Fast failure recovery for in-band OpenFlow networks

Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet and Piet Demeester
Department of Information Technology (INTEC), Ghent University - iMinds
E-mail: {firstname.lastname}@intec.ugent.be

Abstract—In OpenFlow, control and data plane are decoupled from switches/routers. Direct programming of routers/switches is realised from one or more servers (so called controllers). In the case of an in-band OpenFlow network, the control traffic (traffic to or from the controllers) is sent on the same channel used to transport data traffic. Therefore, when a failure occurs along the data traffic path, both control and data traffic can be affected. This paper explains how failure recovery can be deployed in such a network. To achieve carrier-grade quality, the network should be able to recover from the failure within 50 ms. We apply two well-known recovery mechanisms -restoration and protection- for the control and the data traffic, and run extensive emulation experiments. The emulation results show that restoration does not allow to recover within 50 ms. Moreover, the restoration of the control traffic delays the restoration of the data traffic. The emulation results also show that protection for both control and data traffic can meet the carrier-grade recovery requirement, even in a large-scale network serving many flows.

I. INTRODUCTION

A standard switch/router comes with the functionality to support all standard protocols. However, a switch uses only some of them. This makes the design extremely inefficient, complex, and expensive. Moreover, it constraints innovation opportunities for network operators, researchers and vendors. This problem is addressed by OpenFlow [1] by separating control and data plane. It removes the control plane functionality from the switches and embeds it into one or more servers called controllers, and hence makes switches inexpensive. In addition, it imparts network flexibility, as the control functionality is moved to the controllers, while only forwarding is required to be done in hardware. Any added functionality can be implemented seamlessly by programming the controllers.

OpenFlow [1] is developed in a clean-slate future internet program by Stanford University, which aims to offer a programmable network to test new protocols in current Internet platforms. The core idea of OpenFlow is to provide direct programming of a router or switch to monitor and modify the way in which packets are handled by the device. It is based on the fact that most modern routers/switches contain a proprietary FIB (Forwarding Information Base) which is implemented in the forwarding hardware using TCAMs (Ternary Content Addressable Memory). OpenFlow provides the concept of FlowTable that is an abstraction of the FIB. Additionally, it provides a protocol to program the FIB via adding/deleting/modifying entries in the FlowTable. This is achieved by one or more controllers that communicate with OpenFlow switches using the OpenFlow protocol (Fig. 1). The switch/router that exposes its FlowTables through the OpenFlow protocol is called an OpenFlow switch/router.

An entry in the FlowTable consists of: (1) a set of packet fields to match with incoming packets (called as flow), (2) statistics which keep track of matching packets per flow, and (3) actions which define how packets should be processed. When a packet arrives at an OpenFlow switch, it is compared with the Flow Entries in the FlowTable. If a match is found, the actions specified in the matching entry are performed. If no match is found, the packet (a part thereof) is forwarded to the controller. Thereafter, the controller makes a decision on how to handle the packet. It may return the packet to the switch indicating the forwarding port, or it may add a Flow Entry directing the switch on how to forward packets with the same flow.

One of the European projects named SPARC (SPlit ARchi-tecture Carrier-Grade Networks) [2] examined how networks with carrier-grade requirements can benefit from OpenFlow. These networks support hundreds of thousands of customers and require extremely high availability, while allowing a maximum disruption time of 50 ms [3]. A disruption of communication can cause a significant loss in activity, customers, and corresponding revenue. In fact, there is a service level agreement between the business customer and a service provider to deliver a high available service. If a network operator is unable to meet the agreement, it has to compensate for the loss of service.

To achieve carrier-grade quality in OpenFlow networks, both control and data traffic should recover from a failure within 50 ms. Failure recovery is important for both data and control traffic, as the loss in data traffic causes a disruption of service, while loss in control traffic prevents any new flow establishment from the switches affected by the failure.

