A demonstration of automatic bootstrapping of resilient 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*—OpenFlow has disruptive potential in designing a flexible network that fosters innovation, reduces complexity, and delivers the right economics. The core idea of OpenFlow is to decouple the control plane functionality from switches, and to embed it into one or more servers called controllers. One of the challenges of OpenFlow is to deploy a network where control and data traffic are transmitted on the same channel. Implementing such a network is complex, since switches have to search and establish a path to the controller (bootstrapping) through the other switches in the network. We implemented this automatic bootstrapping of switches by using an algorithm where the controller establishes a path through the neighbor switches that are connected to it by the OpenFlow protocol. In the demonstration, we show this by using a GUI (Graphical User Interface) placed at the controller. Additionally, in the GUI, the OpenFlow switch topology gathered during bootstrapping is shown. During the demonstration, we insert a failure condition in one of the links in the topology and show failure recovery by a change in the GUI.**

I. Introduction

In a conventional router or switch, fast packet forwarding (data plane) and high level routing decisions (control plane) occur on the same device. OpenFlow [1] separates these two functions. The data plane portion resides on the same device, while the control plane portion is moved to one or more servers called controllers. Hence, direct programming of routers/switches is realised from the controllers. This makes routers/switches inexpensive and imparts network flexibility, as the control plane functionality is moved to the controllers, while only forwarding is required to be done in hardware.

OpenFlow 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 a 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.

At present, OpenFlow 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. We investigate two challenges of OpenFlow: the first one is to deploy an OpenFlow network where control and data traffic are transmitted on the same channel (in-band network), and the second one is to provide reliability in such an in-band network. Deploying the in-band network is complex, since switches have to search and establish a path to the controller (bootstrapping) through the other switches in the network. Therefore, we implemented a method that facilitates automatic bootstrapping of switches. In automatic bootstrapping, the controller establishes its own control network through the switches that are connected to it by the OpenFlow protocol.

For the reliability challenge in an in-band OpenFlow network, we implemented two well-known mechanisms of failure recovery: restoration and protection [2]. In the case of restoration, alternative paths are not established until a failure occurs. In the case of protection, alternative paths are established before the failure occurs in the network.

In the demonstration, we show automatic bootstrapping and failure recovery by using a GUI (Graphical User Interface) placed at the controller. A switch is shown in the GUI once it completes bootstrapping. Additionally, in the GUI, the OpenFlow switch topology gathered during bootstrapping is shown. During the demonstration, we insert a failure condition in one of the links in the topology and show failure recovery by using restoration and protection mechanisms.

II. Automatic bootstrapping and experiments on a high speed testbed

In this section, we introduce our bootstrapping approach, and present the conclusion of experiments performed on our testbed.

A. *Automatic bootstrapping of OpenFlow switches*

Bootstrapping of OpenFlow switches requires at least two steps:

1) Assignment of connection identifiers for connecting a switch to the controller. The identifiers required are at least the address of the switch and the address of the controller.
2) Instantiation of an OpenFlow session with the controller.

The first step can be accomplished by using a protocol such as DHCP (Dynamic Host Configuration Protocol) or OF-config (OpenFlow management and configuration protocol). These protocols can assign a unique address to each switch, and can allow it to know the address of the controller. We used DHCP for this purpose, and assumed that either the DHCP server is located in the controller node or it is the neighbor of the OpenFlow switch that is directly connected (physically) to the controller.

For bootstrapping, each switch runs a DHCP client and therefore keep on transmitting DHCP messages to its neighbors until it receives a reply from the DHCP server. If a neighbor is the DHCP server, it replies to the switches. In the case the neighbor is a switch connected to the controller by the OpenFlow protocol, the controller allows the switch to forward messages to the DHCP server. In the case the neighbor switch is not connected to the controller, the messages are dropped.

Once a switch has an address and it knows the address of the controller (i.e. after the first step), it establishes an OpenFlow session with the controller. The switch is able to establish the session in at least one of the following cases: (1) the controller is directly connected (physically) to the switch, (2) a neighbor switch has an OpenFlow session with the controller. Bootstrapping of an OpenFlow network completes when each switch in the network has an OpenFlow session with the controller.

