Network Virtualization TP2 - Network Managment

Lin I-Ping[pg41017], Erikson Tomas[pg44102], Eduardo Cunha[pg45515], and Miguel Peixoto[pg45517]

Engineering of Computer Networks and Telematic Services, School of Engineering, University of Minho, Braga, Portugal

Abstract. This work aims to refine our knowledge of the OpenFlow protocol and SDN architecture. The focus is the development of a routing application and a firewall on top of the SDN Ryu controller.

Keywords: Network Virtualization · Software Defined Networks · SDN · OpenFlow · OVSwitch · Ryu · Firewall.

1 Introduction

This work aims to implement two more applications on the application layer of the SDN architecture. The first application is the development of a routing application. It means the application must be able to calculate the best paths among the subnets and apply them to the L3 switches. Moreover, it must be able to recognize changes in the topology the changes must be reflected in the entries in each L3 switch.

The second application is a firewall that must block traffic among networks and allow HTTP traffic in a specific port between specific hosts and the HTTP server. Both applications are going to be tested in the same network topology - Figure 1.

To develop this work a Fedora 34 (kernel 5.11) virtual machine was created. To emulate the network the Mininet (version 2.3.0) is used. Among the two options proposed by the professor - Ryu or FloodLight - the Ryu (version 4.34) was chosen by the group due to the knowledge of the Python language among the elements of the group. A Python IDE was used to create and debug the scripts, in our case the Visual Studio Code (version 1.65.0). The OpenFlow version must be 1.3.

2 Exercise 1

In SDN networks the controllers have a complete view of the network topology. I mean that all the links and paths are known. So our work is to understand how to access the data necessary, calculate the best paths and inject the routes into the switches.

It is possible to monitor changes in the network topology, for example: if some link is down, it is possible to recognize it. So it is necessary to recalculate the best paths again and add the new correspondents' entries into the L3 switches flow table.

The topology is represented in Figure 1 and the script that represents it is attached with this report in case wish to check it in more detail.

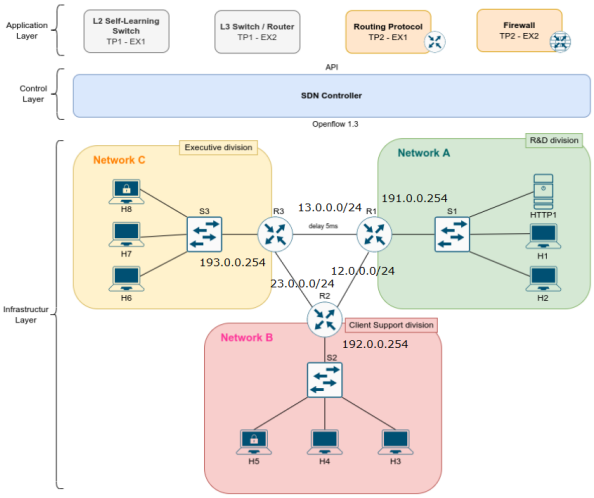

Fig. 1. Network Topology

2.1 Strategy and Implementation

The approach used in this work came due to a lot of effort from the members of the group and a lot of research. To keep the explanation as organized and simple as possible the strategy is divided into two parts: tracks changes in topology and calculation and injection of the best path into the switches flow tables.

Track changes in topology: To track the changes in topology, our group used as base the dumper.py file that comes with the Ryu Controller[4]. This file has a set of functions that follow the changes in the topology. The used functions are the following ones:

- **switch_enter_handler**: Triggered when a new switch enters in topology;
- **switch_leave_handler**: Triggered when a switch leaves topology;
- **port_add_handler**: Triggered when a new port is added to any switch;
- **port_delete_handler**: Triggered when a port is deleted in any switch;

- **port_modify_handler**: Triggered when any port in any switch change the status;
- **link_add_handler**: Triggered when new link are created;
- **link_del_handler**: Triggered when any link is deleted;

All these functions were imported to the L3Switch class and some additions were made. Firstly, our group thought about writing all the events in a file and then developing an application to calculate the best paths. However, during the process, the approach changed and the NetworkX [3] library was used to it.

The NetworkX is a powerful library that has a built-in function to construct graphs, create links and calculate the best paths from sources to destinations. As one of the integrates of the group had previous knowledge it was quite easy to track the topology changes in the network using this approach.