To meet recovery requirements, we study restoration and protection for OpenFlow networks. OpenFlow networks can...
be in-band or out-of-band. In the case of an in-band network, control traffic (traffic to or from the controller) is sent on the same channel used to transport data traffic (Fig. 1A), whereas in the case of an out-of-band network, control traffic is sent on a different channel (Fig. 1B). The out-of-band network is expensive to build due to the requirement of an extra physical port on each switch. However, in the in-band network, there is no need of an extra physical port on each switch. Traffic to or from the controller can be directed via an OpenFlow network (see Fig. 1A). In [4], [5], [6], [7], we performed failure recovery experiments in the out-of-band OpenFlow network, where we emulated restoration and protection mechanisms for data traffic.

In this paper, we focus on failure recovery mechanisms for the in-band OpenFlow network, where we propose restoration and protection techniques for control traffic, while utilizing the previously proposed restoration and protection mechanisms of the out-of-band network for data traffic. We perform an extensive failure recovery experiment, and show that restoration does not recover traffic within 50 ms. Moreover, the restoration of control traffic delays the restoration of data traffic. The emulation results also show that protection for both control and data traffic can meet the carrier-grade recovery requirement, even in a large-scale network serving many flows.

The rest of the paper is organized as follows: Section 2 presents network recovery, Section 3 describes the emulation environment, Section 4 presents results, and finally section 5 concludes.

II. Network Recovery

Network recovery is a general term for any actions that enable a network to return to an operational state after a network failure. We first describe failure recovery mechanisms that are used in networks supporting carrier-grade quality. Then, we describe integration of those mechanisms in OpenFlow in order to recover within a ms interval.

For carrier-grade quality, failure recovery is generally achieved by first designing a network topology with failures in mind in order to provide alternate paths. The next step is adding the ability to detect failures and react to them using proper recovery mechanisms.

Loss of Signal (LOS) and Bidirectional Forwarding Detection (BFD) [9] are widely used to detect failures. LOS can detect failures in one particular port of the forwarding device, whereas BFD detects failures in the path between any two forwarding devices. BFD is a simple Hello protocol and in many aspects is similar to the detection components of many routing protocols such as OSPF (Open Shortest Path First). A pair of systems (end-to-end devices) transmits BFD packets periodically between each other, and if a system stops receiving BFD packets, the forwarding path between the neighboring systems is assumed to have failed.

After failure detection, traffic towards a faulty path is redirected to a fault-free path by using recovery mechanisms. Recovery mechanisms to deliver carrier-grade quality can be divided into two categories: restoration and protection [8]. In the case of restoration, recovery paths can be either pre-planned or dynamically allocated, but resources required by recovery paths are not allocated until a failure occurs. Thus, when the failure occurs, additional signaling is required to establish the restoration path. However, in the case of protection, recovery paths are always pre-planned and reserved before the failure occurs. Hence, when the failure occurs, no additional signaling is needed to establish the protection path; traffic can be immediately redirected. Restoration is therefore a reactive strategy, and protection is therefore a proactive strategy. There are different protection schemes depending on the number of recovery paths that are established for a given number of working paths. These are $1 + 1$, $1 : 1$, $1 : N (N > 1)$ and $M : N$ [8].

A. Failure recovery for OpenFlow networks

Failures can be detected in OpenFlow by LOS. It causes a port to change to the “down” state from the “up” state. When there is a change in the state of a port, an OpenFlow switch sends a notification message to the controller that is called a port-status message [10]. This mechanism detects only link-local failures and may be used in restoration. However, for path protection, end-to-end failure detection in any path in the forwarding switches is required. Therefore, we used BFD in the case of protection.

In [6], we focused on out-of-band OpenFlow networks, where we addressed the challenges of restoration and protection of only data traffic. For in-band networks, however, control traffic is sent along the same channel as data traffic, and thereby the restoration and protection mechanisms addressing both control and data traffic have to be investigated.

Nevertheless, OpenFlow describes recovery of control traffic. When control traffic of a switch is affected by a failure, the switch loses communication with the controller. The switch, then, tries to establish a new connection (e.g. TCP) with the controller after an echo request timeout [12]. If it fails to do so, it waits for a backoff timeout and again tries to establish a connection with the controller. The switch repeats the same process on each failure. As the minimum value of the echo request and the backoff timeouts are 1 second [12], failure recovery by this mechanism cannot be achieved in milliseconds. Therefore, we propose restoration and protection for fast recovery of control traffic in the in-band network.

