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 a 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 particular, large scale experimental testbeds are available for researchers in the US through the GENI project [2], in Japan through the JGN2plus research network [3], and in Europe through the OFELIA project [4]. Several algorithms resulting from experiments on the testbeds have been deployed in many networks supporting various applications. Currently industrial players like Deutsche Telekom, Google, Microsoft, Verizon, and Yahoo have shown a substantial interest towards OpenFlow and have formed Open Networking Foundation (ONF) to standardize the OpenFlow protocol.

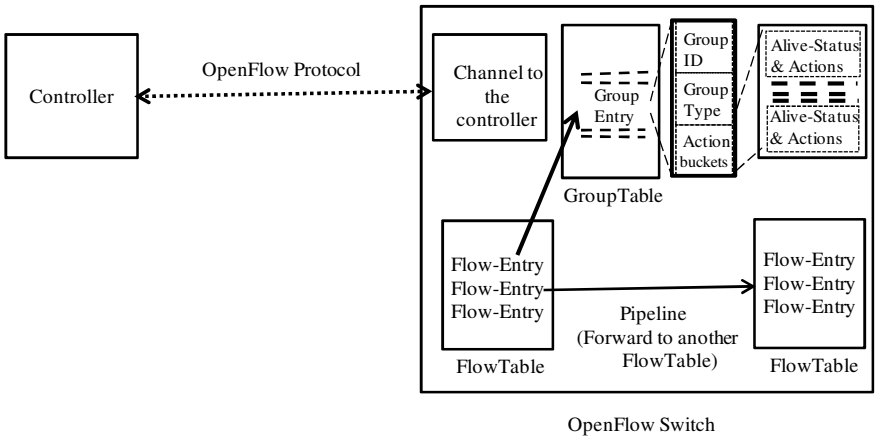

Fig. 1: OpenFlow Architecture

OpenFlow (specification 1.1 and beyond) provides the concept of FlowTables and a GroupTable [5], which is an abstraction of the Forwarding Information Base (FIB). In addition to this, it provides a protocol to program the FIB via "adding/deleting/modifying" entries in the tables. This is achieved by one or more separate devices (controllers) that communicate with the OpenFlow switches via the OpenFlow protocol (Fig. 1). A FlowTable in the switch contains a set of flow-entries. Each flow-entry consists of match fields, counters, and a set of instructions to apply to matching packets. Instructions associated with each flow-entry describe packet forwarding, group table processing, and pipeline processing (Fig. 1). When a packet arrives at the switch, it is compared against the flow-entries in the FlowTable. If a match is found, the instruction of that entry is executed. If no match is found, the packet (a part thereof) is forwarded to the controller; thereafter the controller determines how the packet should be handled. It may return the packet indicating the forwarding port or it may add a valid flow-entry in the switch. The group table in the switch contains group-entries; and each group-entry contains a list of action buckets (Fig. 1). The actions in one or more buckets are applied to packets sent to the group.

OpenFlow does not implement fast recovery mechanisms, which are necessary for reliable networks. 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.

1.2 Our implemented Recovery Mechanisms and Experiment on High Speed Testbed

One of the European projects named SPARC [6] studies how OpenFlow can be deployed in carrier-grade networks. Carrier-grade networks have a strict requirement of failure recovery. The 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 [7,8]. 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 [5] 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. The latter can be achieved by changing the value of an associated alive-status (See Fig. 1) to 0xfffffffff. In our protection experiment, we use BFD (Bidirectional Forwarding Detection) to change the value of alive-status. 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 of the specific bucket, following the same path as the BFD session.