To implement it a few more attributes were need to be created in the object of the L3Switch class, these attributes are:

- **topology_api_app**: Needed to get switch and link information in Ryu controller API
- **links**: Holds all the links among the nodes;
- **best_paths**: Holds the best path among all pairs of nodes in topology;
- **int_port_to_ip**: A deep copy of *interface_port_to_ip* that changes based on the events;
- **topology**: NetworkX object composed by nodes and edges/links.

Inside each function reused from the dumper.py file, changes are applied to the topology attribute. For example: When a switch enters the topology, a new node is added to the NetworkX topology, then when a new link is created between switches, the **link_dd_handler** is triggered, and then a new edge is added between the correspondent nodes.

The subnets that belong to each switch are saved in the NetworX object as a dictionary attribute that changes dynamically as well. For example, if the port of network *191.0.0.0/24* that belongs to R1 is changed to down, then this correspondent network is deleted from the attribute of the node in the NetworkX topology.

Following this logical approach to each event used from the dumper.py file track changes in topology were simple and dynamically because each event represents a change in the NetworkX object that represents our topology. For more details

Calculation and injection in flow tables Once the topology is always being tracked, the next step is to calculate the best path to each network and then add all the necessary entries in the flow table of each switch in topology to allow the packets to flow.

The NetworkX has a build-in function called **all_pairs_shortest_path** that is used to calculate all the shortest paths among all the L3 switches. That routine is used in another function created by us that checks all the sources and destinations and applies the flow in each L3 switch - named **inject_best_paths**.

When any change is detected in topology, the inject_best_paths is called and then the new flows are sent to all the switches again. The code is not functional. Our first idea was to delete all the flows of the inject_best_paths and apply the new ones, but it was not working. So our approach to overcoming this problem has a possible solution: create a table with all the rules as an attribute of the class and delete only the ones that are invalids when any changes occur, using cookies to identify each entry. Our group tried to implement it but the deadline was too close, then we do not implement it.

Meanwhile, when a change occurs, our code prints the change showing that the track of the changes is working perfectly. Our code applied the new rules as well, but because the old entries are not being deleted from the flow tables it does not work in some cases as expected.

All our work is available on GitHub[1] and our group strongly advises readers to take a chance to look at all the codes carefully to understand in more detail all the functions and may try to propose improvements.

2.2 Dry-run

The first step is to ensure that we don't have any other containers running on the system:

```
sudo mn --clean
```

Listing 1.1. Cleaning system environment

With this command, Mininet will stop any running controller and release any network interface used before.

The second step is to start the Ryu controller on port 6633 for layer 2 switches invoking ryu-manager with our previous exercise code:

```
ryu-manager ryu.app.simple_switch_13 --ofp-tcp-listen-port
    6633
```

Listing 1.2. Starting Ryu controller

The third step is to start the Ryu controller on port 6655 for the layer 3 switch invoking ryu-manager with our new exercise code. The **- -observe-links** is necessary to activate the controller to observe the events in topology[2]:

```
ryu-manager ryu.app.tp2_controller --observe-links --ofp-tcp-
    listen-port 6655
```

Listing 1.3. Starting Ryu controller

Then we are ready to start the Mininet with our topology with the command:

```
sudo python3 mininet/custom/tp2_topology.py
```

Listing 1.4. Starting Mininet

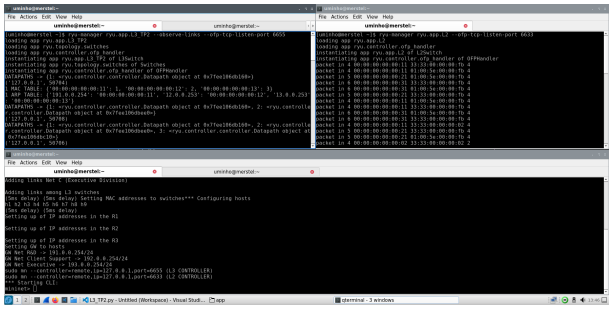

Fig. 2. Starting exercise 1

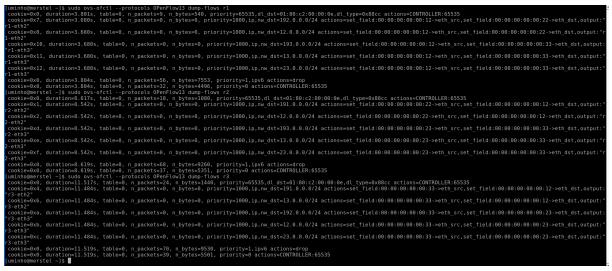

Fig. 3. Flow table L3 switches - exercise 1

2.3 Tests

Starting the controllers and the topology scripts - figure2 - and checking the flow table to all the L3 switches before run any ping test - figure 3.