The following subsections describe restoration and protection mechanisms for in-band OpenFlow networks.

1) Restoration of control traffic: The challenge behind implementing restoration of control traffic is that the controller has to establish a restoration path from the switches with which it has lost the connection. This can be explained with Fig. 1A. Assuming a failure occurs on link BC of the topology shown in Fig. 1A, the control traffic path between switch C and the controller is affected. Now the controller needs to establish the restoration paths $< SADC >$ and $< CDAS >$. Path $< SADC >$ is for the controller to switch C communication, and path $< CDAS >$ is for switch C to the controller communication. Establishment of new path means sending the necessary messages to modify or add the Flow Entries in switches. These messages are called flow-mod messages. As the connection between switch C and the controller is lost by the failure on link BC, switch C will not be able to receive the
flow-mod messages from the controller, which are required to establish the restoration paths \(< SADC >\) and \(< CDAS >\).

If the controller tries to send the flow-mod messages to switch C from switch D via link DC, it cannot be guaranteed that the full flow-mod message will be received by the affected switch C. This is because of segmentation done by the TCP protocol, which can segment a flow-mod message, leading to reception of only the first segment of the flow-mod message. However, since the new path \(< CDAS >\) is still not established, switch C still uses the faulty connection \(< CBAS >\) to send the acknowledgement of the first segment of the flow-mod message, which will not be received by the controller. And thereby, the controller will keep on retransmitting the first segment, and the full flow-mod message will never be received by switch C.

As the flow in the Flow Entry of switch A for the working path \(< SABC >\) and the restoration path \(< SADC >\) is identical but the action is different (i.e. to forward to switch B or D), the controller modifies the Flow Entry at switch A. In addition, as there is no Flow Entry installed in D in the initial setup for \(< SADC >\), the controller adds the corresponding Flow Entry in switch D. Since the incoming port of traffic is not assumed to be the part of the matching header in Flow Entries, the controller does not modify or add the Flow Entry in switch C for the restoration path \(< SADC >\). Furthermore, as switch C floods its own traffic, the controller does not need to modify the Flow Entry in switch C to establish the restoration path \(< CDAS >\).

Now for the restoration path \(< CDAS >\), the controller modifies the Flow Entry at switch D to forward control traffic of switch C to switch A (instead of dropping). For switch A, as the flow and the action of the Flow Entry for the restoration path \(< CDAS >\) and the faulty path \(< CBAS >\) are same, the controller does not modify the Flow Entry at switch A. After establishing the restoration paths \(< SADC >\) and \(< CDAS >\), the controller now completes the activity of failure recovery for switch C.

In the topology shown in Fig. 1A, if a failure occurs on link AB (instead of the link BC), the control traffic paths of two switches, switch B and switch C, are affected. We assume that the controller has established the working paths for switch B similar to switch C, cf. Fig. 2A. As restoration for switch B requires establishment of the restoration paths \(< SADCB >\) and \(< BCDAS >\) through switch C, the controller should first restore the control traffic of switch C before trying to restore switch B. However, the controller does not know when the restoration activity of switch C will complete. To know this, we use the barrier-request and reply messages concept, which is specified in OpenFlow specifications [10]. Therefore, the controller sends the barrier request messages to switch A and switch D after sending all flow-mod messages to them (restoration activity for switch C). After sending the barrier request messages, the controller waits to start the restoration activity of switch C until it receives the barrier-reply messages from switch A and switch D. These barrier-reply messages ensure that the restoration activity of switch C has completed.