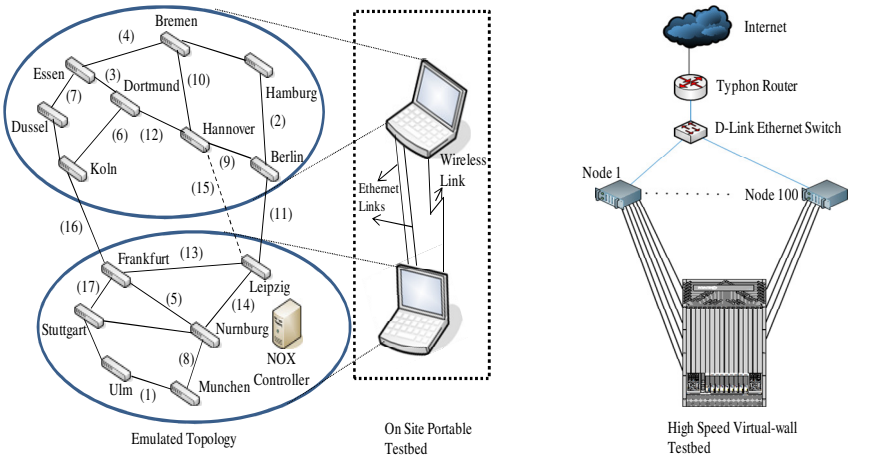

Fig. 2A: Emulated Topology and On Site Portable Testbed B: Virtual-wall Testbed

We emulated our recovery mechanisms. in a nationwide German network topology. This topology consists of 14 OpenFlow switches which are connected as shown in Fig. 2A. In addition to this, each of these switches is also connected to a server (not shown) and has a dedicated interface to a switched Ethernet LAN which establishes the connection (out-of-band) with the controller. The number in brackets of Fig. 2A represents ID of links that we break in our emulation. Our virtual-wall testbed, where this emulation is carried out, is a generic test environment (based on emulab) for advanced network, distributive software and service evaluation. It consists of 100 nodes (dual processor, dual core servers and up to six 1 Gb/s interfaces per node) interconnected by a 1.5 Tb/s Force10 switch (Fig. 2B). To allow arbitrary topologies and to provide users with security, it uses the concept of Virtual LANs.

For the experimentation, we implement our recovery mechanisms in a NOX 1.1 controller and OpenFlow 1.1 software, recently developed by Ericsson [9]. We generate a total of 182 different flows by using the Linux kernel module pktgen, break links one by one and find the 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.

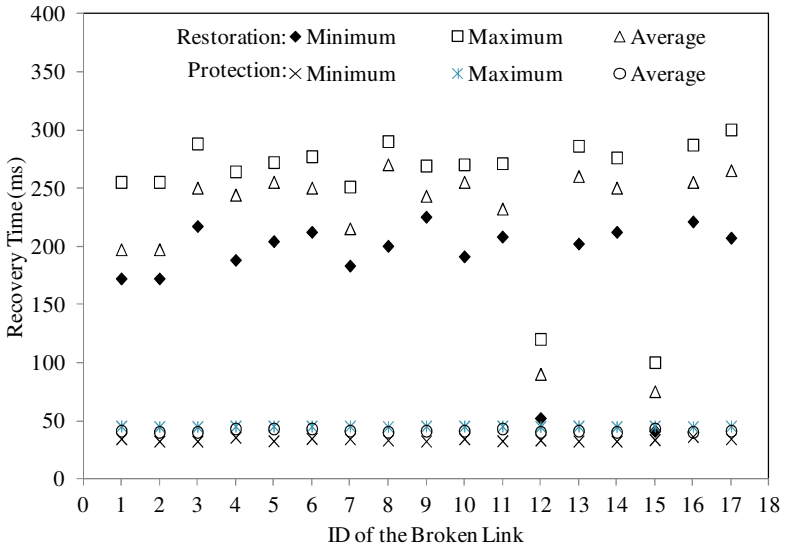

Fig. 3: Recovery Experiment

The results of the experiment are depicted in Fig. 3. In Fig. 3, the X-axis represents the broken links; 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. This is due to the fact that the first flow is recovered very fast when we break these links (12 or 15) in our experiment (Fig. 3). The results show that restoration takes approximately a time of 80 to 100 ms after detection of the failure (or after first flow is restored). On the other hand, protection takes 1 to 3 ms time. Since a real network is faster than our emulated testbed, our experiments show that protection can achieve carrier-grade requirement.

1.3 Demonstration on Portable Testbed

In the on-site demonstration, we show our implemented restoration and protection in OpenFlow networks where a video server at the source streams a commercial video clip continuously, while the video client at the destination 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 three physical ports (one wireless and two Ethernet ports) to communicate with each other. Mininet is used in the experiments to emulate this topology. Mininet is an emulation tool that constructs virtual OpenFlow topologies by creating virtual ports as well as establishing links between those ports. This tool sends or receives the traffic only from the virtual ports. 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 communication between the controller and the switches of the other laptop; whereas other Ethernet port and wireless port are used for the working path and the alternative path of the video traffic respectively. We also send the pktgen traffic from each server to all the other servers in our topology. Thus, the video traffic flows with all the 182 pktgen flows in the network. During the demonstration, we remove the Ethernet cable of the working path and show switching of traffic to the alternative path (restored or protected 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, Rev. 38(2), 69-74, 2008.
2. Global Environment for Network Innovations (GENI): http://www.geni.net3.
3. JG2plus research and development testbed network: http://jgn.nict.go.jp5.
4. OpenFlow in Europe: Linking Infrastructure and Applications (OFELIA): http://www.fp7-ofelia.eu/.
5. OpenFlow Specification: Version 1.1.0: http://www.openflow.org/, 2011.
6. SPlit ARchitecture Carrier-Grade Networks (SPARC): http://www.fp7-sparc.eu/.
7. S.Sharma et. al., Enabling Fast Failure Recovery in OpenFlow Networks, DRCN, 2011.
8. D. Staessens et. al., Software Defined Networking: Meeting Carrier Grade Requirements, LANMAN, 2011.
9. Ericsson OpenFlow and NOX Controller Software: https://github.com/TrafficLab.