Then, let's run the **pingall** command to test connectivity among all the hosts in the topology - figure 4. All the hosts have connectivity, as expected.

To simulate a change in topology, the command **link** is used to change the status of the link between **R1 - R3** to **down**. Looking at the console window of the L3 switches controller in the top-left of figure 5, it is possible to see that the last line in the output shows the current state of the topology in the NetworkX object and the link ports that link R1 and R3 is out of the dictionary for both nodes. The dictionary of the dictionary must be read in this format: {NODE:{PORT: IP_ADDR}}

Now, check the flow tables for R1 and R3. As the tables show, the new entries are correct. Moreover, because we faced a problem deleting the invalid entries from the table when we try to run the pingall test again, the ping does not work from NET A to NET C because the packet matches an invalid flow.

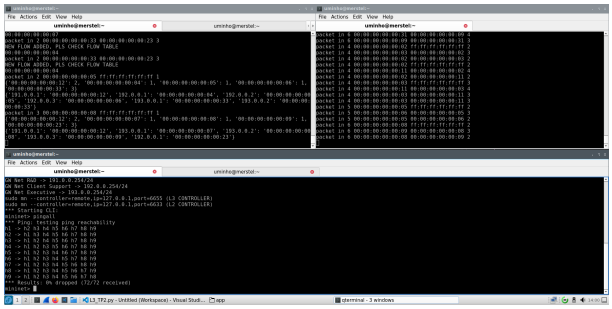

Fig. 4. Ping test to among all hosts - exercise 1

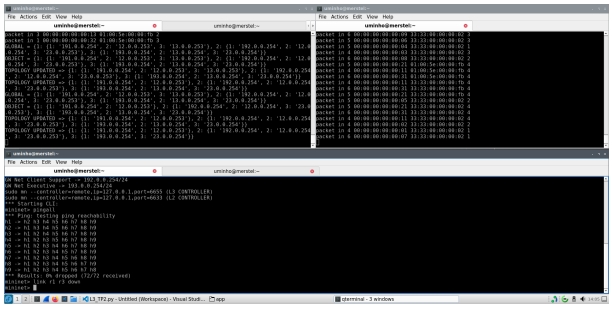

Fig. 5. Change of status of link between R1 and R3 - exercise 1

3 Exercise 2

The last exercise consists in implement a firewall that blocks traffic between networks but HTTP traffic must flow from H5 - H3 (HTTP server) and from H8 - H3.

3.1 Strategy and Implementation

Implement the firewall in SDN because is necessary only to know how to play with the flows and priorities. Our approach was to inject two types of flow entries into the switches. First, a flow to block all the traffic between networks, and then a flow with a higher priority allowing TCP on a specific port between two specific IPv4 addresses - the addresses of the hosts.

So, it was defined as a routine that is called inside the **packet_in_handler** that applies the flows needed to block the traffic between networks and allow TCP traffics on port 5555 - in our case. The priorities defined are high because it is supposed to have priority over the other flows. For block traffic between networks, the priority is 64000, and to allow HTTP traffic is 65000.

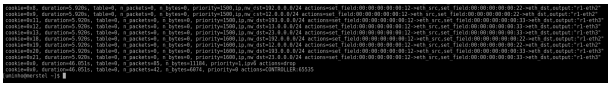

Fig. 6. Flow table R1 with port down - exercise 1

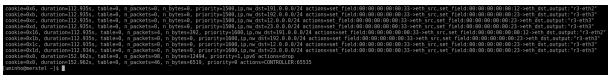

Fig. 7. Flow table R3 with port down - exercise 1

3.2 Dry-run

The first step is to ensure that we don't have any other containers running on the system:

```
sudo mn --clean
```

Listing 1.5. Cleaning system environment

With this command, Mininet will stop any running controller and release any network interface used before.