### 2) Protection of control traffic

In the time between failure detection and the completion of restoration, traffic may be lost. In order to further reduce the traffic loss resulting from delay in the restoration action, we can turn to protection. Protection removes the need of the OpenFlow switches to contact the controller for modification and addition of the Flow Entries to establish the alternative path. This can be accomplished by pre-computing the protection path and establishing it together with the working path. In 1+1 protection, traffic is duplicated at both the working and the protection path, and in 1:1 protection, traffic is transmitted to the protection path upon the failure at the working path. Protection allows fast recovery but requires a larger FlowTable.

Our approach of implementing protection of the control traffic in in-band OpenFlow networks is similar to protection of data traffic in out-of-band OpenFlow networks.
In [6]). In this approach, the GroupTable concept specified for OpenFlow in its version 1.1 [11] is used to switch traffic to an alternative path. In the GroupTable concept, a Flow Entry in the FlowTable points packets to one of the Group Entries of the GroupTable. Each Group Entry consists of a number of action buckets. OpenFlow introduces the fast-failover type [11] of a Group Entry in order to switch traffic to an alternative path without needing to involve the controller. This group type is important for our protection mechanism. Any group entry of this type consists of two or more action buckets with a well-defined order. A bucket is considered alive if its associated alive status is within a specific range (i.e. watch port or watch group is not equal to 0xffffffff). The first action bucket describes what to do with the packet under normal conditions. If this action bucket has been declared as unavailable that is due to change in the status of the bucket (i.e 0xffffffff), the packet is treated according to the next available bucket. The status of the bucket can be changed by the monitored port going into the “down” state or through other mechanisms such as BFD. For protection, we used BFD for this purpose.

1:1 path protection for control traffic of switch C can be seen in Fig. 2B. In Fig. 2B, the working path <SABC> and the protection path <SADC> are established for the controller to switch C communication. In addition, for switch C to the controller communication, the working path <CBAS> and the protection path <CDAS> are established. The paths are established by adding the Flow Entries and the Group Entries in the switches. The OpenFlow switches A and C are the switches in Fig. 2B, which need to take the switching action on failure conditions. For switch A, the action is to send the packets generated by the controller to switch B during the normal condition and to switch D during the failure condition. For switch C, the action is to send the packets generated by it to switch B during the normal condition and to switch D during the failure condition. For these particular flows of packets in switch A and switch C, the GroupTable concept is applied.

The Group Entry in switch A and switch C, therefore, contains two action buckets: one for output port B and the other for output port D. Other than these entries, the corresponding Flow Entries can be added in the respective switches for the working and the protection paths. Once a failure is detected by BFD in switch A and switch C for the working paths <SABC> and <CBAS> respectively, switch A and switch C modify the alive status of the first bucket in the corresponding Group Entry. Thus, the action related to the next bucket, whose output port is D (in the case of switch A and switch C), can be taken. As the Flow Entries in switches D and C related to path <SADC>, and the Flow Entries in switches D and A related to path <CDAS> are already present, there is no need to install these entries upon failure.

3) Restoration and protection of data traffic: For restoration and protection of data traffic, we used the same mechanism as described in [6]. In the case of restoration, an alternative path is established by adding and modifying Flow Entries in the switches of the alternative path. However, in [6], failures in the control traffic are not considered. In order to establish the alternative path for data traffic, failure recovery of control traffic is important. This is because the alternative path cannot be established using the switches with which the controller has lost the connection. Therefore, before restoring data traffic, we established the control traffic paths between the controller and the affected switches of the alternative path.

For protection of data traffic, an alternative path is established before occurrence of a failure. When the failure occurs in the working path, the ingress OpenFlow switch redirects traffic to the alternative path by the GroupTable concept [11]. As the controller does not participate in recovery of data traffic, failure recovery in control traffic does not affect protection of data traffic.

III. EMULATION ENVIRONMENT

In this section, we describe testbed, topologies, and methodology used for experiments.

A. Emulation testbed and topologies

We performed emulation on our virtual-wall testbed which is a generic test environment for advanced network, distributive software and service evaluation. The testbed consists of 100 physical nodes interconnected by a non-blocking 1.5 Tb/s Force10 Ethernet switch. Each node in the testbed has 24 CPU cores and 24 GB RAM.

