A Demonstration of Fast Failure Recovery in Software Defined Networking

Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet and Piet Demeester
Department of Information Technology
Ghent University-IBBT
Email: {firstname.lastname}@intec.ugent.be

Abstract. Software defined networking (SDN) is a recent architectural framework for networking, which aims at decoupling the network control plane from the physical topology and at having the forwarding element controlled through a uniform vendor-agnostic interface. A well-known implementation of SDN is OpenFlow. The core idea of OpenFlow is to provide direct programming of a router or switch to monitor and modify the way in which the individual packets are handled by the device. We describe our implemented fast failure recovery mechanisms (Restoration and Protection) in OpenFlow, capable of recovering from a link failure using an alternative path. In the demonstration, a video clip is streamed from a server to a remote client, which is connected by a network with an emulated German Backbone Network topology. We show switching of the video stream from the faulty path to the fault-free alternative path (restored or protected path) upon failure.

1. Introduction

Split architecture is a concept of decoupling the control functions from the forwarding elements and defining an open programmable interface between them. This split means that there are separated entities (physically) that remotely control several forwarding elements, which allows the independent design of control plane and leads to Software Defined Networking (SDN). One of the most known implementations of SDN is OpenFlow [1], which has gained significant interest from many research communities, and many of the research challenges behind it have been investigated in a number of projects all around the globe. In OpenFlow networks, one or more OpenFlow switches are controlled by separate devices (controllers) that communicate with the OpenFlow switches via the OpenFlow protocol. OpenFlow (specification 1.1 and beyond) provides the concept of FlowTables and a GroupTable [2], which is an abstraction of the Forwarding Information Base (FIB). We implemented fast failure recovery mechanisms in OpenFlow, capable of recovering from a link failure using an alternative path. We demonstrate the effectiveness of the implemented mechanisms by emulating a large scale German backbone network and achieving a recovery time of less than 50 ms.

2. Our implemented Mechanisms and Experiment on High Speed Testbed

One of the European projects named SPARC [3] studies how OpenFlow can be deployed in carrier-grade networks. The carrier-grade network should recover from the failure within 50 ms. We implement two well-known mechanisms of failure recovery i.e. restoration and protection in OpenFlow networks. In the case of restoration, the alternative path is established by the controller when it receives the failure notification from the OpenFlow switches [4,5]. In the case of protection, two disjoint paths (working and protected) are established by the controller before the failure occurs in the network. When the failure is detected in the working path, the traffic is switched to the protected path. We use a fast-failover type of the group-entry [2] to switch traffic between two different paths. This type is responsible for executing one of the action buckets of the group-entry as well as switching to another bucket upon failure. In our protection experiment, we establish an additional BFD session to monitor the failure in the working path. Once the BFD session stops receiving the BFD packets, the OpenFlow switch changes the associated alive-status.

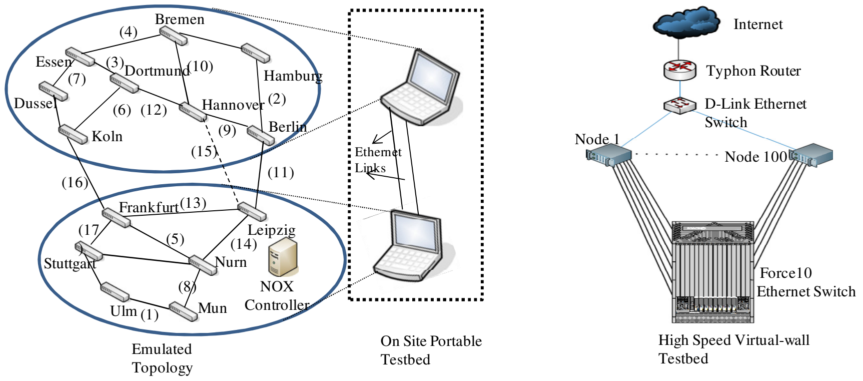

Fig. 2A: Emulated Topology and On Site Portable Testbed B: Virtual-wall Testbed

We emulated our recovery mechanisms. in a nationwide German network topology (Fig. 2A). Each of the switches in Fig. 2A is also connected to a server (not shown) and has a dedicated interface to a switched LAN which establishes connection with the controller. Our testbed where this emulation is carried out is shown in Fig. 2B.