During bootstrapping, the controller also builds the topology in its database by parsing the source MAC address of the DHCP messages, and by flooding a special kind of probe message from the recently connected switches and then by receiving the same probe messages from the neighbor switches. The controller uses this topology to find a path to any switch and to the DHCP server during bootstrapping.

B. *Experiments on a high speed testbed*

We performed bootstrapping and failure recovery experiments on our virtual-wall testbed which is a generic test environment for advanced network, distributive software and service evaluation. We measured suitability of the implemented bootstrapping mechanism by performing experiments on different types of topologies: linear, ring, star and mesh topologies. The experimental results showed that OpenFlow switches can be bootstrapped in a network where control and data traffic are transmitted on the same channel (in-band network). With our bootstrapping approach, we found a linear relationship between the bootstrapping time of a switch and the shortest distance (number of hops) between the switch and the controller.

For failure recovery, we implemented restoration and protection to achieve carrier-grade quality in in-band OpenFlow networks [2]. To achieve carrier-grade quality, a network should be able to recover from a failure within 50 ms. With restoration experiments, we conclude that Openflow can restore traffic, but its dependency on the controller means that it will be hard to achieve 50 ms restoration in a large-scale network. With protection experiments, we conclude that protection is a way in OpenFlow to achieve carrier-grade recovery requirements, even in a large-scale network serving many flows [2], [3].

III. Demonstration

We demonstrate our proposed bootstrapping and failure recovery mechanisms for an in-band OpenFlow network. For the demonstration, we create an OpenFlow network by generating a German backbone topology on two laptops (Fig. 1). The laptops in our demonstration are connected by two Ethernet cables, shown in Fig. 1. We generate half of the topology on the first laptop and the other half on the second laptop. The topology is generated by using Linux processes in different network namespaces. In the demonstration, the DHCP server

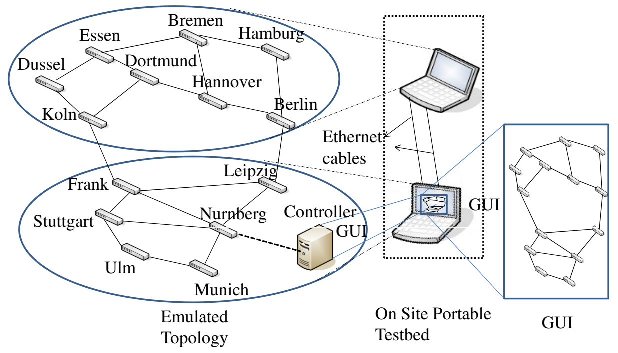

Fig. 1. The demonstration network topology, including the on-site portable testbed and the GUI for the demonstration

is located on the controller node, and one controller connected to the Nurnberg switch controls all the forwarding switches including the switches on the other laptop. A GUI is shown in Fig. 1. When a switch completes bootstrapping, it is shown in the GUI. The GUI in the demonstration also shows the OpenFlow topology gathered during bootstrapping.

In the demonstration, we show both restoration and protection in the emulated German backbone topology by performing two different experiments on the same laptops. During the demonstration, we pull out one of the cables between the laptops and show that the topology changes in the GUI.

With our protection mechanism, failure recovery happens before the controller detects the failure. This is because switches in our protection mechanism use the concept of a GroupTable [4] to redirect the traffic from the working path to the protection path without contacting the controller. Therefore, in the case of protection, the GUI does not show disappearance of the switches after the failure condition. However, in the case of restoration, the controller starts recovery after detecting a failure and recovery itself takes some time after the failure detection. Therefore, in the case of restoration, the GUI shows disappearance of switches for a very small interval.

Acknowledgment

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

References

[1] N. McKeown, T. Andershnan, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, Openflow: Enabling innovation in campus networks, ACM Computer Communication Review, 2008.

[2] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, Fast failure recovery for in-band OpenFlow networks, DRCN, 2013.

[3] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, OpenFlow: Meeting carrier-grade recovery requirements, Computer Communications, 2012.

[4] OpenFlow Switch Specification: Version 1.1.0 (Wire Protocol 0x02): www.openflow.org/documents/openflow-spec-v1.1.0.pdf.