmission, the previous node of failed point detects the failure. The node selects an appropriate backup routing configuration in which the failed point is selected as an isolated link/node. The node then continues data transmission, using the selected backup routing configuration. Furthermore, the node puts the information on the selected backup routing configuration into packets and notifies other nodes of the information. As a result, MRC realizes fast failure recovery only with simple route modification.

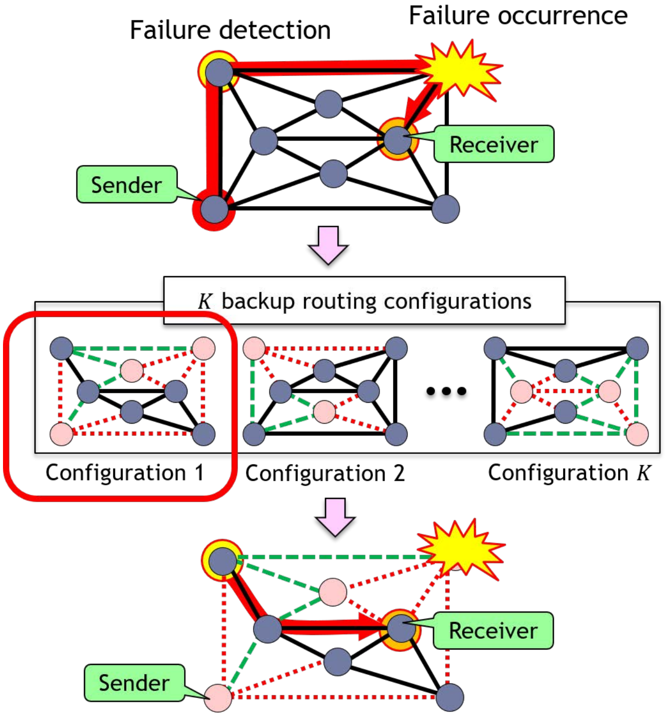

Fig. 2. Failure recovery procedure.

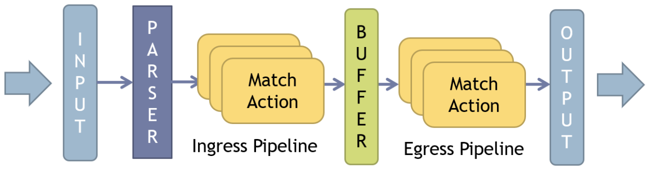

Fig. 3. Procedure of packet forwarding in SDN switches.

III. Implementation of Multiple Routing Configurations

A. *The concept of P4*

P4 uses the concept of match-action pipelines. Packet forwarding in SDN switches is done by table lookups and corresponding actions. Fig. 3 shows the procedure of packet forwarding in SDN switches. An incoming packet is first handled by the parser. The parser only handles the packet header and data in the packet is assume to be buffered. The parser recognizes and extracts fields from header based on the programmed parse graph.

The extracted header field is then passed to the ingress match+action tables consisting of lookup keys (e.g., IP addresses and MAC addresses), corresponding actions (i.e., forward and drop), and parameters. The match+action tables

```
header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    ...
    ...
}
```

Fig. 4. IP packet header.

```
state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept;
}
```

Fig. 5. Header extraction in the parser.

process the packet header according to the lookup keys and the actions, which we can modify by programming. Furthermore, it determines an egress port and a queue into which the packet is placed. Then the packet is passed to the egress match+action tables. We can also define the egress match+action tables. Finally, the packet is forwarded to the output port.

B. Assumption

In this paper, we implement MRC in OpenFlow and P4 environments, which aims at performing fast recovery that selects a backup configuration when a single link/node failure occurs. This paper focuses on making backup routing configurations and confirms that we can use MRC with P4. Note that we do not consider failure detection. We left the failure detection as future work.

We examine the behavior of MRC under the following assumption. We prepare a normal routing configuration and some backup routing configurations. The use of the backup routing configurations is determined in an offline manner. Specifically, each node has the backup routing configurations and set common identification number to them among nodes (i.e., 0 for the normal routing configuration and 1 to K for the backup routing configurations). Each sender puts an identification number into packets in order to use the corresponding routing configuration. Each node selects the routing configuration based on the information in the packets.

In this paper, we realize MRC by selecting routing configurations based on the value of the Type Of Service (TOS) field in the packet header, assuming that each sender detects a failure occurrence immediately after the failure occurs and changes the value of the TOS field according to the failure point. In the following, we particularly explain the implementation of MRC in SDN switches.

C. Implementation

We first define the IP packet header as shown in Fig. 4. The TOS field is “bit <8> diffserv” in the header definition. We change this value according to failures in order to use backup routing configurations. When a packet arrives at an SDN switch, the parser extracts the IP header of the packet, which is described in the P4 program as shown in Fig. 5. The extracted header is passed to the ingress match+action tables according to the value of the TOS filed in the header.

```
table ipv4_lpm {
    key = {
         hdr.ipv4.dstAddr: lpm;
    }
    actions = {
         ipv4_forward;
         drop;
         NoAction;
    }
    size = 1024;
    default_action = drop();
}
```