![Emulated Topology: German Backbone Network Topology](image)

Fig. 3. Emulated Topology: German Backbone Network Topology

![Integration of BFD in OpenFlow](image)

Fig. 4. Integration of BFD in OpenFlow
Mininet tool [13] can create topologies in a single node by using Linux process in different network namespace. We used this tool to create a German backbone network topology (shown in Fig. 3) in our testbed node. In Fig. 3, each switch of the German backbone topology is connected to a server, and the Hamburg switch is directly connected to the controller. To assign a separate CPU core to the controller and to each switch, we bounded a different core of our testbed node to each switch and to each switch of the topology.

B. Emulation methodology

In our emulation, each server generated data packets to all other servers in the topology shown in Fig. 3. Each server in our emulation sent data packets with five different flows to all other servers at the constant interval of 6 ms. There were a total of 910 flows in the network. To transmit the packets, we manually configured the routing table and the ARP table in each server.

There are many extensions of the OpenFlow protocol. Some of the extensions have been released publicly in the form of OpenFlow versions. In April 2012, the OpenFlow version 1.3 has been released by ONF (Open Networking Foundation). In addition, many OpenFlow controllers are also available for controlling OpenFlow networks. These are NOX, Beacon, Onix, Helios, Maestro and Floodlight. We implemented restoration and protection in the NOX controller and the OpenFlow version 1.1 (developed by Ericsson [14]), and used these for our emulation.

For detecting failures in the protection case, each working path was monitored by adding an additional BFD flow in the OpenFlow switches. The BFD process transmits a failure notification message when it detects a failure. To receive the failure notification message from the BFD process, a virtual link (the link veth1 - veth2 in Fig. 4) has been created between the OpenFlow switch process and the BFD process. Furthermore, an alias of the OpenFlow port (eth1:1) has been created for the BFD process to be able to receive the packets from the OpenFlow port (eth1). The BFD failure detection time in the experiment was between 40 to 44 ms. For the restoration case, we did not establish a BFD session. In this case, the OpenFlow switches detected a failure when LOS was declared by a port as a “port down” event.

For restoration and protection of both control and data traffic, we performed four different experiments:

1) Restoration experiment: In this experiment, restoration was performed on both control and data traffic.
2) Protection-Restoration experiment: In this experiment, protection was performed on control traffic and restoration was performed on data traffic.
3) Restoration-Protection experiment: In this experiment, restoration was performed on control traffic and protection was performed on data traffic.
4) Protection experiment: In this experiment, protection was performed on both control and data traffic.

In each of the above experiments, one of the links between the switches was broken and the recovery time was calculated.

We use the NOX traffic intensity in all the above experiments to illustrate our experimental setup. Fig. 5A, Fig. 5B, Fig. 5C and Fig. 5D show the NOX traffic intensity of the restoration, the protection-restoration, the restoration-protection, and the protection experiment respectively.

In our emulation, each switch in the German topology establishes first an OpenFlow session with the controller by the algorithm implemented in [15]. The traffic intensity shown at the beginning of the experiment (at the -200 to -168 seconds Fig. 5) is due to traffic generated to establish OpenFlow sessions with the controller. There is a large spike at -168 seconds in Fig. 5, shown by (1). This is the time in our emulation when the controller established control traffic paths for control traffic. The height of the spike shown by (1) is more in Fig. 5B and Fig. 5D than in Fig. 5A and Fig. 5C. This is because the protection paths were established together with the working paths in both the protection-restoration and the protection experiment.

At (2) in Fig. 5, there are 14 spikes (from -150 to -124 seconds) after each two-second interval. These spikes are due to data traffic from the switches to learn the path to the destination. The interval of two-second between the spikes occurred, because we have started sending data traffic waiting two-second between each server. The two-second interval was used to avoid overloading the NOX controller as the switches can try to establish too many flows in a short time span. These
spikes in Fig. 5C and Fig. 5D are higher than the spikes in Fig. 5A and Fig. 5B because protection of data traffic establishes an alternative path together with the working path.