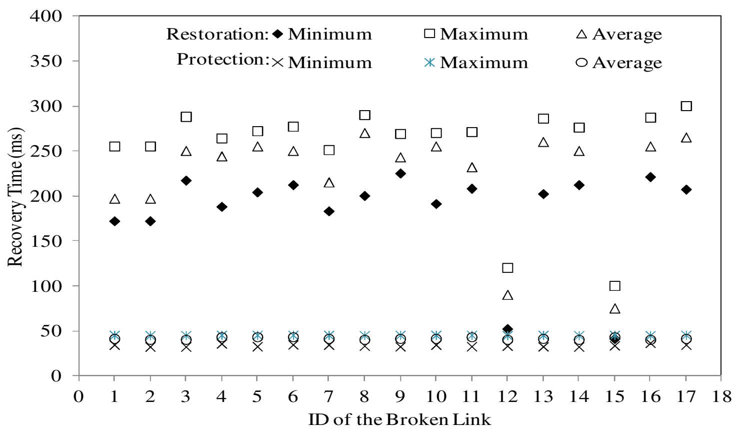

Fig. 3: Recovery Experiment

For the experimentation, we implement our recovery mechanisms in a NOX 1.1 controller and OpenFlow 1.1 software, recently developed by Ericsson [6]. We generate 182 different flows by using the Linux kernel module pktgen, break one link between switches and find recovery time. In our restoration experiment, the switches detect the failure due to loss of signal, which is approximately equal to the time when the first flow is restored in the network. On the other hand, the switches in the protection experiment detect the failure in 33 to 40 ms by establishing BFD sessions.

The results of the experiment are depicted in Fig. 3. In Fig. 3, the X-axis represents the broken link (the number in brackets of Fig. 2A).; the Y-axis represents the recovery time. The minimum value is the time it takes to recover the first flow; the maximum value is the time it takes to recover all the flows; and the average value is the expected time for any flow to be recovered after the failure. In our nodes, Hannover node where we break the link 12 or 15 (Fig. 2A) detects the failure within 50 ms in restoration. However, for other nodes, this value is between 167 to 210 ms (Fig. 3). The results show that restoration takes approximately 80 to 100 ms after detection of the failure (or after first flow is restored) and protection takes 1 to 3 ms.

3. Demonstration on Portable Testbed

In the on-site demonstration, we show our implemented restoration and protection in OpenFlow networks where a video server at one laptop streams a commercial video clip continuously, while the video client at other laptop receives and plays it in real time. We remove the link 15 (dotted link in Fig. 2A) from our emulated German topology to demonstrate it with two laptops, which consist of two Ethernet ports to communicate with each other. Mininet is used in the experiments to emulate this topology. We extend the Mininet software to send or receive the traffic from the physical port. For the demonstration, half of the topology is emulated in the first laptop and the other half is emulated in the second laptop (shown in Fig. 2A). In the topology, one NOX controller controls all the forwarding switches including switches emulated in the other laptop. Therefore, we use one Ethernet port for the working path; whereas other Ethernet port for the protected path and also for the communication between the controller and the switches of the other laptop. During the demonstration, we remove the Ethernet cable of the working path and show switching of traffic to the alternative path.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7) under grant agreement n° 258457 (SPARC) and n° 258365 (OFELIA).

References

1. N. McKeown et al., Openflow: Enabling innovation in campus networks, SIGCOMM, 2008.
2. OpenFlow Specification: Version 1.1.0: http://www.openflow.org/, 2011.
3. SPlit ARchitecture Carrier-Grade Networks (SPARC): http://www.fp7-sparc.eu/.
4. S.Sharma et. al., Enabling Fast Failure Recovery in OpenFlow Networks, DRCN, 2011.
5. D. Staessens et. al., Software Defined Networking: Meeting Carrier Grade Requirements, LANMAN, 2011.
6. Ericsson OpenFlow and NOX Controller Software: https://github.com/TrafficLab.