Fig. 6. Match+action table.

```
action drop() {
    mark_to_drop();
}

action ipv4_forward(macAddr_t dstAddr,
                  egressSpec_t port) {
    standard_metadata.egress_spec =
                  port;
    hdr.ethernet.srcAddr =
                  hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

Fig. 7. Action definition.

Fig. 6 represents the definition of an ingress match+action table described in the P4 program. The table consists of key and corresponding actions. In this case, the key is the destination IP address and the actions are forward and drop. The actions are defined as shown in Fig. 7. The drop action drops incoming packets that do not match the flow table. On the other hand, the forward action rewrites the packet headers such as MAC addresses and the value of TTL and then passes the packets to the egress+match action tables. In our implementation, we prepare some match+action tables and select one according to the value of the TOS field. This implementation is described as shown in Fig. 8, where there are three match+action tables: ipv4_lpm, ipv4_lpm2, and ipv4_lpm3. In this example, if the value of TOS field is equal to 0, the table ipv4_lpm is selected. Also, if the value of TOS field is equal to 1, the table ipv4_lpm2 is selected; otherwise, the table ipv4_lpm3 is selected. Note that the tables ipv4_lpm2 and ipv4_lpm3 are defined as well as the table ipv4_lpm shown in Fig. 6. In the implementation, we do not prepare the egress match+action tables.

In the ingress and the egress processing, the packet is processed based on the flow table, part of which is represented in Fig. 9. The flow table determines which action is applied to the packet and which egress port the packet is sent to. For

```
apply {
    if (hdr.ipv4.isValid()) {
        if(hdr.ipv4.diffserv == 0){
            ipv4_lpm.apply();
        }
        else if(hdr.ipv4.diffserv == 4){
            ipv4_lpm2.apply();
        }
        else{
            ipv4_lpm3.apply();
        }
    }
}
```

Fig. 8. Table selection based on the TOS field.

```
{
    "table": "MyIngress.ipv4_lpm",
    "match": {
        "hdr.ipv4.dstAddr":
          ["10.0.1.1", 32]
    },
    "action_name":
          "MyIngress.ipv4_forward",
    "action_params": {
        "dstAddr": "00:00:00:00:01:01",
        "port": 1
    }
}
```

Fig. 9. Flow table.

each SDN switch, we install a flow table that consists of entries for each match+action table (i.e., ipv4_lpm, ipv4_lpm2, and ipv4_lpm3).

D. Evaluation

In order to confirm the implementation of MRC, we conduct a practical experiment using Mininet [1]. Fig. 10 shows the routing configurations used in this experiment. The bandwidth of each link is equal to 1 Mbps. We prepare an normal routing configuration (A) and two backup routing configurations (B) and (C). In each routing configuration, three UDP flows (node 1 to node 6, node 2 to node 4, and node 3 to node 5) are transmitted for 180 seconds by Iperf. Each sender sets the value of TOS field according to the routing configurations. Specifically, when we use the routing configurations (A), (B), and (C), the value of TOS filed is set to be 0, 4, and 8, respectively, at each sender. In this scenario, we use the backup routing configuration (B) (resp. (C)) against a link failure between nodes 2 and 5 (resp. nodes 2 and 3). Therefore, nodes 2 and 3 can transmit packets, using restricted links.

Table I shows the throughput of each flow obtained from the experiment. As we can see from this table, in the routing configuration (A), flows 2 and 3 share link 2-5. Thus, the throughput of flows 2 and 3 is smaller than the throughput of flow 1. Because all the flows share link 3-6 in the backup routing configuration (B), they have similar throughput. In the backup routing configuration (C), the throughput of flow 3 is the largest because flows 1 and 2 share link 2-5. These observations indicate MRC works well in the experiment. Furthermore, in the experiment, we confirm that these flows correctly use the routing configurations based on the value of the TOS field in transmitted packets.

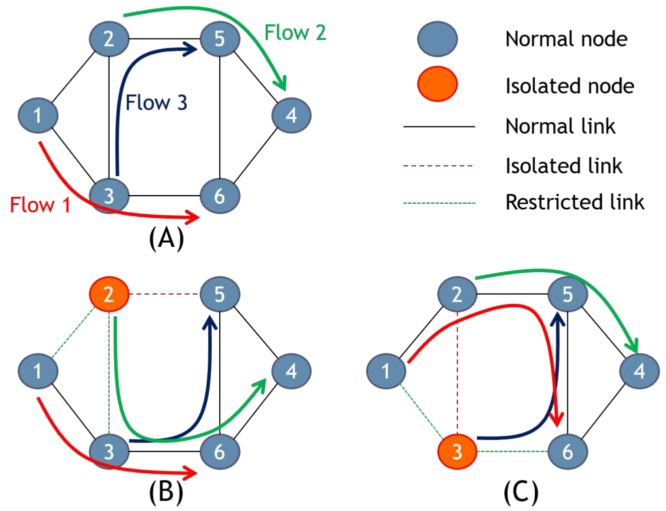

Fig. 10. Model.

TABLE I
THROUGHPUT EVALUATION.

Configuration	Flow 1	Flow 2	Flow 3	Total
(A)	880 kbps	500 kbps	475 kbps	1,855 kbps
(B)	320 kbps	318 kbps	338 kbps	976 kbps
(C)	477 kbps	503 kbps	965 kbps	1,945 kbps

IV. Conclusion

In this paper, we implemented MRC on software-defined networks with P4. We examined the MRC implementation, using Mininet. As future work, we will implement a failure detection mechanism and a mechanism that rewrites the value of TOS field according to the failure detection.

Acknowledgement

This research was partially supported by SCOPE of the Ministry of Internal Affairs and Communications, Japan, under Grant No. 191605004.

References

[1] Mininet: http://mininet.org/
[2] Open Networking Foundation, https://www.opennetworking.org
[3] P. Bosshart et al., “P4: Programming Protocol-Independent Packet Processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 88–95, 2014.
[4] A. Kvalbein, A. Hansen, T. Cicic, S. Gjessing, and O. Lysne, “Multiple routing configurations for fast IP network recovery,” *IEEE/ACM Transactions on Networking*, vol. 17, no. 2, pp. 473–486, 2009.
[5] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: Past, present, future of programmable networks,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.