At (3), there is no spike in Fig. 5A, there is one spike in Fig. 5B, and there are 14 spikes (from -100 to -74 seconds) in Fig. 5C and Fig. 5D. These are the spikes due to BFD traffic. As in the restoration experiment, neither control nor data traffic are protected, we did not establish any BFD session in this experiment. We, therefore, do not see any spike at (3) in Fig. 5A. In the case of the protection-restoration experiment, as control traffic needs to be protected on the failure condition, we established BFD sessions between Hamburg (c.f. Fig. 3) and all other switches. This is shown by the one large spike at (3) in Fig. 5B. However, as data traffic in the case of the restoration-protection experiment, and both data and control traffic in the case of protection experiment need to be protected, we established BFD session between each switch to all the other switches in these experiments. These are shown by 14 spikes at (3) in Fig. 5C and Fig. 5D.

There are also small spikes periodically in Fig. 5. These are due to the echo messages that were sent to check aliveness of the control traffic paths. The height and the number of these spikes are different in all the performed experiments. This was due to the minor time difference between the start of the experiments. At the 0 second (shown by (4)), we have failed the Hamburg-Berlin link (see Fig. 3) by disabling the Ethernet interface at Hamburg and Berlin. When the OpenFlow switches (Hamburg and Berlin) detected this failure, the port-status message was sent to the controller. Since the control traffic path of Berlin failed by the failure, the port-status of Berlin drops. However, when the port-status message of Hamburg reaches to the controller, the controller starts recovery actions. In the case of restoration, the controller sends the necessary messages to establish a new path. In the case of protection, the controller does not send any message to any switch to establish the new path on a failure condition because it has already established it before the failure occurred.

Sending messages to the switches give the spike at the controller traffic. As a result, because restoration is applied for both data and control traffic in the restoration experiment, the spike at 0 second is larger in Fig. 5A than in Fig. 5B and Fig. 5C. However, as protection is applied for both data and control traffic in the protection experiment, there is no spike at 0 second in Fig. 5D.

IV. EMULATION RESULTS

We now show the results of the experiments in which link Hamburg-Berlin was failed in the emulated German topology. Fig. 6 shows traffic at link Hamburg-Berlin (from -0.4 to 0.4 second), which was captured at the Hamburg switch. As the port at Hamburg and Berlin was disabled at 0 second, the Hamburg switch stopped transmitting/receiving packets at this link after 0 second. This is shown by Fig. 6.

Fig. 7 shows traffic on link Hamburg-Bremen (see Fig. 3). After link Hamburg-Berlin was taken down, this was the only link connecting Hamburg. Therefore, after failure, all traffic from and towards Hamburg must follow link Hamburg-Bremen. At the time of the link failure, we see a small drop in total traffic on this link. This was due to the loss in traffic which was coming from link Berlin-Hamburg over Hamburg-Bremen before the failure. This is shown from 0 to 0.050 second in Fig. 7A and Fig. 7B, from 0 to 0.040 second in Fig. 7C and 7D. After this drop, there is a sudden increase in total traffic at the link Bremen-Hamburg. In Fig. 7A and Fig. 7B, this was due to the restoration activity of the controller whereas, in Fig. 7C, this was due to the restoration and the protection activity, and in Fig. 7D, this was due to the protection activity.

In Fig. 7A, there is a small decrease in total traffic (0.050 to 0.070 second) after the sudden increase, followed by a stepwise increase (from 0.070 to 0.140 second) and a sudden decrease at 0.150 second. After 0.150 second, total traffic becomes approximately double to the amount it was before the failure. This was the time in the restoration experiment when the controller has completed the recovery action for both control and data traffic.

In Fig. 7B, after the sudden increase, there is a stepwise increase in traffic (from 0.050 to 0.120 second), followed by a sudden decrease at 0.130 second. After 0.130 second, total traffic becomes approximately double to the amount it was before failure. In Fig. 7A, this time is 0.150 second. Hence, in the restoration experiment (Fig. 7A), total recovery took 20 ms more time than the protection-restoration experiment (Fig. 7B). The only difference between these two experiments
is that in former, restoration is applied to control traffic and in latter, protection is applied to control traffic. Therefore, we can say that the restoration of control traffic in the restoration experiment has delayed the total recovery time by 20 ms.