The second step is to start the Ryu controller on port 6633 for layer 2 switches invoking ryu-manager with our previous exercise code:

```
ryu-manager ryu.app.simple_switch_13 --ofp-tcp-listen-port
    6633
```

Listing 1.6. Starting Ryu controller

The third step is to start the Ryu controller on port 6655 for the layer 3 switch invoking ryu-manager with our new exercise code. The **- -observe-links** is necessary to activate the controller to observe the events in topology[2]. It is necessary to uncomment line 165 of the code to activate the firewall:

```
ryu-manager ryu.app.tp2_controller --observe-links --ofp-tcp-
    listen-port 6655
```

Listing 1.7. Starting Ryu controller

Then we are ready to start the Mininet with our topology with the command:

```
sudo python3 mininet/custom/tp2_topology.py
```

Listing 1.8. Starting Mininet

3.3 Tests

Running the **pingall** test 8 it is clear that only traffic inside the same network is allowed. Looking at the flow tables of R1 9 as an example lines 2-3 of the flow table allow traffic HTTP on port 5555 and the 4 line only allows traffic within the local network.

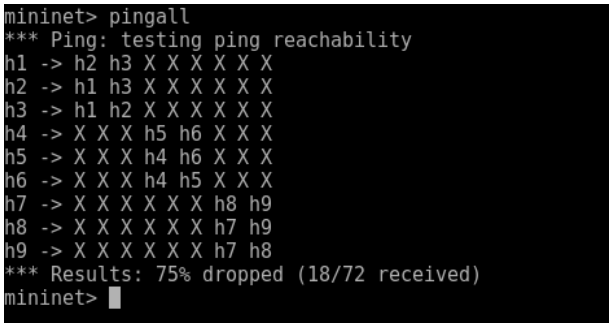

Fig. 8. Pin test with firewall activate - exercise 2

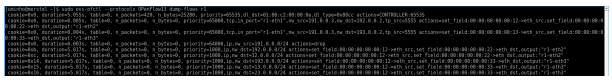

Fig. 9. Pin test with firewall activate - exercise 2

Opening xterm for hosts H5 and H8 and then initializing the HTTP server on H3. Using the command **wget** it is possible to test if HTTP traffic is flowing from H3 to host H5 and H8 - figure 10.

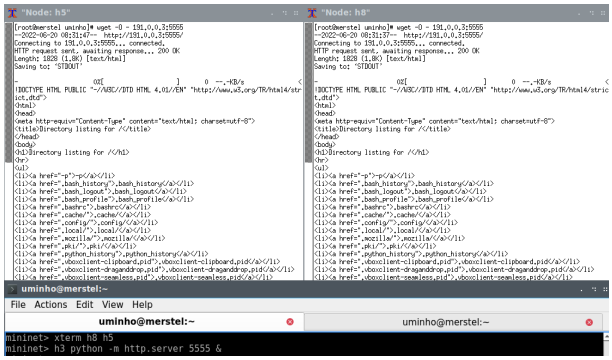

Fig. 10. Testing HTTP on port 5555 - exercise 2

4 Conclusion

This TP was arduous but interesting. Our group is really satisfied with our results, especially because we had to overcome a lot of challenges during this TP.

The routing application must be improved. We think that we have all the skills to implement it. However, the deadline was coming and we thought that it was better to implement it as we did than be stalled in exercise 1 and do not implement the firewall and do not do TP3.

In general, the members of our group learned a lot about SDN and everybody feels comfortable accepting new challenges to work with it shortly. Another important aspect is that sometimes we tend to try to compare the SDN with traditional networks, but it is impossible because SDN brings a lot of abstraction. We think that a lot of the time spent learning about it was much more about understanding the concepts. Once the concepts were learned and fixed, it was quite easy to harvest the results.

References

[1] Thomas E. Cunha E. Peixoto M. and Lin I. *Network Virtualization repository*. `https://github.com/pg45517/VR`. 2022 (cit. on p. 4).

[2] Castro F. *Topology Discovery with Ryu*. Accessed on 17-06-2022. 2014. URL: `https://sdn-lab.com/2014/12/31/topology-discovery-with-ryu/` (cit. on pp. 4, 7).

[3] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15 (cit. on p. 3).

[4] Ashwin Prabhu. *Dumper.py file of RYU controller*. Accessed on 10-06-2022. Youtube. 2014. URL: `https://www.youtube.com/watch?v=85NNiHRhnZA` (cit. on p. 2).