In Fig. 7C, after the sudden increase, there is a sudden decrease in traffic at 0.080 second and then total traffic becomes approximately equal to double to the amount it was before the failure. However, in Fig. 7D, after sudden increase, total traffic becomes approximately double at about 0.050 second. Therefore, in the protection experiment, total recovery (both data and control traffic) took approximately 50 ms time.

Fig. 7 also shows the gap in traffic from Berlin when the failure occured in link Hamburg-Berlin. The gap shows disruption in data traffic at the time of the failure condition.

We performed link failure experiments on all the links indicated by a number in Fig. 3 (the number is the ID of the link). The results of the experiments are depicted in Fig. 8. The x-axis shows the ID of the broken links, the number of affected flows, the number of switches whose control traffic paths are affected by the failure (number of affected switches). The links in x-axis are ordered left to right according to the number of affected flows. The y-axis shows the failure detection time, and the recovery time for both control and data traffic. In the case of restoration of control traffic, the recovery time is calculated as the difference of the time when the failure was given and the controller received all barrier-reply messages for the restoration activity of all affected switches. In the case of restoration of data traffic, the recovery time is calculated as the difference of the time when the failure was given and all flow-mod messages were processed in the network. In the case of protection, the recovery time is calculated as the difference of the time when the failure was given and all affected Group Entries were modified.

Fig. 8A shows that the recovery time of data traffic does not vary linearly with the increased number of affected flows. This is because the number of affected switches (restoration of control traffic) has inserted additional delay to the recovery process. In the case of restoration of control traffic, the recovery time is calculated as the difference of the time when the failure was given and the controller received all barrier-reply messages for the restoration activity of all affected switches. In the case of restoration of data traffic, the recovery time is calculated as the difference of the time when the failure was given and all flow-mod messages were processed in the network. In the case of protection, the recovery time is calculated as the difference of the time when the failure was given and all affected Group Entries were modified.
maximum of 13 group entries (per switch) for all the flows in the network, and modification of these number of entries has taken less than 1 ms time (i.e. $O(n)$ time where $n$ is the number of affected group entries). Therefore, in our results, we see that the traffic in protection recovered immediately after detecting the failure. Hence, the traffic for which we performed protection, recovery met the carrier-grade requirement of 50 ms, and for the traffic we did not perform protection, recovery did not meet the carrier-grade requirement.

![Control traffic recovery time](image)

**Fig. 9.** Control traffic recovery time

The above experiments do not show how the restoration time of control traffic varies with the number of affected switches along the recovery path. The emulated German topology is a mesh topology in which it is difficult to have the increased number of affected switches in the recovery path. Therefore, to show the comparison, we performed a link failure experiment on the ring topology of 70 switches. The results of the experiment are shown in Fig. 9. The results show that the restoration time varies linearly with the increased number of affected switches along the recovery path. However, in the case of protection, we do not see dependence on the number of affected switches in the protection path (Fig. 9). It is approximately constant for all number of affected switches and meets the carrier-grade recovery requirement.

V. Conclusion

In this paper, we have presented restoration and protection for control and data traffic, and have performed failure recovery experiments in an in-band OpenFlow network. It has been shown that OpenFlow can restore traffic, but it does not allow achieving 50 ms restoration in a large-scale network, serving many flows. Moreover, the restoration of control traffic delays the restoration of data traffic. The emulation results also showed that protection for both control and data traffic can meet the 50 ms requirement, even in a large-scale network serving many flows.

In this paper, we did not consider situations where the controller or switches themselves crash. In future work, we will consider these situations, and will solve the controller crash problem by using two controllers. Hence, when one controller crashes, OpenFlow switches can rely on a backup controller to take actions.

Acknowledgment

The research leading to these results has received funding from the EU FP7 Programme under agreement n° 258365 (SPARC) and n° 258457 (OFELIA).

References